Etude et implémentation sur SoC-FPGA d'une méthode probabiliste pour le contrôle de mission de véhicule autonome

Chabha Hireche

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UBO
Université de Bretagne Occidentale

THESE DE DOCTORAT DE

Projet HPeC n°15-CE24-0022-01

L'UNIVERSITE
DE BRETAGNE OCCIDENTALE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Chabha HIRECHE

« Étude et implémentation sur SoC-FPGA d'une méthode probabiliste pour le contrôle de mission de véhicule autonome. »

Thèse présentée et soutenue à UBO, Brest, le 29 Novembre 2019
Unité de recherche : Lab-STICC, UMR 6285

Rapporteurs avant soutenance :

Smail NIAR	Professeur des universités, Université Polytechnique Hauts-de-France
Elodie CHANTHERY	Maître de conférences HDR, INSA Toulouse

Composition du Jury :

Président :	Benoît MIRAMOND	Professeur des universités, Université de Côte d'Azur Polytech Nice Sophia Antipolis
Examinateurs:	Stéphane MOCANU	Maître de conférences, INP Grenoble
Rapporteurs :	Smail NIAR	Professeur des universités, Université Polytechnique Hauts-de-France
	Elodie CHANTHERY	Maître de conférences HDR, INSA Toulouse
Dir. de thèse :	Jean-Philippe DIGUET	Directeur de recherche CNRS, Lab-STICC
Co-encadrante :	Catherine DEZAN	Maître de conférences, Université de Bretagne Occidentale

Remerciements

Je tiens à remercier les personnes qui ont contribué à la réussite de cette thèse.

Tout d'abord, Catherine Dezan et Jean-Philippe Diguet, qui ont su assurer l'encadrement avec brillance et bienveillance tout au long de ces années de thèse.

Je remercie également chaque membre du jury, Élodie Chanthery et Smail Niar qui ont accepté d'être rapporteur de ma thèse, leurs remarques ont été constructives pour mieux présenter mes travaux de thèse, Benoît Miramond d'avoir accepté de présider le jury et d'évaluer mon travail, Stéphane Mocanu, d'avoir accepté d'évaluer mon travail et pour sa participation scientifique. Je remercie donc tous les membres du jury d'avoir accepté d'évaluer mon travail et de me donner l'opportunité de le défendre.

J'adresse des remerciements à ma famille et ami(e)s qui ont su m'encourager et me soutenir durant toutes ces années de thèse.

Résumé

Les systèmes autonomes embarquent différents types de capteurs, d'applications et de calculateurs puissants. Ils sont utilisés dans différents domaines d'application et exécutent diverses missions simples ou complexes. Ces missions se déroulent le plus souvent dans des environnements non déterministes avec des événements aléatoires pouvant perturber l'exécution de la mission. Il est donc nécessaire d'évaluer régulièrement l'état de santé du système et de ses composants dans le but de détecter les défaillances. Par la suite, une décision est prise par le planificateur de mission permettant de poursuivre l'exécution de la mission en réponse à l'événement détecté. Cette décision doit considérer plusieurs contraintes comme l'objectif de la mission, l'état de santé des capteurs et des applications embarqués, la stratégie de déroulement de la mission 'safety' ou 'mission first', etc. D'un autre côté, les systèmes autonomes exécutent différentes tâches qui demandent différentes performances. Il est donc nécessaire de penser à l'utilisation d'accélérateurs matériels pour répondre aux contraintes de calculs hautes performances et aussi pour décharger le CPU si besoin.

Cette thèse répond à trois objectifs. Le premier objectif consiste à identifier les différents événements aléatoires qui peuvent faire échouer ou dégrader la mission du système autonome. Pour cela, nous utilisons les réseaux Bayésiens (Bayesian Networks - BN) pour le diagnostic dans le but d'évaluer l'état de santé du système, capteurs et surtout la qualité de service des applications embarquées qui est souvent négligée. Cette analyse prend en compte les différentes observations sur les capteurs moniteurs et contextes d'apparition des erreurs et propose une solution algorithmique pour atténuer l'erreur observée. Le second objectif est de proposer un modèle de décision probabiliste qui peut considérer à la fois les contraintes et les aléas de la mission. Ce modèle repose sur le Processus de Décision Markovien (MDP) et y associe un BN pour former le planificateur de mission ayant comme objectif la génération de la décision optimale. Et pour le troisième objectif, nous proposons une implémentation matérielle/logicielle embarquée sur FPGA du planificateur de mission afin d'accélérer les calculs et décharger le CPU.

Mots-clés— Planificateur de mission, Décision, Diagnostic, Markov Decision Process, BN, FPGA, Synthèse de haut niveau, Implémentation matérielle/logicielle.

Abstract

Autonomous systems, like UAVs, embed different types of sensors, applications and powerful calculators. They are used in different fields of application and perform different simple or complex missions. These missions usually operate in non-deterministic environments with random events that can disturb the execution of the mission or even lead to its failure. Therefore, it is necessary to regularly check the health of the system and its hardware (sensors) and software (applications) components in order to detect failures. Then, a decision is made by the mission planning to continue the mission in answer to the detected event. This decision must consider several constraints such as the objective of the mission, the health status of the sensors and embedded applications (es), the mission policy "safety" or "mission first", etc. On the other hand, autonomous systems perform different tasks that require different performance. Therefore, it is necessary to consider the use of hardware accelerators to address high-performance computing constraints and also to unload the CPU if nedded.

This thesis attempts to meet three objectives. The first objective is to identify and detect the different random events that can fail or degrade the mission of the autonomous system. So, we use Bayesian Networks for diagnosis in order to evaluate the health status of the system, sensors and especially the quality of service of embedded applications which is often neglected. We proposed a generic BN model based on a FMEA (Failure Mode and Effects Analysis) analysis. This analysis takes into account the different observations on the monitor sensors, context of the error and proposes an algorithmic solution to mitigate the observed error. The second objective is to propose a probabilistic decision-making model that can consider both the constraints and hazards of the mission. We used the Markov Decision Process for this purpose. The two models, BNs and MDPs, form the mission planner with the objective of producing the most optimal decision. Finally, we propose an embedded hardware/software implementation on FPGA to accelerate calculations and unload the CPU.

Keywords— Mission planning, Decision Making, Diagnosis, Markov Decision Process, Bayesian Networks, FMEA, FPGA, High Level Synthesis, HW/SW Implementation.

Table des matières

- Chapitre 1 -

Introduction générale

Sommaire

1.1 Contexte et problématique

Avec le progrès technologique apporté aux niveaux capteurs et puissance des calculateurs embarqués, les systèmes autonomes sont de plus en plus utilisés, et cela dans différents domaines d'application (défense, robotique, médical, etc.). Les systèmes autonomes sont donc amenés à réaliser différentes tâches et missions. Ces missions peuvent être simples ou complexes telles que la navigation, la détection d'objet, la poursuite de cible, l'exploration et la surveillance d'un environnement donné, etc. Souvent les missions assignées aux systèmes autonomes (comme les robots) se déroulent dans un environnement incertain avec des événements aléatoires. Cela peut donc engendrer des scénarios de défaillance, tels que l'apparition d'événements environnementaux aléatoires (obstacle, etc.), les défaillances matérielles ou logicielles (panne de capteur), etc. Un dysfonctionnement peut entraîner des dégâts plus au moins importants selon le type et la gravité de la défaillance apparue. Pour limiter les chances que cela arrive et éviter des dégâts potentiels, il est nécessaire d'évaluer, continuellement, l'état de santé du système et de ses composants matériels et logiciels. Cette évaluation continuelle de l'état du système conduit à l'adaptation du plan de la mission en fonction des observations faites sur l'environnement et le système lui-même. Par conséquent, le système doit prendre des décisions (planification et re-configuration) qui maximisent la capacité à répondre à l'objectif de la mission.

Le sujet de cette thèse s'inscrit dans le cadre d'un projet ANR HPeC (ANR 2015-2019). Le projet HPeC a pour objectif de montrer la capacité des systèmes auto-adaptatifs à faire face aux demandes de calcul haute performance des systèmes autonomes. Le projet HPeC considère un cas d'étude qui est l'exécution d'une mission de tracking par un drone. Afin de pouvoir gérer la mission du drone, le projet HPeC propose une solution qui repose sur trois différents niveaux (Figure 1.1) :

1. Niveau mission : proposer et mettre en œuvre une solution pour la planification de mission. Ce planificateur de mission a pour but d'établir des plans de mission pour faire face aux différents événements qui apparaîtront durant l'exécution de la mission.
2. Niveau configuration : ordonnancer les séquences d'applications ou versions d'applications décidées par le niveau mission. Cet ordonnancement a pour but de configurer dynamiquement l'exécution des applications sur la carte embarquée SoC-FPGA.
3. Niveau architecture : propose de configurer de manière dynamique une architecture SoC-FPGA, au sein d'une carte embarquée HPeC sur laquelle s'exécuteront les applications embarquées sur le drone.

Les travaux présentés dans cette thèse font partie du niveau mission du projet HPeC. Nous exposons un modèle de planification de mission qui repose sur deux aspects : diagnostic et décision. Le diagnostic peut se faire à l'aide de divers modèles

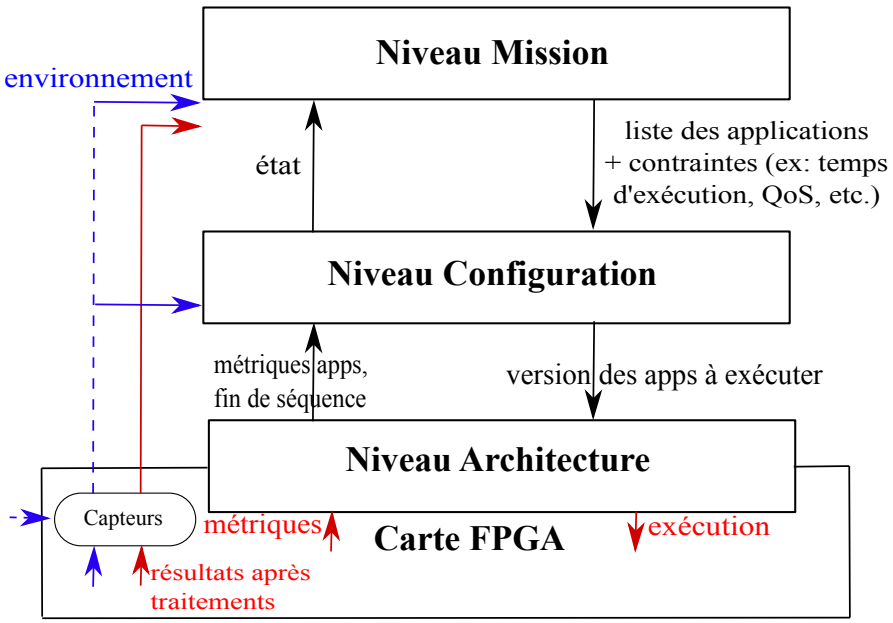


Figure 1.1: *Architecture du projet HPeC.*

tels que les arbres de défaillances, les systèmes experts, les réseaux de neurones, les réseaux Bayésiens, etc. Nous utilisons dans nos travaux les réseaux Bayésiens car c'est le modèle le plus utilisé pour le diagnostic dans un environnement contenant des incertitudes [Pearl and Russell, 1998] [Zermani et al., 2017b]. Pour la prise de décision, nous utilisons le modèle de Markov Decision Process pour décrire la mission étudiée. Nous avons fait le choix des MDP car c'est un modèle qui fait face aux incertitudes dans le cas où tous les états de la mission sont complètement observables [Cassandras and Lafortune, 2009] comme dans le cas de la mission HPeC.

Les systèmes autonomes et auto-adaptatifs sont amenés à exécuter une multitude d'applications (simple et complexes). Ce qui peut conduire très rapidement à une surcharge du CPU. Outre l'accélération matérielle qu'elles peuvent procurer, les cartes embarquées à base de SoC-FPGA peuvent permettre de décharger le processeur, les cartes embarquées hybrides CPU/FPGA fournissent aussi un banc d'essai intéressant pour les implémentations matérielles/logicielles adaptables. Une implémentation sur FPGA permet aussi de re-configurer le support d'exécution avec les différents plans de mission produits par le planificateur de mission.

1.2 Problèmes scientifiques

La Figure 1.2 schématise les problématiques scientifiques auxquelles cette thèse se propose d'apporter des réponses. Les systèmes autonomes réalisent différents types de missions dans différents domaines. Durant l'exécution de la mission par le sys-

tème autonome, des aléas et des événements aléatoires (obstacle, support d'exécution surchargé, etc.) peuvent apparaître causant probablement l'échec de la mission. Par conséquent, la première problématique qui se pose est : **par quel moyen pouvons-nous identifier ces aléas de la mission ?** Une fois l'événement aléatoire identifié et détecté, le système doit réagir à cet événement en prenant la décision du nouveau plan de mission à exécuter pour continuer la mission. Ce qui nous amène à une deuxième problématique qui est : **quel modèle de décision utiliser et comment faire en sorte d'intégrer les contraintes et les aléas de la mission ?** Enfin, comme nous sommes dans un contexte de systèmes autonomes embarqués, la troisième et dernière problématique est **comment faire en sorte que les méthodes de détection et de décision puissent être embarquées au sein d'un système autonome et comme réciproquement prendre en compte les aléas que le système embarqué introduit ?**

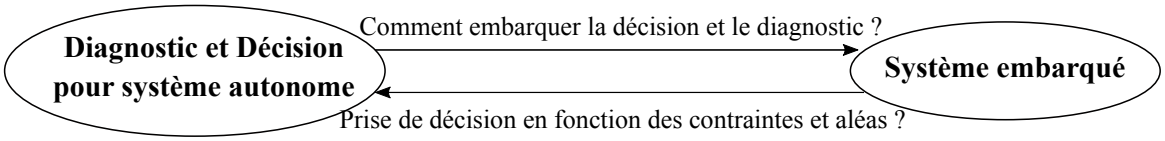


Figure 1.2: *Questions auxquelles la thèse apporte des réponses.*

1.3 Contributions

Nos principales contributions sont résumées dans les points suivants :

1. **Proposition d'un modèle de diagnostic intégrant l'évaluation de la QoS des applications embarquées**
   Classiquement, les planificateurs de mission ne prennent en considération que les événements externes liés à l'environnement. Cependant, ces aléas environnementaux peuvent impacter l'état de santé du véhicule et de ses composants. Pour répondre à cette problématique, nous proposons un modèle de diagnostic et de monitoring. Ce modèle de diagnostic repose sur l'utilisation des réseaux Bayésiens. Nous nous concentrons particulièrement sur l'évaluation de la qualité de service (QoS) des applications embarquées sur le véhicule autonome (drone). L'évaluation de la QoS des applications aidera dans l'estimation de la probabilité de la version algorithmiques de l'application à exécuter en fonction de la disponibilité des ressources et du contexte environnemental. Le modèle Bayésien pour le diagnostic est construit à partir d'une table d'analyse de défaillances de type FMEA (Failure Mode and Effect Analysis).
   Pour un diagnostic fiable il faut que les paramètres du réseau Bayésien soient représentatifs de la réalité. Souvent, on constate un manque au niveau des données d'expertise permettant d'avoir ces paramètres. Pour cela, nous abor-

derons l'apprentissage des paramètres de réseau Bayésien et nous proposerons une version embarquable de l'algorithme d'apprentissage pour BNs.

2. **Proposition d'un modèle de planification de mission compact, générique et embarquable au sein d'un système autonome**
   La planification de mission par prise de décision autonome est de plus en plus nécessaire pour l'exécution de missions par des véhicules autonomes. La prise de décision doit répondre à plusieurs critères tels que l'objectif de la mission, les changements de contexte (événements), les défaillances, etc.
   Pour répondre à cette problématique, nous proposons un modèle de planification de mission capable de faire face aux imprévus lors d'une mission de drone autonome. Le planificateur de la mission doit être générique pour qu'il puisse être appliqué à tous les types de mission et compact en regroupant différents types de modèles probabilistes pour la gestion de la mission. Le modèle proposé est appelé BFM pour modèle Bayésien construit à partir de table FMEA pour alimenter un MDP. Le modèle BFM est divisé en deux niveaux qui collaborent pour prendre une décision optimale. Le premier niveau est le niveau de diagnostic, à l'aide de BNs, qui s'occupe d'évaluer l'état de santé du véhicule autonome et de ses composants matériels et aussi évaluer la QoS des applications embarquées. Le deuxième niveau est le niveau de décision qui s'appuie sur les modèles MDP. Les BNs alimentent les MDPs avec les valeurs de probabilités.
3. **Implémentation matérielle et logicielle des MDPs du planificateur de mission sur SoC-FPGA**
   L'intérêt d'un planificateur de mission est de pouvoir produire des plans de mission (décision) durant l'exécution de la mission par le véhicule autonome. Pour cela, nous proposons une version embarquable sur une carte hybride SoC-FPGA de notre modèle BFM. Plus précisément, proposer une version logicielle/matérielle embarquable du MDP pour mieux répondre aux contraintes du système et réduire le temps de réponse. Nous proposons aussi un atelier logiciel permettant de générer automatiquement le modèle BFM (avec ses deux niveaux : BN diagnostic et MDP décision) ainsi que la version embarquable de l'algorithme de résolution du modèle de décision MDP. La génération de la version matérielle embarquable du planificateur de mission est réalisée avec les outils de synthèse de haut niveau de Xilinx (HLS, Vivado et SDK).
4. **Intégration du modèle de planificateur de mission proposé dans le projet HPeC**
   Pour valider notre modèle, nous avons intégré le modèle BFM dans un cas d'étude de mission de drone. La mission étudiée est une mission de tracking où le drone parcourt une trajectoire pré-définie et ensuite se met en mode détection et poursuite de cible potentielle. La mission est définie dans le cadre du projet HPeC, dans lequel s'inscrit cette thèse. Différents scénarios ont été définis pour valider les différents aspects du modèle BFM (évaluation de la

QoS des applications embarquées et prise de décision). L'intégration du planificateur de mission dans le projet HPeC a été fait en utilisant les outils ROS (Robot Operating System) et HIL (Hardware In The Loop) où chaque niveau du projet HPeC (Figure 1.1) (niveau planificateur de mission, niveau contrôleur de configuration et niveau architecture) est représenté par un nœud ROS. L'ensemble de l'architecture ROS plus HIL est par la suite embarqué sur la carte FPGA du projet HPeC.

1.4 Plan du mémoire

Ce manuscrit de thèse est organisé comme suit :

Dans le second chapitre, nous présentons un état de l'art sur les différents modèles utilisés tout au long de cette thèse. Nous commençons par la partie diagnostic à l'aide de réseaux Bayésiens où nous présentons des généralités sur la construction de BN, l'inférence Bayésienne et enfin l'apprentissage des paramètres d'un BN. Ensuite, nous discutons de la partie décision en présentant le modèle MDP, la résolution d'un MDP et l'application de l'apprentissage par renforcement sur un MDP. Enfin, nous concluons le chapitre par la présentation de planificateur de mission embarqué, les accélérateurs à base de FPGA et les outils de conception d'accélérateurs associés.

Le troisième chapitre est dédié à notre contribution concernant le diagnostic avec les BNs et l'évaluation de l'état de santé des applications embarquées. Nous exposons la spécification et la construction de BN dédiés aux applications à partir d'une analyse FMEA, l'évaluation de l'état de santé (QoS) des applications embarquées en fonction du contexte environnemental et de la disponibilité des ressources de calculs. Ensuite, nous abordons la version embarquable de l'inférence et de l'apprentissage des paramètres d'un BN. Enfin, nous présentons l'inférence Bayésienne à l'aide d'un mécanisme d'inférence embarquable.

Le quatrième chapitre est consacré à la présentation de notre modèle BFM dédié à la planification de mission et la prise de décision. Nous exposons deux versions du modèle BFM, une version monolithique et une version concurrente. Nous présentons, pour chacune des deux versions, la spécification d'une mission de véhicule autonome et dans le cas de la version concurrente nous intégrons la gestion de conflits possibles qui peuvent survenir entre les différentes politiques (décisions) produites par les différents BFMs concurrents. Nous concluons le chapitre en montrant la scalabilité de notre modèle BFM.

Dans le cinquième chapitre, nous abordons l'implémentation embarquée du modèle BFM et particulièrement le module de décision avec les MDPs. Nous présenterons un comparatif entre les deux algorithmes les plus utilisés pour la résolution de MDPs. Nous proposons un flot de conception intégrant des outils de synthèse de haut niveau (HLS) et la conception d'architecture embarquable (Vivado Xilinx et SDK) afin de pouvoir proposer une implémentation matérielle/logicielle (HW/SW)

qui s'exécute sur une carte FPGA dans le but d'accélérer les calculs et ainsi générer des plans de mission durant l'exécution de la mission. Un framework pour générer automatiquement le modèle BFM est alors présenté. Enfin, nous validons la version matérielle du modèle BFM avec des scénarios définis à partir de la mission de tracking étudiée.

Finalement dans le sixième chapitre, nous présentons l'intégration et validation du modèle BFM dans le cadre du projet HPeC. Nous commençons par lister les scénarios de la mission de tracking ainsi que les différents capteurs embarqués sur le drone de la mission HPeC (mission de tracking). Nous validons notre planificateur de mission avec les scénarios prèdéfinis, ensuite nous intégrons le planificateur dans l'architecture globale du projet HPeC et nous montrons l'interaction entre le planificateur de mission et le contrôleur de configuration dans le but d'assurer le bon déroulement de la mission.

Enfin, dans le dernier chapitre, nous concluons avec un rappel de la problématique et des contributions, ainsi que nos perspectives futures.

1.5 Publications

Revues internationales

1. C. Hireche, C. Dezan, S. Mocanu, D. Heller et J.P. Diguet. "**Context/Resource-Aware Mission Planning based on BNs and concurrent MDPs for autonomous UAV**", *MDPI-Sensors Journal*, 2018. [Hireche et al., 2018b]
2. S. Zermani, C. Dezan, C.Hireche, R. Euler, et J. Diguet, "**Embedded Context Aware Diagnosis for a UAV SoC Platform**", *Elsevier Embedded Hardware Design Journal (MICPRO' 17)*, 2017. [Zermani et al., 2017a]

Conférences internationales

3. C. Hireche, C. Dezan, J.P. Diguet et L. Mejias. "**BFM : a Scalable and Resource-aware Method for Adaptive Mission Planning of UAVs**", *International Conference on Robotics and Automation (ICRA' 18)*, Australie, Brisbane, mai 2018. [Hireche et al., 2018a]
4. C. Hireche, C. Dezan, et J.P. Diguet. "**Online diagnosis updates for embedded health management**", *6th Mediterranean Conference on Embedded Computing (MECO' 17)*, Montenegro, juin 2017. [Hireche et al., 2017]
5. S. Zermani, C. Dezan, C.Hireche, R. Euler, et J. Diguet, "**Embedded and Probabilistic Health Management for the GPS of Autonomous Vehicles**", *5th Mediterranean Conference on Embedded Computing (MECO' 16)*, Montenegro, juin 2016. [Zermani et al., 2016]

Conférences nationales

6. S. Zermani, C. Dezan, C.Hireche, R. Euler, et J. Diguet, "**Génération de composant "état de santé" pour monitorer le système embarqué de véhicule autonome**", *Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS' 16)*, Lorient, juillet 2016.

- Chapitre 2 -

État de l'art

Sommaire

2.1 Introduction

L'utilisation des systèmes autonomes s'est énormément démocratisée et cela dans différents domaines. Parmi ces systèmes autonomes, nous retrouvons les robots terrestres, les drones ainsi que les robots sous-marins. Les domaines d'utilisation des robots sont divers et variés tels que l'agriculture, la défense, la surveillance des forêts, etc. Souvent les missions assignées aux robots se déroulent dans un environnement incertain et loin de la base. Il est donc nécessaire que le robot soit autonome au niveau de la gestion de sa mission.

La gestion de la mission implique une prise de décision autonome et en ligne (durant la mission). Dans ce cas, des mécanismes de prise de décision embarquables sur le système autonome et qui permettent de traiter les incertitudes liées au contexte externe et interne sont nécessaires. Il existe une multitude de modèles probabilistes pour la prise de décision, parmi eux les MDPs, les POMDPs (Partially Observable Markov Decision Process), les diagrammes d'influences, etc. Pour qu'une décision soit produite par le modèle de décision, il est essentiel d'avoir des observations sur le contexte dans lequel évolue le système et la mission. Ce qui nous amène à un autre type de modèles, cette fois dédié au diagnostic.

Dans la famille des modèles probabilistes pour le diagnostic, nous retrouvons les réseaux Bayésiens, les Fault Tree, etc. Ces modèles permettent d'une part de surveiller le système ainsi que le contexte environnemental et d'autre part évaluer l'état de santé du système et de ses composants matériels et logiciels. L'observation des changements de contexte ainsi que la détection de défaillance peut être faite à l'aide des modèles de diagnostic. Par la suite générer la décision la plus adéquate et adapter au changement observé.

Dans ce chapitre, nous allons aborder les différents niveaux et aspects pour le contrôle et la gestion de mission de système autonome. La Section 2.2 présente les réseaux Bayésiens, leurs constructions, l'inférence dans un réseau Bayésien pour le diagnostic et enfin l'apprentissage des paramètres (probabilités initiales) d'un BN. La Section 2.3 présente le modèle de décision utilisant les MDPs. Dans cette section, nous définissons un MDP ainsi que la méthode de construction d'un MDP. La résolution et l'objectif d'un MDP et en dernier nous présentons l'apprentissage par renforcement dans le cas de MDP. Enfin, la Section 2.4 abordera de planificateur de mission (mission manager) pour l'embarqué. Nous présenterons les outils utilisés dans le but de proposer une implémentation matérielle et logicielle embarquable du planificateur de mission. Nous terminerons ce chapitre par une conclusion.

2.2 Diagnostic

Avec l'avancée de la technologie dans divers domaines, le diagnostic est de plus plus nécessaire et appliqué à tous les domaines. Il existe différents modèles et méthodes pour le diagnostic tels que les réseaux de neurones, les arbres de défaillances, les BNs, FMEA, etc. Les réseaux de neurones sont utilisés dans le domaine médical [Amato et al., 2013] et aussi pour la détection de défaillances sur un système complexe [Li et al., 2000]. Les arbres de défaillances (FTA : Fault Tree Analysis) sont, quant à eux, utilisés pour la détection de fautes dans des systèmes électroniques [Reese and Leveson, 1997]. Le principe des FTA consiste à identifier un événement que nous souhaitons analyser (ex : une panne particulière dans un système), appelé événement principal (TE : Top Event) [Bobbio et al., 2001]. La construction du FTA se fait du haut vers le bas (de l'événement principal vers les causes de cet événement) jusqu'à ce que les défaillances des composants de base soient atteintes. La structure d'un FTA respecte les points suivants : 1) les événements sont binaires (fonctionne et ne fonctionne pas), 2) les événements sont statistiquement indépendants et 3) la relation causale entre les événements et les causes est représentée par des portes logiques (ET, OR, XOR). La méthode FMEA (Failure Mode and Effect Analysis) est aussi une méthode classiquement utilisée pour faire du diagnostic, notamment dans le domaine de la robotique [Korayem and Iravani, 2008]. La FMEA est une méthode qualitative qui liste les modes de défaillances durant un processus de conception. Cette méthode utilise le RPN (numéro de priorité du risque) pour trier les mode de défaillances en fonction du contexte observé. Des actions correctives peuvent être proposées. Le fait d'établir une table FMEA pour un système donnée (ex : robot) peu améliorer la fiabilité de ce dernier.

D'autre part, les réseaux Bayésiens [Pearl, 1988] sont des modèles probabilistes utilisés pour analyser le comportement d'un système donné. Souvent les systèmes évoluent dans des contextes incertains, ce qui rend l'utilisation des réseaux Bayésiens légitime et plus appropriée. Les BNs permettent de représenter les éléments, nécessaires à l'analyse du comportement d'un système, sous forme d'un graphe et de stocker les informations et connaissances sur le système dans ses variables. Le modèle Bayésien peut être construit soit de manière intuitive manuelle ou bien à l'aide de méthode d'apprentissage en s'appuyant sur les connaissances recueillies sur le système. La résolution des BNs se fait grâce à des algorithmes d'inférence [Naïm et al., 2007]. Cela consiste à propager l'information certaine ou l'observation que nous avons du système dans l'ensemble du graphe du BN afin de calculer les probabilités a posteriori des variables souhaitées.

2.2.1 Modèle pour diagnostic : Les réseaux Bayésiens

De nos jours, les BNs représentent un formalisme complet s'appuyant sur les théories des probabilités et des graphes ainsi que l'association avec des technolo-

gies de l'intelligence artificielle. L'utilisation des BNs est motivée comparativement à l'utilisation d'autres modèles, tels que les arbres de défaillances, les réseaux de neurones, les arbres de décision, etc, par les points suivants [Naïm et al., 2007] :

- Évolution du système dans un environnement avec incertitudes (incertitudes dans les observations et apparitions des événements).
- Représentation graphique du système simple et intuitive, compréhensible par un utilisateur non-expert.
- Diverse utilisation possible des BNs (diagnostic, prédiction et décision) selon l'objectif souhaité.
- Disponibilité de plusieurs outils logiciels graphiques ou non pour la manipulation des BNs. Ces outils incluent toutes les méthodes de résolution et d'apprentissage sur les BNs.

Les domaines d'application des réseaux Bayésiens sont divers et variés. Parmi ces domaines, nous retrouvons :

- La détection de fraude
  L'application développée par la société de télécommunications ATT permet de répondre à deux objectifs en utilisant les réseaux Bayésiens [Ezawa and Schuermann, 1995]. Premièrement, détecter un risque élevé de non-recouvrement soit au niveau du client ou bien au niveau des appels. Deuxièmement, décider des actions à effectuer en fonction du niveau de risque détecté.
- La santé
  Les réseaux Bayésiens on été très rapidement utilisés pour le diagnostic médical. L'utilisation dans ce domaine a été motivée par le fait que les BNs se basent sur l'expertise humaine et les données statistiques. Nous citons les travaux de [Becker et al., 1998] et [Friedman et al., 2000] pour le diagnostic médical, analyse d'arbre généalogique pour la localisation de gènes .
- La défense
  Les BNs sont utilisés pour la fusion de données car ils offrent la possibilité de considérer des données incertaines pour établir un diagnostic ou une vérification [Liu and Wu, 2011].
- L'industrie
  Les BNs sont souvent utilisés dans le cas de systèmes autonomes ou adaptatifs dans le but de trouver des solutions pour faire face aux changements et événements liés au contexte dans lequel évolue le système [Skaanning, 2000] ; [Hart and Graham, 1997].
- L'informatique et les télécommunications
  Les BNs sont utilisés pour le diagnostic et la détection de défaillances [Ezawa and Schuermann, 1995] ; [Kim and Valtorta, 1995].

Dans ce qui suit, nous allons définir et présenter la construction d'un réseau Bayésien de manière intuitive puis de manière plus formelle.

Définition d'un réseau Bayésien

Les réseaux Bayésiens font partie de la famille des modèles probabilistes. Ils permettent de représenter et modéliser la connaissance et le comportement des systèmes complexes.

1. **Structure du réseau Bayésien** : un BN est représenté par un graphe orienté acyclique (Directed Acyclic Graph - DAG) où chaque variable du système ou composant est désigné par un nœud du graphe. Chaque nœud (variable) est caractérisé par au moins deux états (voir plus). Les nœuds sont reliés entre eux par des arcs qui représentent une relation causale. Intuitivement, un BN représente une relation de *cause* à *effet* entre les variables, d'un système, décrivant son comportement.
2. **Paramètres du réseau Bayésien** : après la définition de l'ensemble des variables du système à décrire ; à chaque variable (nœud) du BN est associée une table de probabilité conditionnelle (Conditionnel Probability Table - CPT). Chaque CPT contient les valeurs de probabilités associées à chacun des états d'un nœud du BN en fonction des états considérés pour le(s) nœud(s) parent(s).

La Figure 2.1 donne un exemple de BN avec sa structure et ses paramètres. De manière plus formelle, un réseau Bayésien est défini comme suit [Naïm et al., 2007], [Jensen and Nielsen, 2007] :

— Un réseau Bayésien est un DAG (graphe orienté acyclique), G = (V,E). V est l'ensemble des variables du BN. E est l'ensemble des arcs du BN.

— Un espace probabiliste fini (Ω, ζ, p).

— Un ensemble de variables aléatoires associées aux nœuds du graphe et définies sur (Ω, ζ, p), tel que :

$$P(V_1, V_2, ..., V_n) = \prod_{i=1}^{N} p(V_i|C(V_i)). \tag{2.1}$$

C(V_i) correspond à l'ensemble des parents (causes) du nœud V_i. P(V_1, V_2, ..., V_n) représente la distribution de probabilité jointe du BN sur l'ensemble N.

Exemple simple de réseau Bayésien

Il existe une multitude d'exemples dans la littérature expliquant le raisonnement et la construction de réseaux Bayésiens. Dans ce chapitre, nous reprenons l'exemple du problème de démarrage d'une voiture [Jensen and Nielsen, 2007]. Le problème est formulé comme suit :
La voiture ne démarre pas ce main. Le démarreur est lancé, il fonctionne bien, mais la voiture ne démarre toujours pas. Il peut y avoir plusieurs raisons à ce problème. On entend le bruit du démarreur, donc il y a assez d'énergie dans la batterie. Par

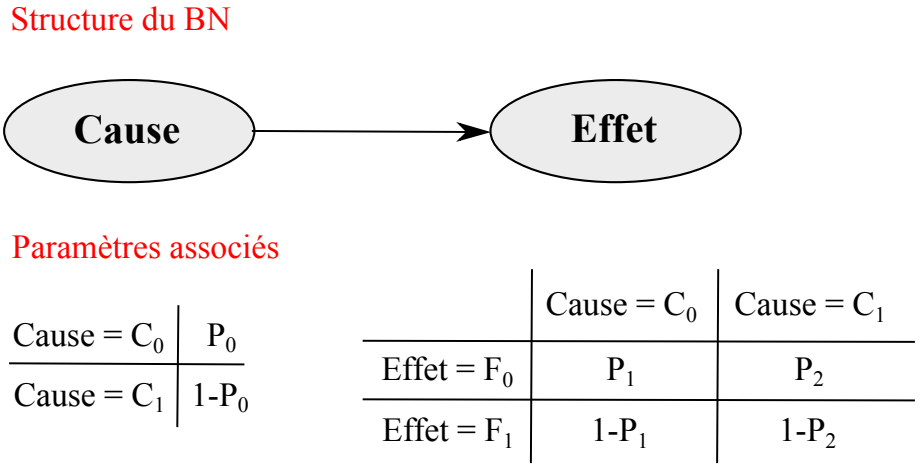


Figure 2.1: *Illustration de la représentation d'un réseau Bayésien.*

conséquent, les causes les plus probables sont : le carburant a été volé durant la nuit, les bougies d'allumage ne produisent plus d'étincelle, un problème dans le système d'allumage ou bien présence de saleté dans le carburant. L'objectif est de déduire la cause du problème à partir d'observations.

A l'aide de cet exemple, nous allons construire le réseau Bayésien décrivant le problème de démarrage de la voiture en suivant les étapes de construction d'un BN. Premièrement, définir les variables (nœuds) du BN ainsi que l'ensemble de leurs états. Ensuite, définir la structure du BN en déterminant les liens entres les nœuds en s'assurant de respecter la circulation d'information au sein du BN. Enfin, établir les tables de probabilités conditionnelles (CPT) associées au nœuds du BN.

1. Définir l'ensemble des variables et leurs états : il s'agit d'identifier les informations ou les variables qui définissent le problème à résoudre. Par la suite, pour chacune des variables identifiées, définir l'ensemble de ses états. Pour le problème de démarrage de la voiture, nous identifions les variables suivantes :
   - Carburant : deux états possibles, 'Oui' et 'Non', pour indiquer si oui il y a un problème de carburant ou non.
   - Bougies d'allumage : deux états sont définis, l'état 'Oui' et l'état 'Non', pour dire si les bougies sont sales ou pas.
   - démarrer : indique si la voiture démarre ou pas. Pour cela, deux états sont définis 'Oui' et 'Non'.
   - compteur de carburant : surveille et donne l'information sur le niveau du carburant. Trois états sont définis 'Rempli', 'Moitié' et 'Vide'.
2. Définir la structure du BN : cela revient à établir les liens de dépendance conditionnelle entre les différents nœud du BN. Par définition, un BN est un graphe acyclique, donc il est interdit d'avoir des boucles dans le graphe. Afin de construire le raisonnement et trouver la cause du problème à l'aide du BN,

Graphe	Règle de circulation de l'information
	Type : connexion convergente L'information circule de V_1 à V_3 si seulement si V_2 est connue
	Type : connexion divergente L'information circule de V_1 à V_3 si seulement si V_2 n'est pas connue
	Type : connexion série L'information circule de V_1 à V_3 si seulement si V_2 n'est pas connue

Tableau 2.1: *Circulation d'information selon le type de connexion dans un BN.*

il faut s'assurer que l'information circule entre les différents nœuds du graphe. Pour cela, il faut respecter les règles de circulation d'information au sein d'un BN dite D-séparation [Naïm et al., 2007]. Pour l'exemple de démarrage d'une voiture, les liens établis, en respectant les règles de la Table 2.1 sont :

- Nœud 'Carburant' vers nœud 'Démarrer' : l'absence de carburant cause le fait que la voiture ne démarre pas.
- Nœud 'Bougies d'allumage' vers nœud 'Démarrer' : idem que le carburant. Des bougies d'allumage non propre cause le non démarrage de la voiture.
- Nœud 'Carburant' vers nœud 'Compteur de carburant' : la disponibilité ou non du carburant est monitorer par le compteur de carburant.

Les nœuds 'Carburant', 'Bougies d'allumage' et 'Démarrer' forment une connexion convergente. Les nœuds 'Carburant', 'Démarrer' et 'Compteur de carburant' forment une connexion divergente.

3. Définition des paramètres du BN : les valeurs de probabilités de chaque nœud Bayésien représentent les probabilités d'observer l'$état_0$ ou l'$état_1$ du nœud. Pour les nœuds racine du BN (nœuds sans parents) nous parlons de probabilités a priori. Pour les nœuds fils (nœuds ayant des nœuds parents ou causaux), nous parlons de probabilités conditionnelles.

Les tables 2.2 et 2.3 représentent respectivement les tables de probabilités a priori et les tables de probabilités conditionnelles associées à chacun des nœuds Bayésiens décrivant l'exemple du problème de démarrage d'une voiture. Une fois tous les éléments nécessaires à la construction d'un réseau Bayésien définis (Variables, structure et paramètres), nous obtenons le BN suivant pour l'exemple de démarrage d'une voiture (voir Figure 2.2).

((a)) *Probabilités du nœud 'Carburant'.*

Carburant = Oui	0.5
Carburant = Non	0.5

((b)) *Probabilités du nœud 'Bougies d'allumage'.*

Bougies = Oui	0.5
Bougies = Non	0.5

Tableau 2.2: *Tables de probabilités a priori des nœud racine.*

((a)) *CPT du nœud 'Compteur carburant'.*

	Carburant=Oui	Carburant=Non
Compteur=Rempli	0.39	0.001
Compteur=Moitié	0.60	0.001
Compteur=Vide	0.01	0.998

((b)) *CPT du nœud 'Démarrer'.*

	Carburant=Oui		Carburant=Non	
	Bougie=Oui	Bougie=Non	Bougie=Oui	Bougie=Non
Démarrer=Oui	0.99	0.01	0	0
Démarrer=Non	0.01	0.99	1	1

Tableau 2.3: *Tables de probabilités conditionnelles des nœuds fils.*

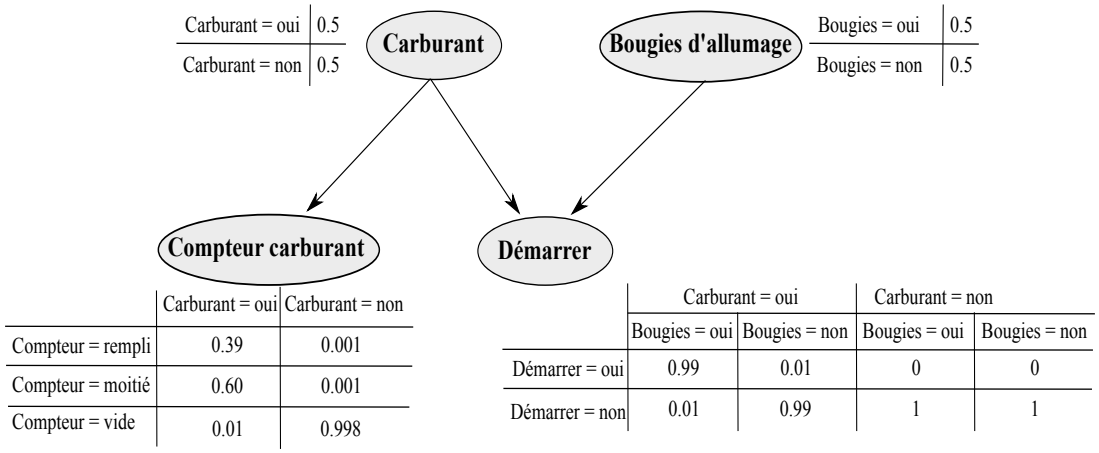


Figure 2.2: *Réseau Bayésien représentant le problème de démarrage d'une voiture.*

2.2.2 Inférence dans les Réseaux Bayésiens

Un réseau Bayésien permet de faire du diagnostic, prédiction, décision, etc. Cela revient à construire un raisonnement Bayésien. Le raisonnement Bayésien est un calcul de probabilité conditionnelle dans le but de trouver la cause du problème décrit en fonction des observations faites sur l'ensemble du système. Ce calcul suit le théorème de Bayes.

Théorème 2.2.1 *Théorème de Bayes*
Soit deux variables A et B. A étant le nœud parent et B le nœud fils. L'ensemble des états de A sont (a_0, a_1, ...) et l'ensemble des états de B sont (b_0, b_1, ...). Le calcul de la probabilité conditionnelle du nœud A (dont l'état a_0 est observé) sachant l'observation faite sur le nœud B (avec l'observation de l'état a_0) est comme suit :

$$P(A = a_0|B = b_0) = \frac{P(B = b_0|A = a_0) * P(A = a_0)}{P(B = b_0)} \tag{2.2}$$

Maintenant, si nous ajoutons une troisième variables C(c_0, c_1, ...) contexte aux deux précédentes (A et B). Le calcul de la probabilité du nœud A sachant l'observation sur le nœud B et sur le nœud C se fait comme suit :

$$P(A = a_0|B = b_0, C = c_0) = \frac{P(B = b_0|A = a_0, C = c_0) * P(A = a_0|C = c_0)}{P(B = b_0|C = c_0)} \tag{2.3}$$

Nous pouvons facilement remarquer que le calcul des probabilités conditionnelles (a posteriori) est relativement complexe selon le réseau Bayésien défini. Cette complexité varie en fonction de la taille du BN (nombre de nœuds), la connectivité au sein du BN (nombre de liens (arcs)), nombre d'états et enfin le nombre de parents pour chaque nœud du BN. Afin de répondre à ce besoin de calcul, des recherches ont été faites pour établir des algorithmes dit d'*inférence*. Il existe deux catégories de méthodes d'inférence, les méthodes exactes et les méthodes approchées. Dans la catégorie des méthodes exactes, nous retrouvons l'algorithme de propagation de l'information au sein d'un réseau [Jensen and Nielsen, 2007], [Lauritzen, 1992], [Lauritzen and Spiegelhalter, 1990], etc. Pour les méthodes approchées, nous retrouvons les travaux [Henrion, 1986], [Mengshoel et al., 2011].

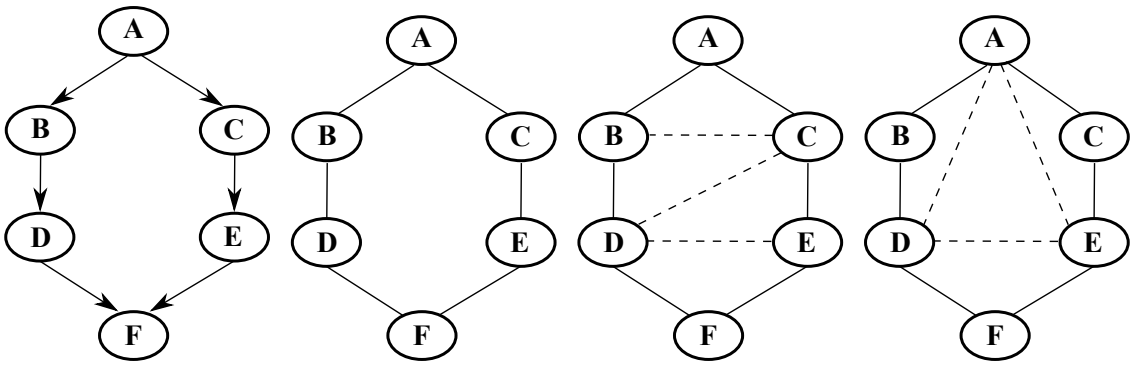
Dans la suite de cette section, nous nous intéressons aux méthodes exactes comme l'algorithme par *arbre de jonction*. L'arbre de jonction est l'algorithme le plus utilisé pour le calcul des probabilités conditionnelles a posteriori dans les logiciels traitant les réseaux Bayésiens ou bien dans la littérature. Ensuite, nous aborderons l'inférence à l'aide d'une méthode de compilation appelée *circuit arithmétique* qui est une méthode exacte plus simple que l'arbre de jonction.

Inférence par arbre de jonction

L'inférence utilisant l'algorithme *arbre de jonction* s'appuie sur le fait de consruire un arbre de jonction à partir du réseau Bayésien initial et d'effectuer par la suite le

calcul de probabilité. L'inférence par arbre de jonction est divisée en deux phases. la première phase consiste à traduire le BN en un arbre de jonction et la deuxième phase consiste à calculer les valeurs de probabilités a posteriori.

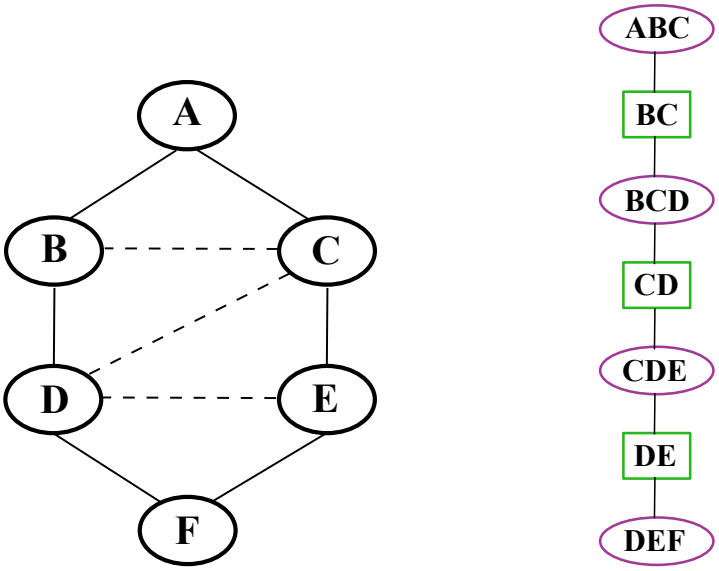
1. Construction de l'arbre de jonction à partir d'un BN : les étapes de construction de l'arbre de jonction pour l'inférence à partir d'un BN donné sont les suivantes.
   - **La moralisation** : consiste à rendre, dans un premier temps, le BN non-orienté en supprimant les directions des arcs du BN. Ensuite, relier les parents d'un même nœud par un arc non-orienté.
   - **La triangulation** : reprendre le graphe moralisé. Pour chaque clique de plus de 3 nœuds, la réduire en clique de trois nœuds maximum [Kjærulff, 1990]. Cela se fait en ajoutant un arc non-orienté entre une variable (nœud) de la clique (de plus de 3 nœuds) et tous les voisins de cette même clique.
   - **Constituer l'arbre de jonction final** : une fois le BN moralisé et triangulé, l'arbre de jonction peut être construit. Une clique est l'ensemble de trois nœuds maximum reliés entre eux par des arcs non-orientés. Un séparateur est l'intersection entre deux cliques. Par conséquent, un séparateur est un (ou plusieurs) nœud (s) appartenant à deux cliques différentes. La construction de l'arbre de jonction revient donc à déduire du graphe moralisé triangulé toutes les cliques et tous les séparateurs.



((a)) *Réseau Bayésien* **((b))** *Graphe moralisé* **((c))** *Graphe triangulé* **((d))** *Graphe triangulé*

Figure 2.3: *Moralisation et triangulation d'un BN.*

La Figure 2.3 illustre à travers un exemple les étapes de moralisation et de triangulation d'un BN qui permettront de construire par la suite l'arbre de jonction. Nous constatons bien que selon la structure du graphe, plusieurs triangulations sont possible. La Figure 2.3(c) et 2.3(d) montre bien cela. À partir du graphe triangulé (Figure 2.3(c)), nous allons, maintenant, construire l'arbre de jonction associé.



((a)) *Graphe triangulé*

((b)) *Arbre de jonction associé*

Figure 2.4: *Arbre de jonction associé au graphe triangulé de l'exemple.*

Le graphe triangulé montre explicitement les familles de cliques qui constitueront l'arbre de jonction. À partir de la Figure 2.3(c), nous déduisons les familles de cliques suivantes : (ABC), (BCD), (CDE) et (DEF). Les séparateur sont : (BC), (DC) et (DE). La Figure 2.4 montre l'arbre de jonction obtenu.

Reprenons l'exemple du problème de démarrage d'une voiture (décrit plus haut) et voyons l'arbre de jonction associé (Figure 2.5).

2. Calcul des probabilités par arbre de jonction : se fait à l'aide de la propagation d'informations dans un arbre. Le calcul des probabilités conditionnelles est fait en suivant trois étapes : initialisation, propagation de l'information et normalisation [Naïm et al., 2007], [Huang, 1996].
   - Initialisation : revient à initialiser les probabilités potentielles (Φ) des nœuds à travers les CPT du BN.
     - Pour chaque clique et séparateur C : $\Phi_c \leftarrow 1$.
     - Pour chaque clique : $\Phi_c \leftarrow \Phi_c$ P(V | Pa_X), V représente les variables de la clique et Pa_X un parent de X inclu dans la clique.
   - Propagation de l'information : se fait en deux phases, envoi des informations puis propagation de l'information.
     - Envoi des informations : toutes les cliques envoient une information, en passant par tous leurs voisins, vers une clique choisie C. À chaque passage d'une information d'une clique C1 à une clique C2, les probabilités potentielles de C2 et R (séparateur entre C1 et C2) sont mises à jour

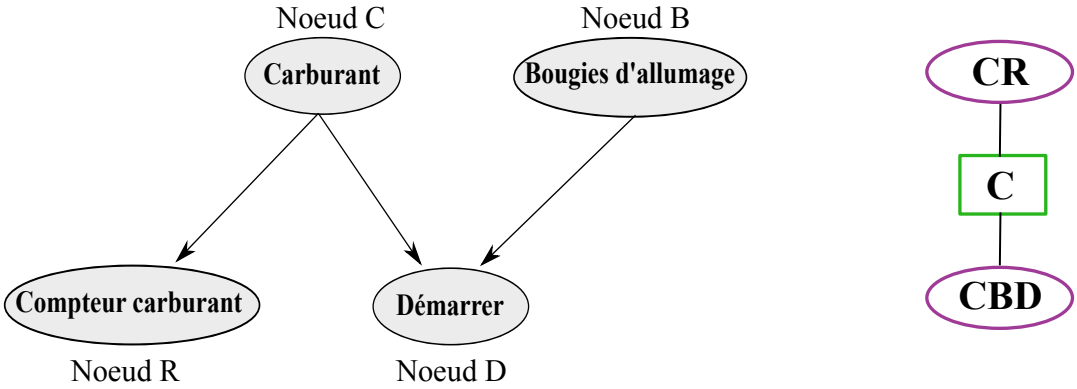


Figure 2.5: *Arbre de jonction associé au BN du problème démarrage d'une voiture.*

comme suit :

$$\begin{cases} \Phi'_R = \Phi_R \\ \Phi_R = \sum\limits_{C1|R} \Phi_{C1} \\ \Phi_{C2} = \Phi_{C2} \dfrac{\Phi_R}{\Phi'_R} \end{cases}$$

- Propagation des informations : une fois que la clique choisie a reçue toutes les informations, elle les propage vers toutes les autres cliques. La mise à jour des valeurs de probabilités se font comme dans la phase 1 (envoie des informations).

— Normalisation : l'obtention des probabilités conditionnelles d'une variables V(v_0, v_1, ...) sachant les évidences se fait en déduisant les probabilités marginales à partir de la propagation globale de l'information. Cela pour avoir :
$P(V = v_i|e) = \frac{P(V=v_i,e)}{P(e)}, i \in \{0, 1, \dots\}$
Où $P(e) = \sum\limits_{C|e} \Phi_C$, où C est une clique contenant e,
et $P(V = v_i, e) = \sum\limits_{C|e,v_i} \Phi_C$ où C est une clique contenant v.

Le calcul d'inférence par arbre de jonction a une complexité de l'ordre de O(n exp(w)), n étant le nombre de nœuds et w la largeur de l'arbre de jonction construit à partir d'un réseau bayésien.

Inférence par circuit arithmétique

Dans cette section, nous allons aborder l'inférence par circuit arithmétique (AC). Il existe des travaux fait autour de l'inférence avec AC, notamment ceux de [Darwiche, 2003] dans le but de proposer des algorithmes puissants en termes de calculs et de performances ; mais surtout une méthode d'inférence pouvant fournir des performances en temps réel. L'inférence AC s'appuie sur la factorisation de fonctions

multi-linéaires et sur la dérivée partielle. Comme pour l'inférence avec arbre de jonction, l'inférence avec AC est décomposée en deux phases. Dans ce qui suit, nous allons présenter ces phases.

1. Phase de construction de l'AC : consiste à traduire le BN en une fonction multi-linéaire (MLF) comme suit :
   - Génération du polynôme : définition de la fonction MLF du BN. La méthode classique utilisée pour la définition du polynôme revient à définir une fonction f unique pour la structure du BN selon l'équation suivante :

$$f = \sum_{v} \prod_{vu \sim x} \lambda_v \theta_{v|u} \tag{2.4}$$

   Où : λ_v représente l'ensemble des évidences, $\theta_{v|u}$ représente les valeurs des probabilités associées aux variables V et U et $\sim$ désigne la relation de compatibilité entre les instanciations [Darwiche, 2003].
   Soit V une variable du BN et v_0, v_1, ... ses états. L'ensemble des évidences est défini comme suit :

   — Si V est une observation : $\lambda v_i = 1, i \in \{0, 1, \dots\}$ lorsque v_i est l'état observé par V, 0 sinon.
   — Si V n'est pas une observation : $\lambda v_i = 1, i \in \{0, 1, \dots\}$ pour tous les états v_i de V.

   Soient deux nœuds Bayésien A et B. A la cause et B l'effet (Figure 2.6). Chacun des nœuds A et B ont respectivement deux états (a_0, a_1) et (b_0, b_1). Le polynôme qui définit cet exemple illustratif (A cause B) est représenté par l'équation 2.5

Figure 2.6: *BN de A cause B ;*

$$f = \lambda_{a0}\theta_{a0}\lambda_{b0}\theta_{b_0|a_0} + \lambda_{a0}\theta_{a0}\lambda_{b1}\theta b_1|a_0 + \lambda_{a1}\theta_{a1}\lambda_{b0}\theta b_0|a_1 + \lambda_{a1}\theta_{a1}\lambda_{b1}\theta b_1|a_1 \tag{2.5}$$

Le polynôme f permet de calculer les probabilités des évidences P(e). Par exemple dans le cas où l'état de la variable A est a_0 et l'état de la variable B est b_1, alors le calcul P(e) est comme suit (équation 2.6) :

$$\begin{aligned} P(e) &= f(\lambda_{a0} = 1, \lambda_{a1} = 0, \lambda_{b0} = 0, \lambda_{b1}) \\ &= \lambda_{a0}\theta_{a0}\lambda_{b1}\theta_{b_1|a_0} \\ &= \theta_{a0}\theta_{b_1|a_0} = 0.3 * 0.9 = 0.27. \end{aligned} \tag{2.6}$$

((a)) *Nœud A*

A	θ_{A}
a_0	$\theta_{a0} = 0.3$
a_1	$\theta_{a1} = 0.7$

((b)) *Nœud B*

A	B	$\theta_{\mathrm{B\|A}}$
a_0	b_0	$\theta_{b_0\|a_0} = 0.1$
a_0	b_1	$\theta_{b_1\|a_0} = 0.9$
a_1	b_0	$\theta_{b_0\|a_1} = 0.8$
a_1	b_1	$\theta_{b_1\|a_1} = 0.2$

Tableau 2.4: *CPT des nœuds A et B.*

- Factorisation du polynôme : simplification du polynôme afin de construire facilement l'arbre AC. Les nœuds feuilles de l'arbre AC représentent les λ et les θ, Les nœuds intermédiaires sont des opérations de multiplications (*) et d'additions (+). L'équation 2.7 montre le résultat de la factorisation du polynôme f de l'équation 2.5.

$$\begin{aligned} f &= \lambda_{a0}\theta_{a0}(\lambda_{b0}\theta_{b_0|a_0} + \lambda_{b1}\theta_{b_1|a_0}) \\ &+ \lambda_{a1}\theta_{a1}(\lambda_{b0}\theta_{b_0|a_1} + \lambda_{b1}\theta_{b_1|a_1}). \end{aligned} \tag{2.7}$$

2. Phase de calcul des probabilités avec l'AC : le calcul des probabilités conditionnelles P(V | e) (V une variable et e les évidences) se fait par dérivée partielle en s'appuyant sur le théorème suivant :

Théorème 2.2.2 *Soit f le polynôme décrivant un réseau Bayésien. Pour chaque variable V et toutes évidences e :*
$\frac{\partial f}{\partial \lambda_v}(e) = P(v, e - V)$. *($e - V$: les évidences sans la variable V).*
Ce qui implique : $P(v|e) = \frac{1}{f(e)}\frac{\partial f}{\partial \lambda_v}(e)$ *($P(v|e) = \frac{P(v,e)}{P(e)}$.)*

Le calcul de la probabilité de P(A = a_0 | B = b_0) se fait comme suit :

$$P(A = a_0|B = b_0) = \frac{P(A = a_0, B = b_0)}{P(B = b_0)}. \tag{2.8}$$

$$\begin{aligned} P(B = b_0) &= P(e) = f(\lambda_{a0} = 1, \lambda_{a1} = 1, \lambda_{b0} = 1, \lambda_{b1} = 0) \\ &= \lambda_{a0}\theta_{a0}\lambda_{b0}\theta_{b_0|a_0} + \lambda_{a1}\theta_{a1}\lambda_{b0}\theta b_0|a_1 \\ &= 0.3 * 0.1 + 0.7 * 0.8 = 0.59. \end{aligned} \tag{2.9}$$

$$\begin{aligned} P(A = a_0, B = b_0) &= \frac{\partial f}{\partial \lambda_{a0}}(b_0) = f(\lambda_{a0} = 1, \lambda_{a1} = 0, \lambda_{b0} = 1, \lambda_{b1} = 0) \\ &= \lambda_{a0}\theta_{a0}\lambda_{b0}\theta_{b_0|a_0} = 0.3 * 0.1 = 0.03. \end{aligned} \tag{2.10}$$

Donc P(A = a_0 | B = b_0) = 0.03 / 0.59 = 0.05084 (=5.084%)

Nous constatons que le calcul d'inférence par circuit arithmétique croit en fonction du nombre de nœuds (structure du BN). Une méthode basée sur l'AC est proposée par [Darwiche, 2003] afin de calculer tous les nœuds à la fois. Aussi, l'inférence par AC traduit un réseau Bayésien en un arbre arithmétique composé d'opérations de multiplications et d'additions. Cette structure en AC permet une implémentation matérielle de l'inférence en exploitant les optimisations de parallélismes et de ressources [Zermani et al., 2017b] contrairement à l'inférence par arbre de jonction ; car la moralisation et la triangulation sont compliquées à implémenter en matériel.

2.2.3 Apprentissage dans les réseaux Bayésiens

Comme exposé dans la section précédente, le calcul d'inférence par arbre de jonction ou par circuit arithmétique, s'appuie sur les observations et les paramètres initiaux des nœuds du réseau Bayésien. Les paramètres d'un réseau Bayésien sont les valeurs de probabilités renseignées dans les tables de probabilités conditionnelles (CPT) associées à chaque nœud du BN.

Les valeur de probabilités initiales indiquées par les CPT sont généralement obtenues par expertise (c'est-à-dire par un expert du système que l'on souhaite représenter par un BN). Cependant, ce n'est pas toujours le cas pour tous les domaines et les systèmes. Pour cette raison, il est nécessaire d'aborder l'apprentissage des paramètres d'un BN [Naïm et al., 2007]. Il existe différentes méthodes d'apprentissage des paramètres pour BN en considérant des données complètes [Grossman and Domingos, 2004] ou des données incomplètes [Friedman, 1998]. Dans la suite de cette section, nous allons présenter l'apprentissage de paramètres dans le cas de données complètes et incomplètes.

Apprentissage à partir de données complètes

Lorsque nous sommes dans un contexte où toutes les données sont observées (contexte de données complètes), il existe trois méthodes permettant d'apprendre les probabilités d'un BN à partir des données observées complètes. Ces méthodes sont EAP (Espérance A Posteriori), MAP (Maximum A Posteriori) et MV (Maximum de Vraisemblance) [Naïm et al., 2007]. L'apprentissage Bayésien consiste à trouver les paramètres θ les plus probables.

1. Maximum de vraisemblance (MV) : revient à estimer la probabilité d'un événement par la fréquence d'apparition de l'événement dans la base de données. Cette estimation est calculée par l'équation 2.11 :

$$P(V_i = v_k | pa(V_i) = v_j) = \theta_{i,j,k}^{MAP} = \frac{N_{i,j,k}}{\sum_k N_{i,j,k}} \tag{2.11}$$

   Où $N_{i,j,k}$ indique le nombre d'occurrence de l'événement dans la base de données pour lesquels V_i est dans l'état v_k et ses parents sont dans l'état v_j.

2. Maximum a posteriori (MAP) : l'approche de maximum a posteriori consiste à trouver les valeurs θ en fonction des données observées complètement et en utilisant aussi une distribution de Dirichlet. L'avantage de la distribution de Dirichlet est qu'elle facilite l'expression de la loi a posteriori des paramètres (P(θ | D), D étant l'ensemble des données) [Ghosh, 1996]. L'équation de l'approche de MAP est donnée comme suit (2.12) :

$$P(V_i = v_k | pa(V_i) = v_j) = \theta_{i,j,k}^{MAP} = \frac{N_{i,j,k} + \alpha_{i,j,k} - 1}{\sum_k (N_{i,j,k} + \alpha_{i,j,k} - 1)} \tag{2.12}$$

   Où $\alpha_{\mathrm{i,j,k}}$ représente le paramètre de Dirichlet.

3. Espérance a posteriori (EAP) : au lieu de calculer le maximum a posteriori, il est possible de calculer l'espérance a posteriori des paramètres θ du BN (équation 2.13).

$$P(V_i = v_k | pa(V_i) = v_j) = \theta_{i,j,k}^{EAP} = \frac{N_{i,j,k} + \alpha_{i,j,k}}{\sum_k (N_{i,j,k} + \alpha_{i,j,k})} \tag{2.13}$$

Parmi ces trois méthodes, la méthode la plus utilisée pour l'apprentissage des paramètres d'un BN dans le cas de données complètes est la méthode de calcul du Maximum de Vraisemblance (MV).

Apprentissage à partir de données incomplètes

Dans les applications réelles, par exemple l'évaluation de l'état de santé d'un système autonome pendant la réalisation d'une mission, les bases de données sont souvent incomplètes. Ce manque de données peut résulter d'une panne de capteur, manque d'expertise ou imprécision de la mesure des variables. Dans ce cas il est possible d'apprendre les paramètres du BN à l'aide de l'algorithme EM (*Expectation et Maximisation*) [Naïm et al., 2007], [Tembo et al., 2016].

Description de l'algorithme EM

EM est un algorithme classique utilisé pour l'estimation des paramètres θ (probabilités) d'un BN sachant que les données ont été observées de manière incomplète (manquantes). EM est un algorithme itératif, son fonctionnement est basé sur deux étapes principales : **l'étape E** et **l'étape M** [Lauritzen, 1995].

— **Étape E** : estimation des valeurs manquantes à partir des paramètres θ à l'instant t. Pour chaque nœud V du réseau Bayésien et pour chaque échantillon de la base de données, la probabilité conditionnelle P($\mathrm{V_{manquants}}$ | $\mathrm{V_{mesurés}}$) est calculée, V étant un nœud du BN.
Le calcul de ces probabilités s'appuie sur le mécanisme d'inférence utilisant

généralement l'inférence par *arbre de jonction* (Section 2.2.2).

$$N(V_i = v_k, pa(V_i) = v_j | V^0, \theta^r) = \sum_{t=1..T} \sigma^t_{i,j,k}, \forall j, k. \tag{2.14}$$
$$\sigma^t_{i,j,k} = P(V_i = v_k, pa(V_i) = v_j | V^0, \theta^r)$$

Où V^0 représente les évidences de l'ensemble des données mesurées et $pa(V_i) = v_j$ indique l'état du nœud parent de V_i.

— **Étape M** : ré-estimation des paramètres à partir des résultats de l'étape E par *maximum de vraisemblance* ou par *maximum a posteriori* (MAP qui est une approche Bayésienne). L'approche la plus répondue est le maximum de vraisemblance.

$$\theta^{r+1}_{i,j,k} = \frac{N_{i,j,k}}{\sum_k N_{i,j,k}} \tag{2.15}$$

2.3 Décision pour la planification de mission

Avec l'évolution des systèmes embarqués et de la robotique, de plus en plus de missions sont réalisées avec des systèmes autonomes (robot, drones, etc.). Généralement ces missions se déroulent dans des environnements incertains. Pour garantir l'autonomie et l'adaptabilité de la réalisation de la mission par le système autonome, il est nécessaire d'utiliser des mécanismes ou modèles de prise de décisions pour la planification de mission. Il existe différents modèles permettant de décrire et de contrôler une mission d'un système autonome en prenant des décisions en fonction des événements observés. Parmi ces modèles, nous retrouvons les réseaux de Petri [Roldán et al., 2015], les systèmes multi-critères, le Processus de Décision Markovien (MDP) [Cassandras and Lafortune, 2009], le Processus de Décision Markovien Partiellement Observable (POMDP) [Chanel et al., 2013], langage du diagramme d'influence dynamique relationnelle (Relational Dynamic Influence Diagram Language (RDDL)) [Sanner, 2010], etc.

Dans cette section, nous allons nous intéresser aux deux modèles les plus utilisés dans le domaine de la robotique et des systèmes embarqués, à savoir les MDPs [Tipaldi and Glielmo, 2017], [Puterman, 2014] et les POMDPs [Vanegas et al., 2017], [Chanel et al., 2013]. Ces deux modèles sont préconisés lorsque le système évolue dans un environnement avec des incertitudes (environnementale, défaillance, etc.).

2.3.1 Processus de Décision Markovien

Les MDPs sont des chaînes de Markov décisionnelles. Un MDP est un modèle stochastique probabiliste qui permet la description d'un système et le contrôle de

ce dernier en fonction des événements. Les domaines d'applications des MDPs sont multiples, par exemple :

- Recherche opérationnelle : les premières applications des MDP ont été le contrôle des stocks et la publicité ciblée [Howard, 2002].
- Domaine des Finances : la finance est un domaine où les MDPs sont très utilisés car le processus de décision doit intégrer les coûts à long termes [Bäuerle and Rieder, 2011].
- Théorie du contrôle : les MDPs sont utilisés pour décrire et tester le comportement des systèmes électriques ou mécaniques par les ingénieurs. Par exemple, tester les caractéristiques des moteurs des systèmes [Stone et al., 2005].
- Neurosciences : la psychologie comportementale s'intéresse à l'apprentissage du renforcement qui est un domaine de l'intelligence artificielle fondé sur les MDPs [Rao, 2010].
- Intelligence artificielle : l'utilisation des MDPs apparaît fortement dans le domaine de la robotique pour modéliser et vérifier le comportement des système autonomes comme les robots.

Les MDPs dans les différents domaines cités partagent les même caractéristiques, à savoir :

- Pour chaque problème traité, le système MDP doit respecter l'enchaînement d'une suite d'états.
- Les transitions entre les états se produisent en réponse à une suite d'actions prise par un agent à différentes étapes de la décision.
- À chaque étape de la décision, l'état du système est complètement observable par l'agent.
- Chaque action exécutée dans un état peut conduire le système vers plusieurs états possibles avec différentes probabilités de transitions connues par l'agent.
- Le choix d'une action produit une récompense immédiate.

Définition formelle d'un MDP

Définition 2.3.1 *Un MDP est représenté par un tuple <S, A, D, T, R> [Kolobov, 2012], où :*

- S : appelé *ensemble d'états*, représente l'ensemble fini de tout les états possibles du système.
- A : appelé *ensemble d'actions*, indique l'ensemble fini de toutes les actions que l'agent peut choisir afin de réaliser sa mission.
- D appelé *time steps*, représente une séquence de temps fini ou infini dans laquelle une décision doit être prise.

— T : S x A x S x D → [0, 1] : est la fonction de transition spécifiée par une probabilité T(s_1, a, s_2, t) ; T exprime la probabilité d'atteindre s_2 en choisissant l'action a lorsque l'agent est dans l'état s_1 à l'instant t.

— R : S x A x S x D → $\mathbb{R}$: est la fonction de coût (ou de récompense). Une récompense R(s_1, a, s_2, t) est générée lorsque l'action a permettant le passage de s_1 vers s_2 est choisie par le système en étant dans l'état s_1 à l'instant t.

Un MDP est un mécanisme de prise de décision qui permet d'une part la description du comportement d'un système et d'autre part de proposer une solution, en terme de décision, pour le système en fonction des entrées reçues par le MDP. La résolution d'un MDP consiste à trouver une solution optimale globale qui respecte les critères fixés pour le système décrit par le MDP. La Figure 2.7 illustre le modèle MDP qui respecte la définition 2.3.1.

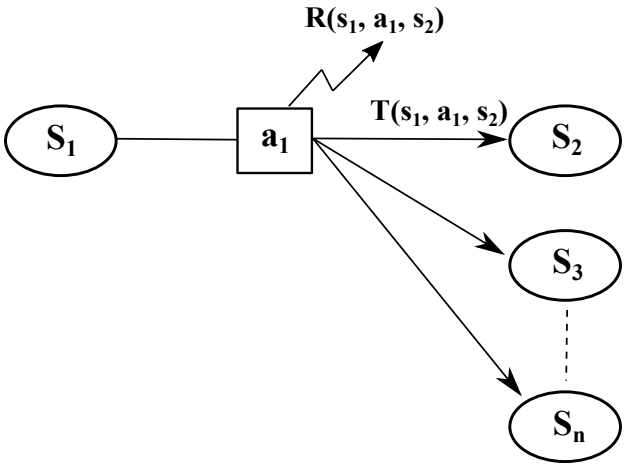


Figure 2.7: *Modèle MDP.*

Définition 2.3.2 *SOLUTION OPTIMALE*
La recherche d'une solution optimale pour un MDP consiste à trouver une politique π. La politique π représente l'ensemble d'actions choisi par le processus de décision c.à.d la décision qui sera exécutée par l'agent. Cette politique π est calculée en fonction du critère de coût dans le but de maximiser la fonction de coûts (ou de récompenses) sur un horizon fini (équation 2.16) ou infini (équation 2.17).

$$V_\pi(S(i)) = E_\pi \left[\sum_{k=0}^{N} R(S(k), A(k)) \right], \tag{2.16}$$

$$V_\pi(S(i)) = E_\pi \left[\sum_{k=0}^{\infty} R(S(k), A(k)) \right], \tag{2.17}$$

où $V_\pi(S(i))$ est le coût généré par la politique π sur $N+1$-step avec l'état initial $S(0)$, E est l'opérateur d'espérance et $R(S(k), A(k))$ est la récompense (coût) générée par l'action $A(k)$ en étant dans l'état $S(k)$.

Définition 2.3.3 *HORIZON FINI*
Un MDP sur horizon fini est décrit avec les même propriétés définies plus haut (def 2.3.1) sauf que l'intervalle de temps D est connu et fini sur [0, T_{max}], $T_{max} < \infty$.

Définition 2.3.4 *HORIZON INFINI*
Un MDP sur horizon infini est décrit avec les même propriétés définies plus haut (def 2.3.1) sauf que l'intervalle de temps D est infini.

Résolution d'un MDP sur horizon fini et infini

La résolution d'un MDP sur un horizon fini consiste à trouver la politique optimale π à partir d'un état initial s_0 en maximisant une fonction de coût. Cela revient à chercher les valeurs de fonction (values function V) associées à la politique calculée à chaque step de l'intervalle temps D fini. Le calcul de la politique sur horizon fini est donnée comme suit :

$$\begin{aligned} V^*(S,t) &= max_\pi V_\pi(S,t) \\ V^*(S,t) &= max_{a\in A}\left[\sum_{s'\in S} T(s,a,s',t)\left[R(s,a,s',t) + V^*(s',t+1)\right]\right] \\ \pi^*(S,t) &= argmax_{a\in A}\left[\sum_{s'\in S} T(s,a,s',t)\left[R(s,a,s',t) + V^*(s',t+1)\right]\right] \end{aligned} \tag{2.18}$$

Dans le cas d'un horizon infini, l'intervalle de temps D est inconnu et surtout infini, donc si l'équation 2.19 est utilisée, le processus de résolution ne convergera pas vers une politique optimale car le calcul de la politique se fera à l'infini. Pour remédier à cela, un facteur *discount* γ est intégré à l'équation comme suit :

$$\begin{aligned} V^*(S) &= max_{a\in A}\left[\sum_{s'\in S} T(s,a,s')\left[R(s,a,s') + \gamma * V^*(s')\right]\right] \\ \pi^*(S) &= argmax_{a\in A}\left[\sum_{s'\in S} T(s,a,s')\left[R(s,a,s') + \gamma * V^*(s')\right]\right] \end{aligned} \tag{2.19}$$

L'intégration du facteur γ prend ses valeurs dans l'intervalle [0 1]. Dans la résolution d'un MDP sur horizon infini, il permet à l'algorithme de résolution de converger vers une politique optimale en calculant les coûts futurs espérés (discounted rewards) des actions, définies par le MDP, à partir de l'état initial s_0.

Exemple d'illustration

Nous prenons comme exemple d'illustration un robot (terrestre) qui se déplace sur une grille [Russell and Norvig, 2016]. La grille est divisée en plusieurs cases. L'objectif du robot est de se déplacer sur la grille à partir de la case de départ (s_0) et essayer d'atteindre la case objectif, représentée par le symbole diamant sur la Figure 2.8. Deux obstacles sont présent sur la grille, le premier obstacle est représenté par un mur et le second obstacle est représenté par un feu. Le robot ne peut pas, par conséquent, passer la case contenant l'obstacle mur. Si le robot passe la case contenant le feu, il perdra une récompense de (-100). Par contre, si le robot atteint la case objectif avec succès alors il gagnera une récompense de (+100). Pour se déplacer, le robot dispose de trois actions possibles : nord, est et ouest. La distribution de probabilités est comme suit : L'action choisie a une probabilité de 0.8 (80%), les 20% restante sont distribuée équitablement entres les deux actions non choisies.

Maintenant que l'énoncé du problème est défini, nous pouvons déduire les éléments nécessaires à la construction du MDP associé à l'exemple de la grille.

— Ensemble des états : cases de la grille.

— Ensemble des actions : constitué de trois actions qui sont 'nord', 'est' et 'ouest'.

— fonction de transition : l'action choisie a une probabilité de 80% et les deux autres actions ont chacune une probabilité de 10%.

— Fonction de coûts : une récompense de (+100) est générée si l'objectif est atteint et une récompense de (-100) est générée si le robot passe par la case feu.

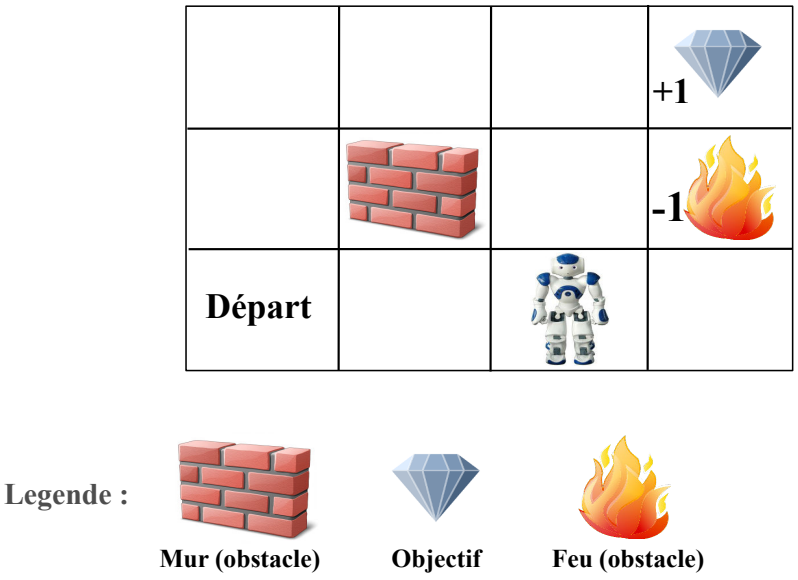


Figure 2.8: *Exemple de planification de mission de robot sur une grille.*

La Figure 2.9 illustre la distribution et la représentation du choix de l'action

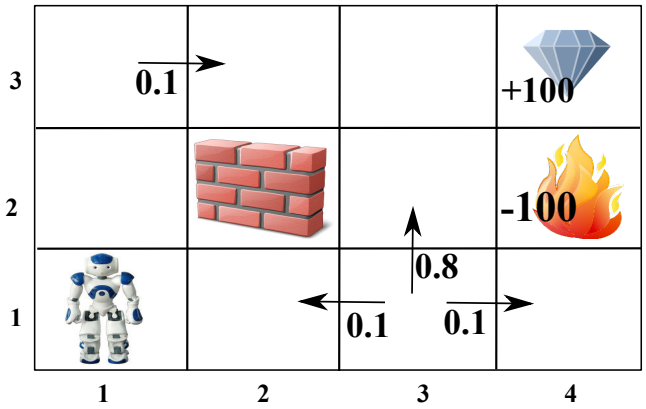


Figure 2.9: *illustration du choix de l'action.*

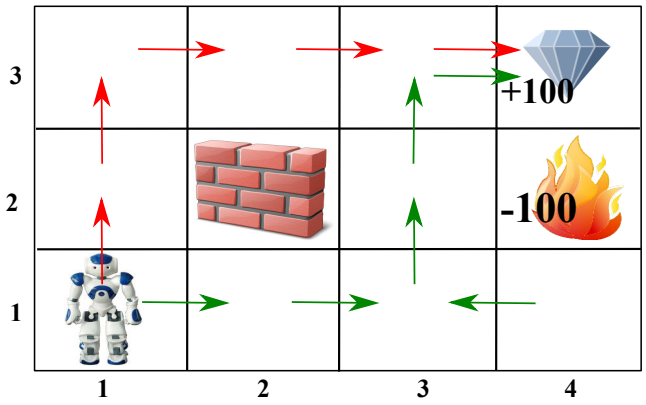


Figure 2.10: *Politiques possibles pour atteindre l'objectif.*

adéquate par le robot. La case (1, 3) représente le cas classique où le robot à le choix entre les trois actions possibles (nord, est et ouest) avec leurs probabilités respectives. Cependant, il existe des cas particuliers comme la case (3, 1) et la case (1, 2). Dans le cas de la case (3, 1), seulement l'action 'est' peut être choisie. Pour la case (1, 2), le robot a le choix entre les actions 'est' et 'ouest' seulement car le choix de l'action 'nord' conduira à un obstacle qui n'est pas une solution optimale.

Rappelons que le but du robot est de trouver le chemin le conduisant à l'objectif (case (3, 4)). En fonction de l'action choisie à l'état initial de départ et exécutée par le robot, le MDP décrivant le problème de la grille produit deux politiques possibles π_1 et π_2. La Figure 2.10 montre ces deux politiques. La première politique π_1 est représentée par les flèches rouges, cette politique est produite par le MDP et exécutée par le robot dans le cas où l'action de départ choisie est 'nord'. Dans le cas où l'action de départ est 'est' alors c'est la politique π_2 (flèches vertes) qui est exécutée.

2.3.2 Processus de Décision Markovien Partiellement Observable

Les POMDP (Partially Observable Markov Decision Process) sont une variante des MDP. Un modèle POMDP est généralement utilisé lorsque l'agent du problème à résoudre a une incertitude sur la connaissance des différents états du système. En d'autres termes, les états du système ne sont pas complètement observables [Ponzoni Carvalho Chanel et al., 2013].

Définition 2.3.5 *Définition d'un POMDP*
Un POMDP est tuple <S, A, O, T, Z, R, γ>, où :

- *S : représente l'ensemble des états du système.*
- *A : représente l'ensemble des actions exécutées par le système.*
- *O : indique l'ensemble des observations.*
- *T : est la fonction de transition T(s,a,s'), probabilité de passer de s à s' en exécutant l'action a.*
- *Z : est la fonction de distribution qui décrit la probabilité d'avoir l'observation o à partir de l'état s après avoir choisie l'action a.*
- *R : représente le coût généré par une action a à chaque état s.*
- *γ : facteur discount.*

La particularité des POMDPs, contrairement aux MDPs, est que l'état du système n'est pas représenté par une valeur unique mais par une distribution de probabilité de tous les états possibles dans l'espace d'état défini pour le système à un instant donné. Cela est appelé état de croyance ou 'belief-state' et est noté par la croyance b. Dans une modélisation avec un POMDP, les états sont partiellement ou pas du tout observables. Par conséquent, le système reçoit que les observations $o \in O$ de l'état du système global. La solution produite par un POMDP est une politique $\pi : \mathcal{B} \rightarrow \mathcal{A}$ qui représente les actions choisies à partir des 'belief-states' ($\mathcal{B}$ est l'ensemble des belief-states). L'ensemble des 'belief-states' est mis à jour à chaque fois qu'une nouvelle observation est reçue. Par conséquent, l'objectif du POMDP est de trouver la politique π, en supposant b l'état courant, en maximisant les valeurs de fonctions V lorsqu'une séquence d'actions et d'observations est suivie. Quant aux valeurs de probabilités des transitions, elles sont souvent estimées à l'aide du théorème de Bayes [Ponzoni Carvalho Chanel et al., 2013], [Vanegas et al., 2017] ou bien en utilisant le principe de la théorie d'information [Van Nguyen et al., 2017].

Bilan

Après avoir présenté le principe des MDPs et POMDPs, nous constatons que les deux modèles sont quasi identiques dans leurs façons d'aborder un problème de prise de décision. La différence entre les deux réside dans les points suivants :

— Les MDPs sont utilisés dans un contexte où les états du système décrit sont complètement observables contrairement aux POMDPs qui sont utilisés lorsque les états du système sont partiellement observables.

— La résolution d'un MDP consiste à trouver la politique optimale à exécuter par le système à partir d'un état initial.

— La résolution d'un POMDP consiste dans un premier temps à trouver l'ensemble des états probables courant du système et ensuite calculer la politique optimale à partir de ces états probables. Par conséquent, une décision ne peut être produite (en réponse à un événement) seulement si l'ensemble des états probables a été mis à jour.

Le calcul de la décision par les deux modèles (MDPs et POMDPs) croit en fonction du nombre d'états et d'actions défini pour le système à résoudre. Cependant, la complexité des POMDPs peut être plus élevée que celle des MDPs ; car à chaque exécution du POMDP, il est nécessaire de recalculer l'ensemble des états courant probables (en plus de la mise à jour des valeurs de transition) avant de produire la décision.

Dans les chapitres qui suivent, nous allons nous intéresser plus en détails à l'application d'un MDP dans le cas de planification de mission de drone autonome dans un contexte avec incertitudes. Nous aborderons les MDPs car les états de la mission de drone étudiée sont complètement observables.

2.3.3 Apprentissage par renforcement

Les mécanismes de prise de décision, comme les MDPs et POMDPs, génèrent une décision en fonction des valeurs de probabilités de transitions et des valeurs de coûts associées aux actions qui seront exécutées par le système. Parfois, ces valeurs de probabilités et de coûts ne sont pas connues au préalable. Dans ce cas, il existe des méthodes et algorithmes permettant l'apprentissage des valeurs de coûts (rewards). L'algorithme le plus connu et le plus utilisé pour le 'reinforcement learning' est le *Q-learning* [Sigaud and Buffet, 2008].

Principe du Q-learning

Une fois le modèle MDP décrivant le système ou le problème à résoudre construit, l'algorithme Q-learning peut être appliqué. Le principe de l'algorithme Q-learning repose sur le fait de mettre à jour de manière itérative la fonction de valeur de récompense courante Q_i pour le couple (s_i, a_i) à la suite de chaque transition (s_i, a_i, s_{i+1}, r_i).

s_i représente l'état courant, a_i est l'action choisie à l'état s_i, s_{i+1} l'état suivant atteignable avec a_i depuis s_i et enfin r_i représente la récompense (coût) immédiate générée par le choix de a_i. Le calcul des valeurs de Q (Q-values) se fait avec l'équa-

tion 2.20.

$$Q(s,a) += \alpha\,[r + \gamma(max_a Q(s',a')) - Q(s,a)] \tag{2.20}$$

Où, α représente le taux d'apprentissage (learning rate), $\alpha \in [0,\ 1]$ et γ le facteur discount.

Exemple d'illustration

Pour illustrer le fonctionnement de l'algorithme Q-learning, nous allons utilisé l'exemple du robot se déplaçant sur une grille. Pour des raisons de simplification, nous considérons une grille sans obstacles. La valeur de α est fixée à 1 et celle du discount γ à 0.5 et aussi les valeurs de récompenses immédiates r valent zéro pour chacune des actions. Les figures suivantes montrent l'application du Q-learning sur l'exemple de la grille.

S1, a12	0
S1, a14	0
S2, a21	0
S2, a23	0
S2, a25	0
S3, a32	0
S3, a36	0
S4, a41	0
S4, a45	0
S5, a54	0
S5, a52	0
S5, a56	0

Tables des Q-values

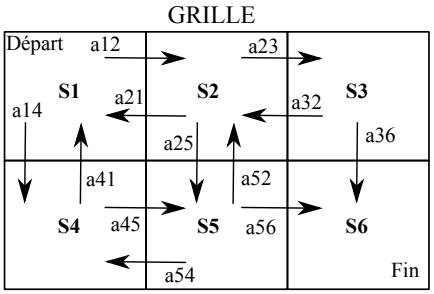
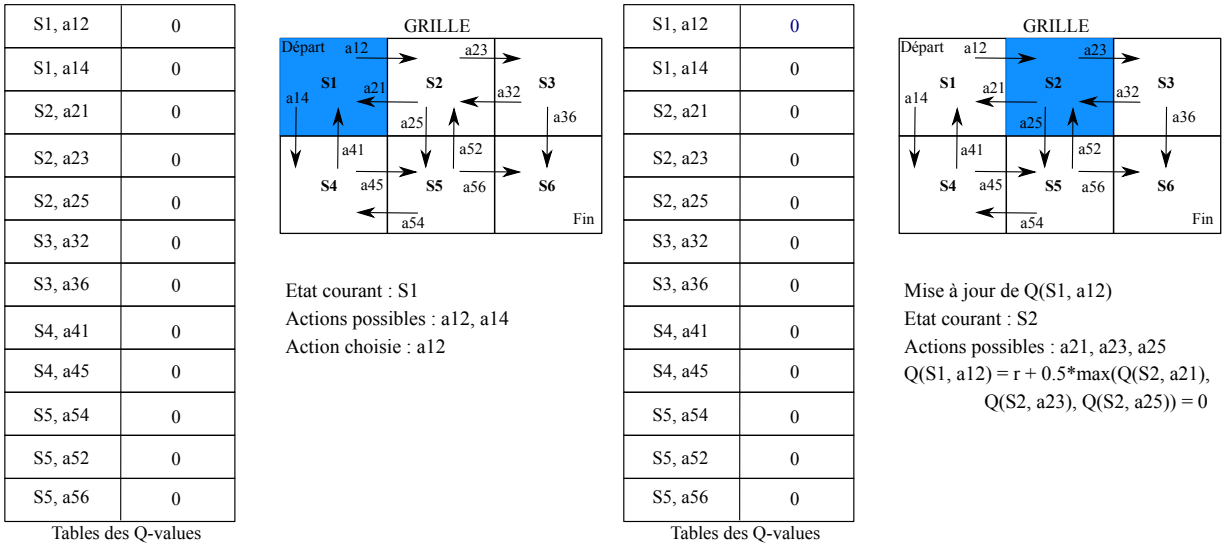


Figure 2.11: *Initialisation de la table des Q-values.*

La Figure 2.11 montre la toute première étape de Q-learning, à savoir l'initialisation des valeurs de coûts dans la table Q-values. La table est initialisée à zéro. Soit s1 l'état de départ, l'algorithme commence à explorer la grille à partir de s1 en choisissant une action aléatoire à partir de chaque état. La Figure 2.12 montre comment le calcul de Q(s, a) est fait à partir de s1 en ayant choisi l'action a12. Pareil pour la Figure 2.13. La Figure 2.14 montre ce qui se passe quand l'agent arrive à son objectif, ici c'est la case s6. Comme r (coût immédiat) vaut zéro pour tout



((a)) *exploration à partir de s1* **((b))** *mise à jour de Q(s1, A12)*

Figure 2.12: *Itération 1 de Q-learning : étape 1.*

déplacement (dans cet exemple) sauf lorsque l'argent atteint son objectif r=100. La première valeur de Q-values est égale à 100 pour le choix de l'action a36 à partir de l'état s3. À partir de là, l'algorithme effectue une deuxième itération pour calculer les coûts des actions qui ont permis d'atteindre la case s6 (l'objectif). Après plusieurs itérations et convergence de l'algorithme, la Figure 2.15 montre les Q-values obtenues.

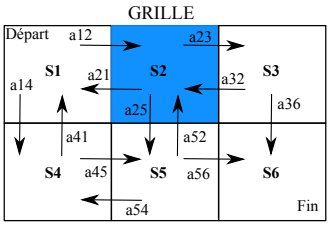
Bilan

L'apprentissage par renforcement par Q-learning est utilisé lorsque l'utilisateur n'a aucune information sur le MDP décrivant le problème à résoudre. Ni la structure, ni les paramètres (probabilités de transitions et rewards) du MDP ne sont connus. Le Q-learning permet donc d'une part d'apprendre les paramètres du MDP et d'autre part de résoudre le problème totalement inconnu.

Dans la suite de cette thèse, nous n'allons pas appliqué l'apprentissage par Q-learning car le modèle que nous proposons (modèle BFM) permet d'estimer les paramètres du MDP ; notamment les probabilités de transitions à l'aide des BNs pour le diagnostic (plus de détails dans les chapitres qui suivent).

S1, a12	0
S1, a14	0
S2, a21	0
S2, a23	0
S2, a25	0
S3, a32	0
S3, a36	0
S4, a41	0
S4, a45	0
S5, a54	0
S5, a52	0
S5, a56	0

Tables des Q-values

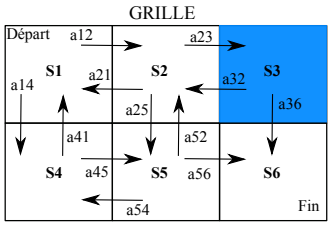


Etat courant : S2
Actions possibles : a21, a23, a25
Action choisie : a23

((a)) *exploration à partir de s2*

S1, a12	0
S1, a14	0
S2, a21	0
S2, a23	0
S2, a25	0
S3, a32	0
S3, a36	0
S4, a41	0
S4, a45	0
S5, a54	0
S5, a52	0
S5, a56	0

Tables des Q-values



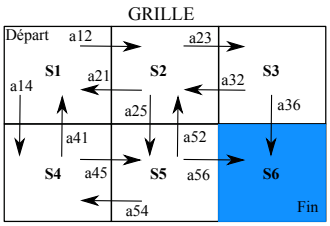
Etat courant : S3
Actions possibles : a32, a36
Misa à jour de Q(S2, a23) :
Q(S2, a23) = r + 0.5*max(Q(S3, a32), Q(S3, a36))
= 0

((b)) *mise à jour de Q(s2, A23)*

Figure 2.13: *Itération 1 de Q-learning : étape 2.*

S1, a12	0
S1, a14	0
S2, a21	0
S2, a23	0
S2, a25	0
S3, a32	0
S3, a36	100
S4, a41	0
S4, a45	0
S5, a54	0
S5, a52	0
S5, a56	0

Tables des Q-values

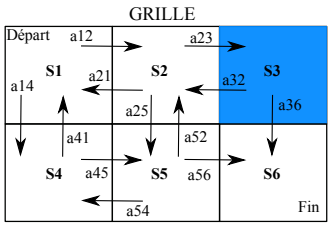


Etat final atteint : S6
Mise à jour de Q(S3, a36) :
Q(S3, a36) = r = 100

((a)) *exploration et mise à jour de Q(s3, a36)*

S1, a12	0
S1, a14	0
S2, a21	0
S2, a23	50
S2, a25	0
S3, a32	0
S3, a36	100
S4, a41	0
S4, a45	0
S5, a54	0
S5, a52	0
S5, a56	0

Tables des Q-values



Etat courant : S3
Actions possibles : a32, a36
Mise à jour de Q(S2, a23) :
Q(S2, a23) = r + 0.5*max(Q(S3, a32), Q(S3, a36))
= 0 + 0.5*100 = 50

((b)) *exploitation des premières Q-values trouvées*

Figure 2.14: *Itération 1 de Q-learning : étape 3.*

S1, a12	25
S1, a14	25
S2, a21	12.5
S2, a23	50
S2, a25	25
S3, a32	25
S3, a36	100
S4, a41	12.5
S4, a45	50
S5, a54	25
S5, a52	25
S5, a56	100

Tables des Q-values

Figure 2.15: *Q-values obtenues après convergence de l'algorithme Q-learning.*

2.4 Version embarquée de la planification de mission

Les missions assignées aux systèmes autonomes sont de plus en plus complexes et requièrent l'exécution d'un grand nombre d'applications embarquées. Par conséquent, cela demande de faire face aux contraintes de performances, souvent nécessaire pour le bon déroulement de la mission. Pour répondre aux contraintes de performances, les développeurs ont recours à des accélérateurs matériels.

Les accélérateurs matériels sont utilisés pour alléger et décharger le processeur lorsque celui-ci est occupé totalement ou bien les tâches qui s'exécutent sont coûteuses. Il existe différents types d'accélérateurs comme les GPU (processeurs graphiques), les ASIC, et les FPGA (circuits logiques programmables) que nous utilisons dans le cadre du projet. Parmi les travaux de recherche fait sur la planification de mission de drones autonomes, nous citons [Fuller et al., 2014] qui propose d'implémenter une interface MAVLink sur FPGA afin de communiquer avec le module de navigation du drone utilisée. [Kok et al., 2012] propose une implémentation matérielle sur FPGA du module de navigation basé sur un algorithme génétique pour un drone autonome. Le module de navigation servirait à produire les points de trajectoire (points GPS) à suivre. Les travaux de [Bethke et al., 2008] traite de la planification de mission à l'aide de MDP et propose une version embarquée (en logicielle) du solver MDP pour l'estimation des paramètres (probabilités de transitions). Il existe aussi des propositions commerciales alliant un auto-pilote plus un FPGA sur un même support tel que la carte OcPoC de Xilinx [1].

La réalisation d'un accélérateur matériel pour une cible FPGA passe par une description au niveau RTL de l'application que nous souhaitons accélérer. La description RTL est classiquement faite à l'aide de langages de bas niveau tels que Verilog et VHDL, ce qui rend la tâche de développement fastidieuse. Pour faciliter la réalisation d'accélérateurs matériels, il est possible de concevoir l'application avec un langage haut niveau et ensuite utiliser des outils de synthèse haut niveau (HLS) pour générer la description RTL optimisée de l'application souhaitée. Dans la suite nous allons définir les FPGAs ainsi que leurs intérêts. Ensuite, nous présenterons la méthodologie et les outils de conception d'un accélérateur matériel à l'aide de la HLS.

2.4.1 Accélérateur à base de FPGA

Définition 2.4.1 *Un FPGA (Field Programmable Gare Array) est un circuit électronique programmable. Il peut être programmer pour exécuter une application donnée et il a l'avantage et la possibilité d'être re-programmer plusieurs fois.*

1. https://aerotenna.com/ocpoc-xilinx-zynq-supports-px4-autopilot/

Les FPGAs sont composés des éléments [Rose et al., 1990] suivants :

— Blocs logiques programmables.

— Mémoires embarquées.

— Blocs de calcul spécifiques.

— Broches d'entrées/sorties.

— Interconnexions.

Les éléments de base d'un FPGA sont les blocs logiques configurables (CLB). Chaque CLB se compose de bascules Flip-Flop (FF), de Look-Up Table (LUT) et de multiplexeurs. Les LUTs permettent de programmer des fonctions logiques et les FFs sont des supports de stockages. Dans les FPGAs récents, des mémoires BRAM (mémoires à accès direct) ont été ajoutées pour faciliter le stockage des informations. Des DSP ont aussi été ajoutés dans le but d'accélérer les calculs. Les interconnexions permettent de connecter l'ensemble des éléments cités entre eux. Les entrées/sorties quant à elles, gèrent les échanges d'informations avec les modules extérieurs. La Figure. 2.16 montre l'architecture d'un FPGA.

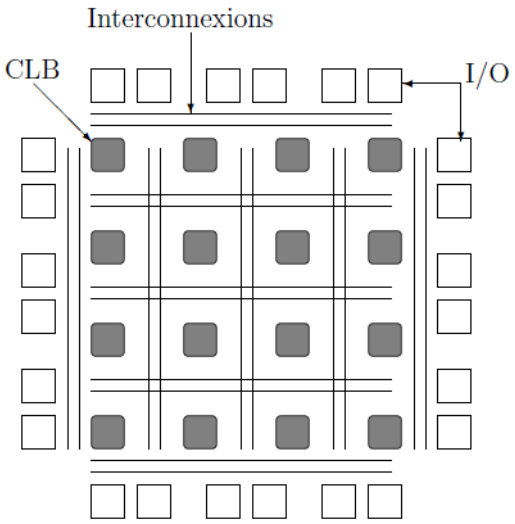


Figure 2.16: *Architecture d'un FPGA.*

2.4.2 Outils de conception d'accélérateurs pour l'embarqué

Méthodes de génération

Plusieurs étapes sont nécessaires à la génération d'accélérateur FPGA pour une application donnée. Deux méthodes permettent cela : méthode classique s'appuyant sur les transferts entre registres (RTL) et la méthode utilisant la synthèse haut niveau (HLS).

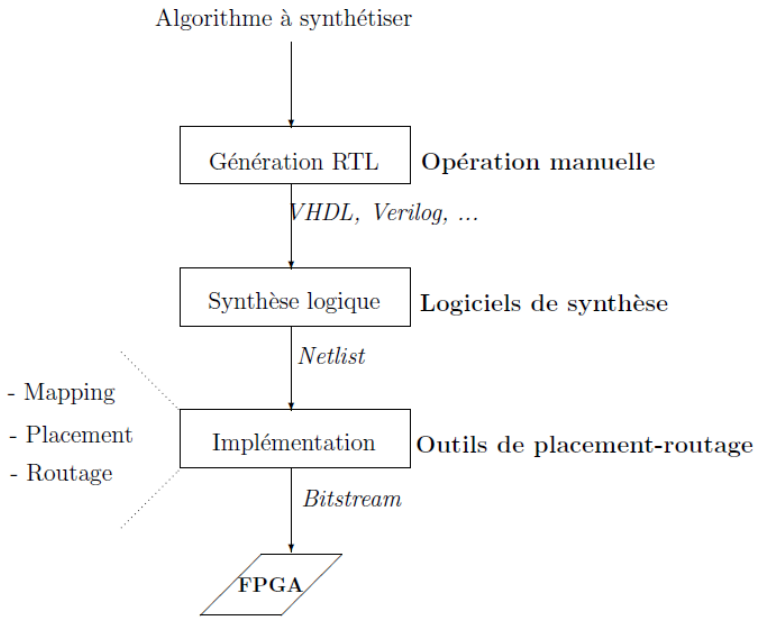


Figure 2.17: *Étapes de conception RTL.*

1. Méthode classique RTL
   La conception classique d'un circuit FPGA s'appuie sur la description RTL ou bien à partir d'une description faite par le développeur dans un langage de description matérielle tel que le VHDL. Le code RTL passe par une phase dite de synthèse logique, où le code RTL est traduit en un ensemble de composants logiques (additionneurs, blocs mémoires, etc.) appelé netlist. Ensuite vient la phase d'implémentation physique pour une cible FPGA. Cette phase passe par le mapping, le placement et le routage. Après toutes ces étapes, un bitstream est généré dans le but de le charger sur le FPGA et ainsi programmer le FPGA. La Figure 2.17 résume ces étapes.
2. Méthode par HLS
   Cette méthode a pour objectif d'automatiser les étapes de conception du RTL [Coussy and Morawiec, 2008]. Contrairement à la méthode classique, la conception par HLS prend en entrée une description haut niveau de l'algorithme de l'application souhaitée. Cette description est faite avec des langages de haut niveau tels que C/C++. En utilisant la synthèse haut niveau, l'utilisateur a la possibilité et la facilité de spécifier différents éléments comme le types des opérateurs, des variable, le partage de ressources, etc. Après la synthèse de haut niveau, l'utilisateur obtient la description de son application en VHDL ou Verilog. Par conséquent, les outils HLS permettent d'explorer rapidement l'espace des solutions en fonctions des contraintes et spécifications souhaitées. La Figure 2.18 montre les étapes de conception RTL à l'aide de la HLS.

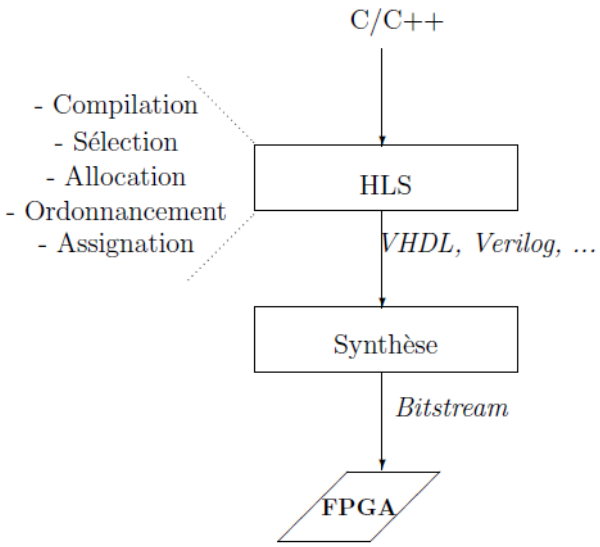


Figure 2.18: *Etapes de conception RTL pour FPGA à l'aide de la HLS.*

La synthèse de haut niveau regroupe en elle plusieurs opérations. La compilation permet la vérification de la syntaxe et la sémantique et traduit la description de l'algorithme en une représentation intermédiaire [Bacon et al., 1994]. La sélection définie les types d'opérateurs à partir de la bibliothèque des composants matériels. L'allocation fixe le nombre d'instances nécessaire dans l'architecture finale. L'ordonnancement permet d'affecter le moment de début de l'exécution de chaque opérateur de la spécification faite à la compilation. Enfin l'assignation assigne une instance d'opérateur à chaque opération et un registre à chaque variable [Cong and Xu, 2008].

3. Conception SoC-FPGA
   Les systèmes embarqués basés sur une plateforme SoC-FPGA (System-On-Chip- FPGA) sont les plus performants (Figure 2.19). Un système SoC-FPGA est un système hybride constitué d'un processeur dur et de logique programmable. Les grandes familles de fournisseurs FPGA sont Xilinx et Altera. La conception RTL pour SoC-PFGA s'appuie sur le Co-Design software/hardware. Cela permet d'exploiter le placement de l'application en software ou en hardware en respectant les contraintes définies lors de la spécification haut niveau. La Figure 2.20 montre les étapes de conception pour SoC.

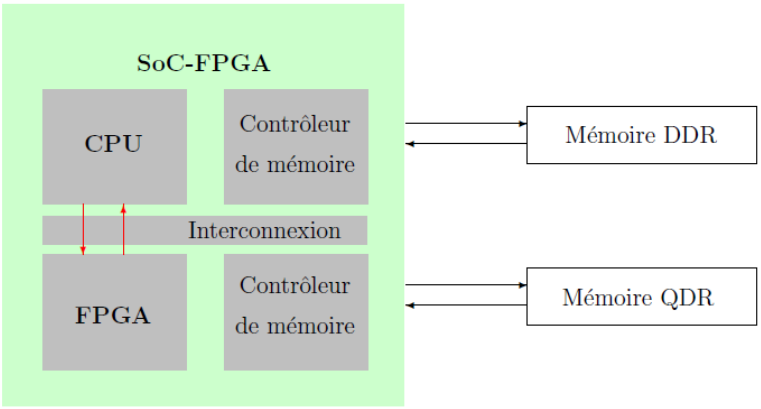


Figure 2.19: *Architecture d'un système SoC-FPGA.*

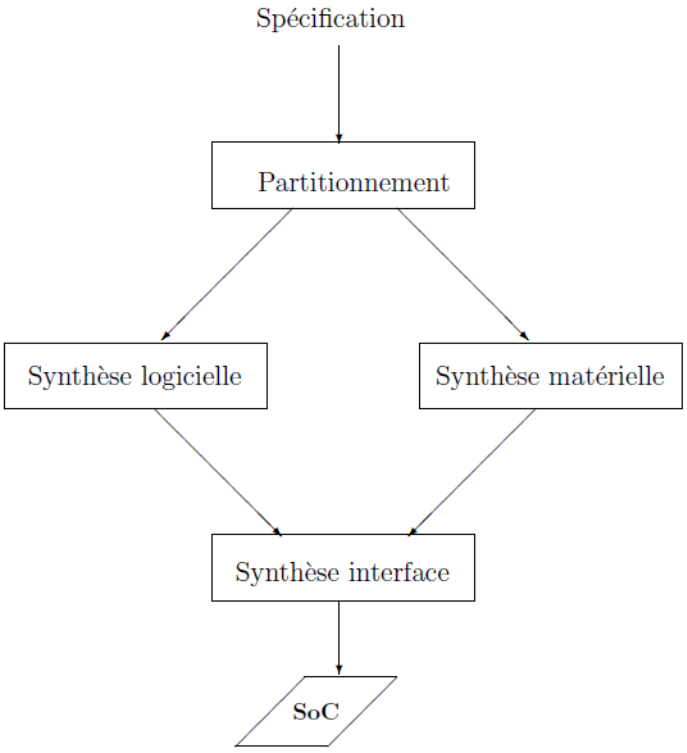


Figure 2.20: *Étapes conception RTL pour SoC.*

2.5 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art sur les différents aspects traités et utilisés tout au long des travaux de recherche détaillés dans les différents chapitres qui suivent. Premièrement, nous avons abordé un état de l'art sur les réseaux Bayésiens dans le cadre du diagnostic depuis l'étape de construction de BN jusqu'à l'inférence et l'évaluation du diagnostic. Ensuite, nous avons présenté les mécanismes de prise de décision, notamment les MDPs qui sont utilisés dans le cadre du contrôle de mission de systèmes autonomes. Enfin, nous avons terminé ce chapitre par la présentation des méthodes et outils dédiées à l'embarqué, et plus précisément les FPGAs.

- Chapitre 3 -

Évaluation de l'état de santé des applications embarquées

Sommaire

3.1 Introduction

Nous présentons dans ce chapitre nos contributions au monitoring et diagnostic pour la prise de décision dans le cadre d'une mission de véhicule autonome. Lors d'une mission, le véhicule autonome rencontre divers aléas qui peuvent engendrer des défaillances au niveau des composants matériels du système mais aussi impacter la qualité de service (QoS) des applications embarquées sur le système autonome. Les types de défaillances peuvent être liés à l'environnement externe (obstacle, etc.) ou bien à l'état interne du système comme la disponibilité des ressources de traitement, espace mémoire, etc.

La Figure 3.1 illustre la démarche proposée. Les réseaux Bayésiens sont générés à partir de l'analyse FMEA. Les données récupérées par les capteurs représentent des observations et sont considérées comme des entrées du réseau Bayésien dans le but de calculer l'état de santé (diagnostic) du système et de ses composants matériels et/ou logiciels. Les paramètres du réseau Bayésien peuvent être inconnus ou incomplet causé par un manque de données d'expertise. Par conséquent, nous faisons appel à l'apprentissage des paramètres d'un réseau Bayésien pour apprendre les paramètres initiaux (probabilités initiales) et améliorer le diagnostic. Par la suite, une décision sur le plan de mission à appliquer est prise par un modèle de décision.

Notre première contribution consiste en la génération d'un modèle de diagnostic à l'aide de réseaux Bayésiens avec deux principaux objectifs :

— Évaluation de la QoS des applications embarquées en fonction du contexte de la mission du véhicule autonome.

— Choisir la version algorithmique de l'application à exécuter en fonction du contexte environnemental et de l'état interne du système en termes d'occupation des ressources de calculs.

La génération du modèle Bayésien est faite à partir d'une méthode d'analyse de défaillances de type FMEA (Analyse des Modes de Défaillances et leurs Effets) [Korayem and Iravani, 2008].

La seconde contribution est la proposition d'une version embarquée pour l'apprentissage des paramètres d'un BN qui repose sur un mécanisme d'inférence qui peut être embarqué sur un véhicule autonome.

Dans la Section 3.2 nous présenterons la génération du réseau Bayésien à partir de l'analyse FMEA et l'adaptation du BN au contexte environnemental et aux ressources du système pour l'évaluation de la QoS des applications embarquées. Ensuite, dans la Section 3.4 nous proposons une version embarquée de la méthode d'apprentissage des paramètres d'un BN en utilisant un mécanisme d'inférence basé sur la compilation AC qui sera exposé dans la Section 3.3. Enfin, nous concluons sur cette partie et nous donnons un aperçu sur le chapitre suivant.

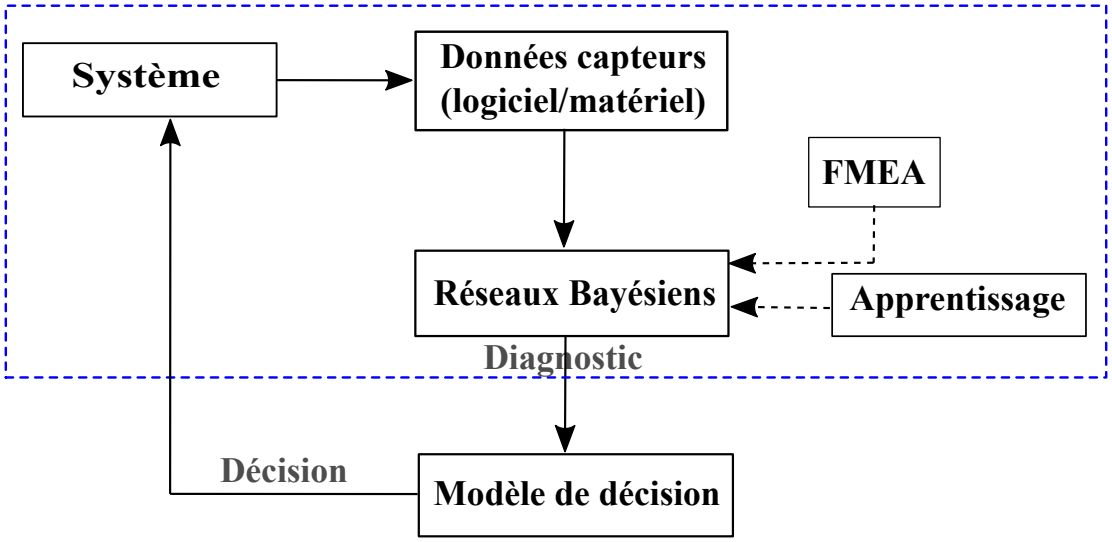


Figure 3.1: *Démarche diagnostic sur l'état de santé du système et ses composants.*

3.2 État de santé des applications à l'aide de réseau Bayésien

Durant une mission de véhicule autonome, l'évaluation de l'état de santé du système et de ses composants (capteurs, énergie, etc.) est nécessaire. D'autre part, l'évaluation de l'état de santé, en terme de QoS (Qualité de Service), des applications embarquées (ex : application de tracking) est souvent sous-estimée voire ignorée. Dans cette section, nous allons nous intéresser à comment évaluer la QoS d'une application à l'aide de réseaux Bayésiens dans le but de gérer au mieux le déroulement d'une mission.

3.2.1 BN pour évaluer la qualité de service des applications

Pour chaque application, nous définissons l'ensemble des types d'erreurs possibles qui peuvent impacter l'état de santé de l'application en diminuant sa QoS. Les types d'erreurs sont monitorés par des capteurs matériels ou logiciels, et par des contextes d'apparitions des types d'erreurs pour renforcer le diagnostic.

Prenons l'exemple de présence de vibrations durant une mission de véhicule autonome. Les vibrations peuvent être monitorées en utilisant le capteur IMU. La fiabilité de l'information peut être renforcée par la présence de vent (contexte d'apparition). Durant une mission, l'erreur de vibrations (causée par du vent par exemple) peut dégrader la QoS de l'application en cours d'exécution, par exemple une application de traitement d'image. Pour corriger l'erreur observée (vibrations) et garder la QoS souhaitée, nous activons l'application de stabilisation comme solution.

La probabilité de la QoS de l'application est calculée en fonction des entrées du BN. Les entrées du BN sont les valeurs des capteurs et des métriques renvoyées par les applications qui représentent les états observables (évidences). Les nœuds 'contexte d'apparition' peuvent aussi être observables pour renforcer la croyance sur le type d'erreur ou l'état de santé des capteurs.

Le rôle de ce modèle Bayésien est, en premier lieu, de détecter le type d'erreur selon le contexte observé et ensuite évaluer la probabilité de la QoS de l'application.

La Figure 3.2 montre le BN correspondant à l'exemple de l'erreur de vibrations. Les nœuds du BN ont chacun un nombre d'états précis à définir. Pour chaque nœud du BN, nous fixons une table de probabilité conditionnelle (CPT) qui contient les valeurs de probabilités initiales.

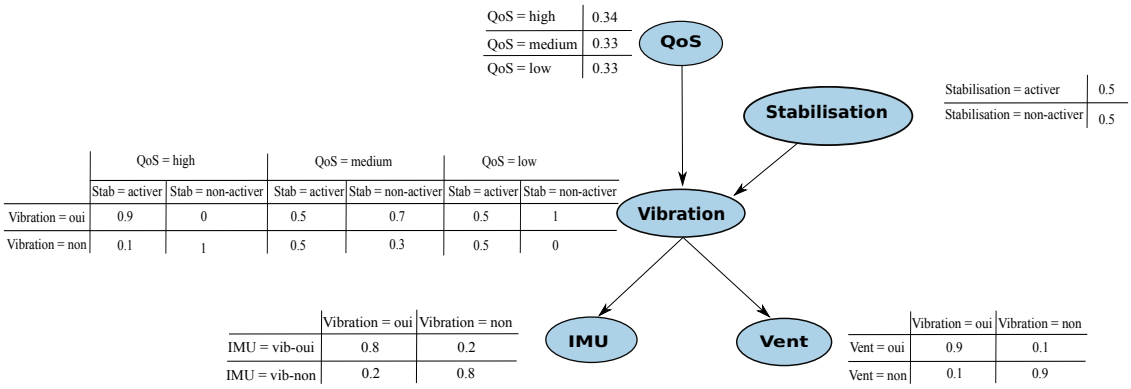


Figure 3.2: *BN associé au type d'erreur vibration.*

L'observation faite sur le contexte de la mission est capturée par les nœuds feuilles du réseau Bayésien sous forme d'évidences selon l'état observé du nœud. Par exemple, si nous sommes en présence de vent lors du déroulement de la mission du véhicule autonome, l'évidence fixée sur respectivement le nœud capteur IMU et le nœud contexte d'apparition (vent) correspondra à l'état 'vibration détectée' pour l'IMU et à l'état 'vent présent' pour le nœud vent de la Figure 3.2. Le premier usage du BN est l'évaluation de la qualité de service de l'application (nœud QoS) sachant les évidences mises sur les nœuds feuilles du BN. La Figure 5.4 montre l'impact du contexte observé sur la probabilité de QoS d'une application. Nous pouvons constater que lorsque nous sommes dans un contexte sans vent et donc sans vibrations, la QoS de l'application est élevée (high) avec une probabilité de 0.45. Dans le cas contraire, où nous observons une présence de vibrations, la QoS de l'application diminue avec une probabilité 0.26 pour l'état high et 0.41 pour l'état low.

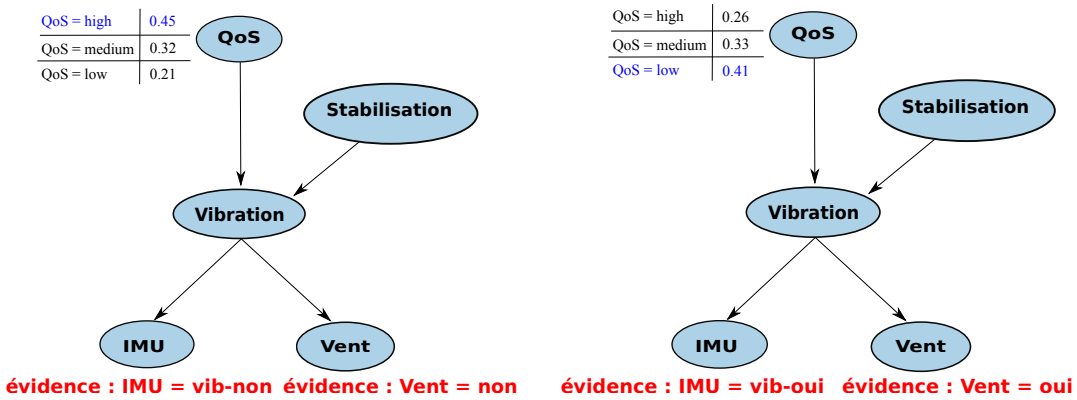


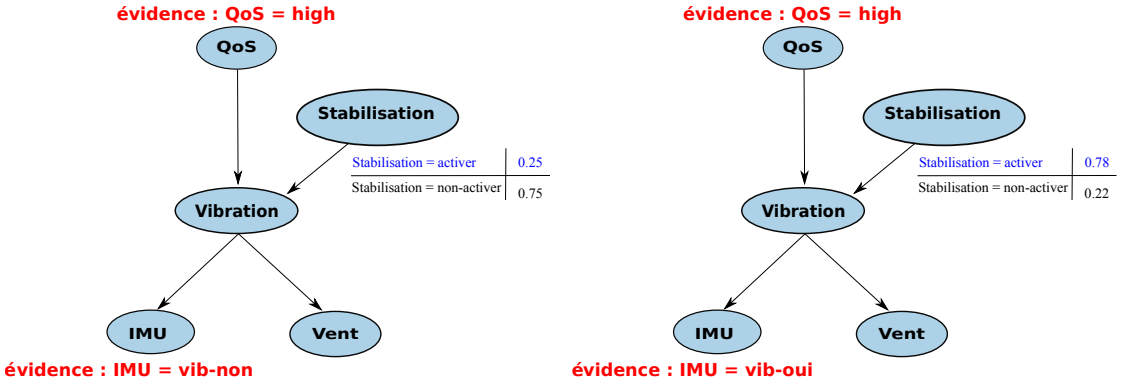
Figure 3.3: *Évolution de la QoS d'une application selon le contexte observé.*

3.2.2 BN pour adapter l'application au contexte environnemental

En plus de l'évaluation de la QoS de l'application, notre modèle Bayésien offre la possibilité de discriminer les versions algorithmiques en évaluant pour chacune la probabilité qui permet d'atteindre l'objectif de QoS fixé. Dans ce sens, le BN joue un rôle d'aide à la décision.

Une fois le type d'erreur détecté selon le contexte environnemental observé, la QoS est fixée à la valeur souhaitée dans le but de garder le bon fonctionnement de l'application. Le nœud QoS du BN est considéré, cette fois-ci, comme un nœud observable sur lequel nous fixons une évidence. Par conséquent, en fonction du contexte observé à l'aide des nœuds moniteurs et contexte d'apparition et du niveau de la QoS fixé, nous pouvons calculer la probabilité du nœud 'solution' pour avoir une estimation sur le choix de la version algorithmique de l'application à exécuter par le véhicule autonome pour continuer sa mission.

À l'aide de l'exemple vibrations Figure 3.2, nous allons montrer comment obtenir l'estimation de la solution à choisir en utilisant un réseau Bayésien. La Figure 3.4(a) représente un contexte sans erreur de vibrations avec une évidence portée par le nœud IMU. L'évidence du nœud IMU est donc fixée à la valeur 'vib-no' (vibrations non détecté). Nous fixons aussi l'évidence du nœud QoS à 'high' car l'objectif est d'estimer la probabilité d'activation du nœud solution afin de garantir le bon fonctionnement de l'application. Nous pouvons constater que dans le cas d'un contexte sans présence de vibrations, la probabilité d'activer la solution 'stabilisation' est de 0.25 qui est interprété par la désactivation de l'application de stabilisation. Par contre, dans le cas de présence de vibrations, l'évidence du nœud IMU est modifiée pour 'vib-yes' (présence de vibrations). La croyance en la présence de vibration peut être renforcée par le nœud vent (contexte d'apparition) dans le cas où nous avons cette information. La Figure 3.4(b) montre le cas de présence de vibrations. Notons



((a)) *Contexte sans erreur* ((b)) *Contexte avec erreur de vibration*

Figure 3.4: *Estimation du choix de solution pour une application à l'aide de BN.*

que la probabilité d'activation de l'application de stabilisation est passée de 0.25 à 0.78. Cette probabilité est interprétée par le fait que l'application de stabilisation doit être activée pour corriger l'erreur de vibrations et ainsi prolonger la mission du véhicule autonome.

3.2.3 BN pour adapter l'application aux ressources du système

Souvent, lors de l'évaluation de l'état de santé des composants (capteurs, etc.) d'un système autonome, seul le contexte environnemental est considéré. Le modèle Bayésien que nous proposons prend en compte non seulement le contexte environnemental mais intègre aussi le monitoring de l'état interne du système. Dans notre cas, nous intégrons le monitoring du système en termes d'occupation des ressources de calculs car chaque application consomme un certain nombre de ressource selon la variation algorithmique exécutée et les performances demandées. Nous verrons plus tard que ces ressources peuvent être matérielles (ex : 2 tuiles du FPGA) ou logicielles (ex : 40% de CPU).

Continuons avec l'exemple de vibrations. Nous allons ajouter un nœud 'ressources' au BN précèdent. La Figure 3.5 montre le nouveau BN obtenu. Chaque application a besoin d'un nombre de ressources disponibles pour pouvoir s'exécuter et l'occupation en ressources diffère d'une application à une autre. Le nœud 'ressources' est un nœud moniteur, à chaque fois une évidence est fixée selon l'observation faite sur l'occupation des ressources du système. Supposons que l'application de stabilisation consomme 35% des ressources, nous allons montrer l'adaptation du choix de l'application à exécuter aux ressources.

La Figure 3.6(a) et la Figure 3.6(b) montrent un exemple de l'estimation du choix de la solution algorithmique en fonction du contexte environnemental et de la

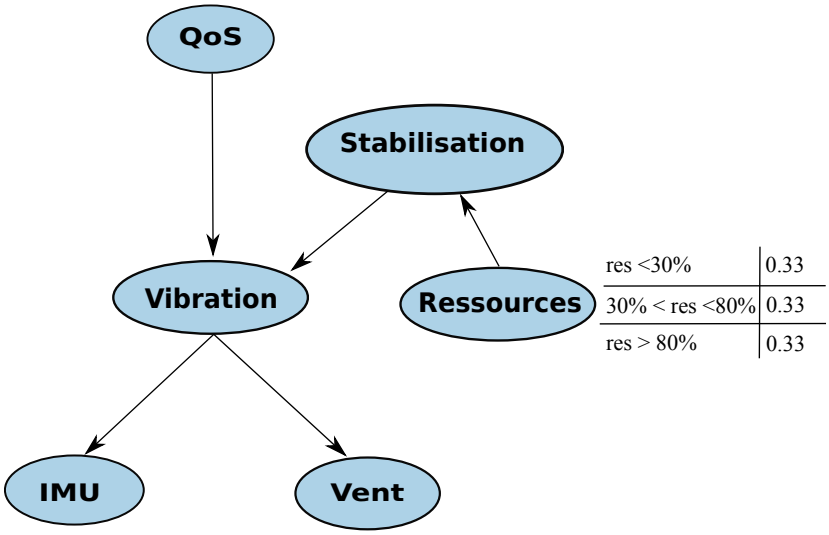
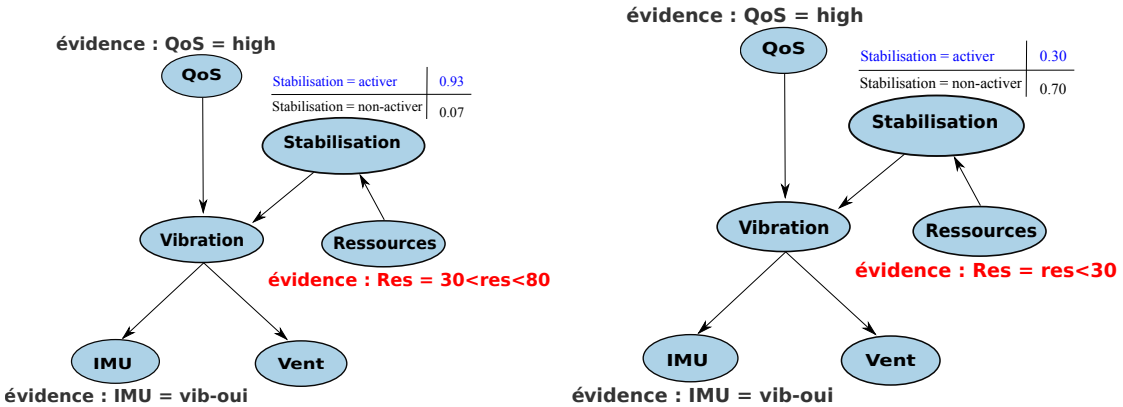


Figure 3.5: *BN vibrations adapté aux ressources*

disponibilité des ressources d'exécution. En plus de l'évidence indiquant le contexte observé (vibrations) et celle fixant la QoS à la valeur souhaitée, nous avons l'évidence portée par le nœud 'ressources' qui représente le taux de ressources disponibles monitoré par le système. L'application de stabilisation consomme 35% des ressources disponibles. Dans le cas où le taux de ressources disponibles est inférieur à 30%, l'application de stabilisation ne pourra pas être exécutée. La Figure 3.6(b) montre cela avec une probabilité de 0.30 pour l'état 'activer' et 0.70 pour l'état 'désactiver'. À l'inverse, avec le même contexte environnemental si nous observons qu'il y a entre 30% et 80% de ressources disponibles, la probabilité d'activation de l'application de stabilisation augmente à 0.93 comme le montre la Figure 3.6(a).



((a)) *ressources disponibles* ((b)) *ressources disponibles insuffisantes*

Figure 3.6: *Choix de la solution (application) adaptée aux ressources à l'aide de BN.*

En plus des ressources, l'exécution d'une application requière un certain niveau de performances pour avoir un résultat précis. La prise en compte des performances par le BN peut se faire de deux façons possibles :

— Représentation des performances sous forme de nœud Bayésien *Perf_Ci*. Avec cette représentation, nous avons besoin d'identifier le contexte d'apparition dans lequel la performance d'une application peut changer (diminuer ou augmenter la performance). Dans ce cas, le nœud *Perf_Ci* est le fils du nœud solution *C_Ei*, le/les nœud(s) contexte d'apparition pour les performances sont à leurs tours fils du nœud *Perf_Ci*. La Figure 3.7 montre un exemple où les performances sont modélisées avec un nœud Bayésien.

— La seconde représentation possible des performances consiste à mettre à jour la table de probabilités conditionnelles initiales du nœud 'solution' selon la performance souhaitée pour l'application à exécuter. La Figure 3.8 montre la table de probabilités conditionnelles initiales mise à jour à travers un exemple. Chaque application (ex : stabilisation) peut avoir au maximum trois variantes algorithmiques (HW, HW/SW et SW) selon le choix de l'implémentation. Si l'occupation ressources (ex : CPU) est inférieure à 30% alors la version SW de l'application peut s'exécuter sans problème. Si l'occupation ressources est entre 30% et 80%, la version HW/SW est privilégiée. Au delà de 80% des ressources (CPU) occupés, la version HW est choisie.

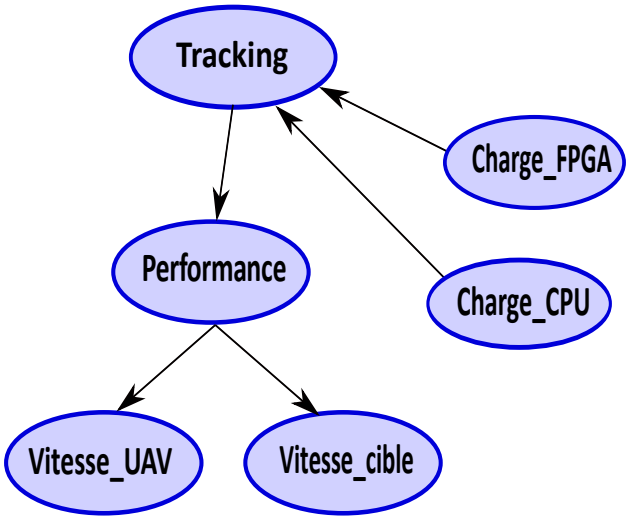


Figure 3.7: *Sous-Réseau Bayésien représentant les performances associées à une application.*

Pour la suite du chapitre, nous choisissons d'intégrer la notion de performances au niveau des tables de probabilités initiales des nœuds 'solution' *C_Ei*. La raison pour laquelle nous optons pour ce choix est qu'avec la représentation avec des nœuds Bayésiens, la difficulté réside dans l'initialisation des tables de probabilités

Cas performance = 100MHz

	res < 30%	30% < res < 80%	res > 80%
Stabilisation = activer	0.9	0.7	0.1
Stabilisation = non-activer	0.1	0.3	0.9

Cas performance = 250MHz

	res < 30%	30% < res < 80%	res > 80%
Stabilisation = activer	0.8	0.5	0.1
Stabilisation = non-activer	0.2	0.5	0.9

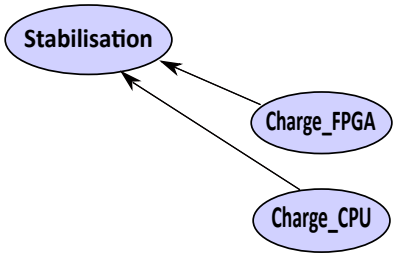


Figure 3.8: *Intégration des performances dans les tables de probabilités initiales.*

conditionnelles des nœuds *Perf_Ci* et contexte d'apparitions des performances. La mise à jour des tables de probabilités des nœuds 'solutions' selon la performance souhaitée pour une application est plus appropriée car nous pouvons faire appel à l'apprentissage des paramètres de BNs pour affiner les valeurs de probabilités.

3.2.4 Spécification des BNs à partir d'une table FMEA

L'analyse par **FMEA** (Analyse des Modes de Défaillances et leurs Effets) est une méthode souvent utilisée dans l'industrie pour la détection d'erreurs, de pannes et l'évaluation du risque des tâches. En plus de ces informations (types d'erreurs, moniteur et contextes d'apparitions), nous ajoutons l'information '**solution**' à l'analyse FMEA qui consiste à lister les solutions algorithmiques possibles pour corriger ou atténuer les types d'erreurs identifiés. La table FMEA associée à l'exemple précédent (présence de vibrations) est résumée par la Table 3.1.

Types d'erreurs	**Moniteurs possibles**	**Contexte d'apparition**	**Solution**
Vibration	Capteur IMU	Vent	Activation de la stabilisation

Tableau 3.1: *Table FMEA associée à l'erreur de vibration.*

Génération du modèle Bayésien des applications à partir de FMEA

Une fois la table FMEA établie, le réseau Bayésien associé à l'application peut être construit. La génération du réseau Bayésien a pour but de produire un diagnostic probabiliste et de décider de la solution la plus efficace à appliquer dans le cas où un type d'erreur est observé. La génération du réseau Bayésien doit respecter

les relations de causes à effets et le comportement des composants du systèmes (matériels et logiciels).

Modèle Bayésien générique à partir d'une table FMEA Chaque élément de la table FMEA-application est traduit sous forme d'un nœud du réseau Bayésien en respectant le principe suivant : pour chaque type d'erreur, nous construisons un sous réseau Bayésien, où l'erreur est représentée par un nœud inobservable *U_Ei*. Cette erreur peut-être monitorée par un capteur (matériel ou logiciel) représenté par un nœud moniteur *S_Ei* qui est le fils du nœud *U_Ei*. Le type d'erreur apparaît dans un contexte spécifique, donc nous lui associons un nœud *A_Ei* qui est aussi un nœud fils du nœud *U_Ei*. Pour chaque erreur observée, nous associons une solution probable, cette solution est représentée par un nœud *C_Ei* qui est un nœud parent du nœud *U_Ei*. Le nœud moniteur *S_Ei* pouvant être défaillant, nous ajoutons alors un nœud *H_S_E* pour surveiller l'état de santé du nœud moniteur. L'état de santé du nœud moniteur peut être renforcer par le contexte d'apparition lui aussi représenté par un nœud *A_H_E*. Le nœud état de santé du moniteur *H_S_E* a pour fils les deux nœuds *S_Ei* et *A_H_E*.

La Table 3.2 récapitule le contenu de l'analyse FMEA où :

— **U_Ei** : représente le type d'erreur qui peut être observé sur une application ou un capteur.

— **S_Ei** : représente le moniteur matériel (capteur) ou logiciel (calcul spécifique) du type d'erreur.

— **A_Ei** : représente le contexte d'apparition du type d'erreur.

— **C_Ei** : représente la solution algorithmique proposée pour corriger ou atténuer le type d'erreur observé et maintenir la QoS souhaitée pour une application.

Types d'erreurs	**Moniteurs possibles**	**Contexte d'apparition**	**Solution**
U_Ei	S_Ei	A_Ei	C_Ei

Tableau 3.2: *Table FMEA pour une application.*

En se basant sur le principe décrit ci-dessus, nous proposons un modèle de BN générique pour l'évaluation de la QoS des applications embarquées. Pour construire le BN global, nous ajoutons un nœud racine *H_App* qui représente la QoS de l'application. La QoS de l'application dépend des types d'erreurs listés. Afin de respecter la circulation d'information dans le réseau Bayésien, le nœud *H_App* est le parent des nœuds *U_Ei*. De cette façon, tous les sous réseaux Bayésiens des types d'erreurs, construits selon le principe noté précédemment, sont reliés au nœud *H_App* dans le but d'évaluer la QoS de l'application. La Figure 3.9 montre le BN générique obtenu.

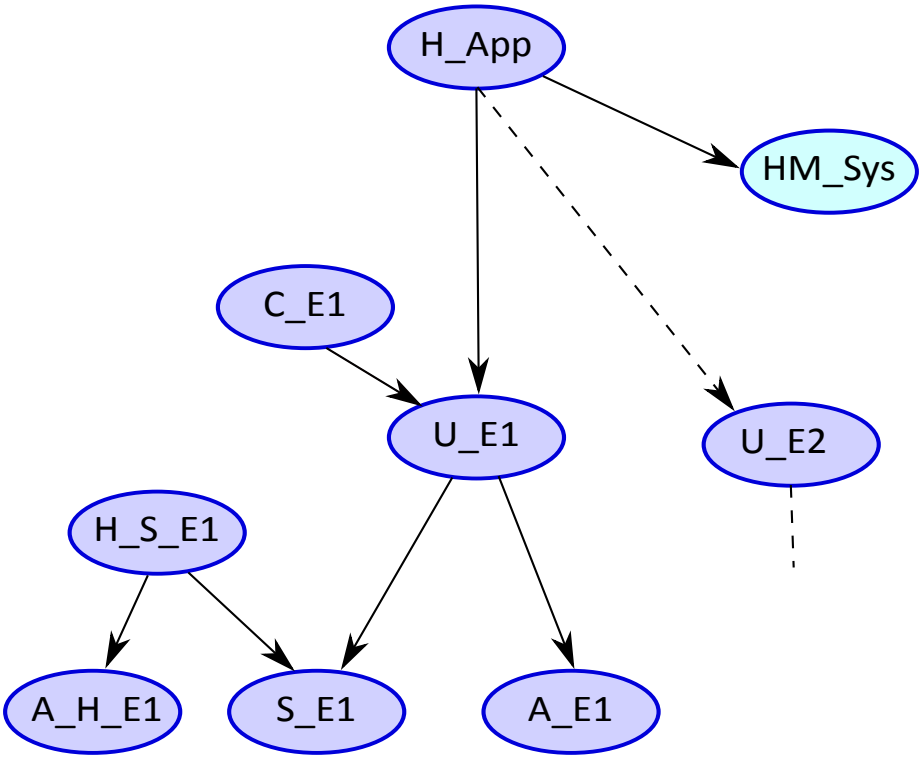


Figure 3.9: *BN générique pour évaluer la QoS des applications*

Signification des différents nœuds du BN générique :

— nœud ***H_App*** : représente le nœud QoS d'une application.

— nœud ***U_Ei*** : représente le nœud inobservable type d'erreur.

— nœud ***S_Ei*** : indique le type de moniteur (matériel ou logiciel) utilisé pour observer les types d'erreur.

— nœud ***H_S_Ei*** : indique l'état de santé du nœud capteur *S_Ei*.

— nœud ***A*** : représente le contexte d'apparition des types d'erreurs *A_Ei* ou de l'état de santé des capteurs *A_H_Ei*.

— nœud **C_Ei** : désigne la solution à adapter pour atténuer l'erreur observée.

— nœud ***HM_Sys*** : renseigne sur le bon fonctionnement ou non du système ou capteur utilisé par l'application dédiée (ex : caméra dans le cas d'une application de traitement d'image).

Ainsi, ce modèle Bayésien peut être construit automatiquement à partir d'une table FMEA dédiée aux applications.

Table FMEA incluant les ressources Pour qu'une application puisse s'exécuter correctement, il est nécessaire d'avoir suffisamment de ressources de calcul. Le choix de la 'solution' à exécuter dépend ainsi de l'état interne du système, notamment

la disponibilité des ressources, en plus du contexte externe (environnement). Nous proposons de compléter la table FMEA 3.2 en ajoutant une colonne 'ressources' qui indique le nombre minimum de ressources requis pour assurer l'exécution de chacune des solutions de la table FMEA. Nous rajoutons aussi une colonne 'performances' qui représente la performance minimal et maximal demandée pour une application donnée selon le contexte de la mission. La Table 3.3 résume la nouvelle analyse FMEA incluant les ressources. Le modèle Bayésien généré par la nouvelle table FMEA 3.3 respecte le même principe décrit plus haut. La modification à apporter par rapport au BN obtenu par la précédente table FMEA (sans les ressources), est que nous ajoutons des informations supplémentaires sous forme de nœuds Bayésiens. La Table 3.3 résume la nouvelle table FMEA incluant les ressources.

Types d'erreurs	**Moniteurs possibles**	**Contexte d'apparition**	**Solution**	**Ressources**	**Performances**
U_Ei	S_Ei	A_Ei	C_Ei	Ress_min, Ress_max	Perf_min, Perf_max

Tableau 3.3: *Table FMEA application avec ressources.*

Afin que le BN générique puisse être adapté aux ressources, nous effectuons les modifications suivantes : les ressources sont représentées par un nœud observable *Ress*. Ce nœud ressources est le parent de tous les nœuds 'solution' *C_Ei*, parce que l'activation ou non d'une solution dépend du nombre de ressources disponible, mais aussi pour garantir la circulation de l'information entre les différents nœuds du BN.

En ce qui concerne les performances souhaitées pour une application, nous choisissons d'intégrer les performances au niveau des nœuds 'solutions' en mettant à jour les tables de probabilités initiales comme expliquer dans la Section 3.2.3. Le réseau Bayésien générique incluant les ressources est défini dans la Figure 3.10.

Une fois la structure du réseau Bayésien définie, nous associons une table de probabilités conditionnelles pour chaque nœud du BN. Ces valeurs de probabilités peuvent être données soit par expertise, simulation ou bien par apprentissage. Nous présenterons l'apprentissage des probabilités d'un BN plus tard dans ce chapitre.

3.2.5 Exemple de BN-application : application de tracking

Dans les sections précédentes, nous avons vu comment générer un BN spécifique à partir de l'analyse FMEA d'une application donnée. Dans cette partie, nous allons illustrer cette approche par un exemple réel. Nous considérons une application de 'tracking' pour laquelle nous évaluerons d'une part sa QoS, et d'autre part choisir la version de tracking adéquate selon le type d'erreur observé dans un contexte particulier. L'application de 'tracking' étudiée dans cet exemple est le **TLD** (**T**racking,

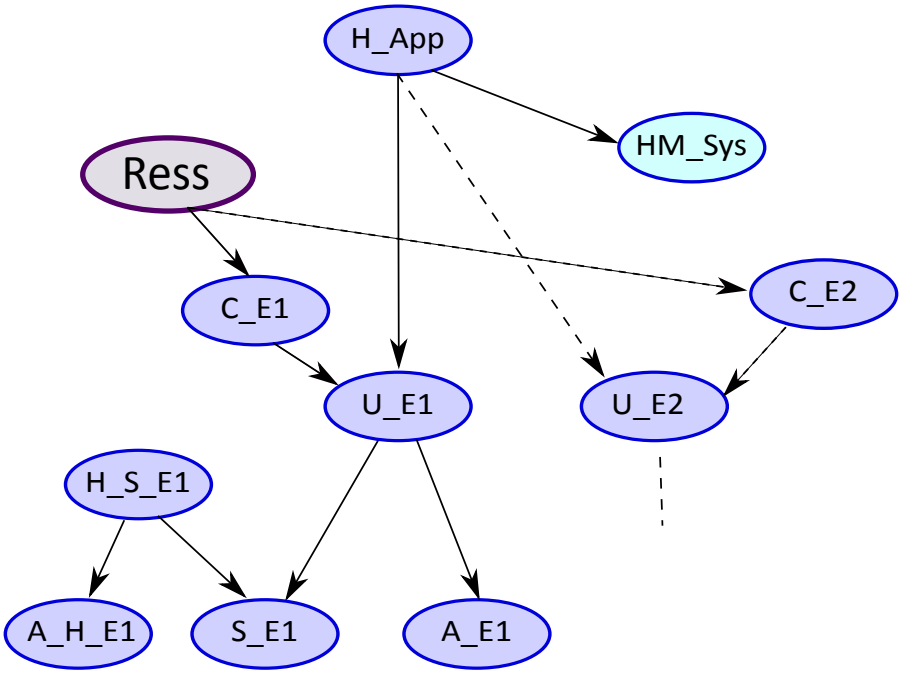


Figure 3.10: *BN générique considérant la consommation des ressources.*

Learning & **D**etection) [Kalal et al., 2012] appliquée durant une mission de drone pour suivre une cible potentielle.

L'application de 'tracking' fait partie de la famille des applications de traitement d'image où la qualité de l'image utilisée est importante pour obtenir le meilleur résultat possible. Parmi les événements qui peuvent altérer la qualité de l'image, nous trouvons la mauvaise luminosité, mauvaises conditions climatiques (brouillard, vent), etc. Par conséquent, ces événements externes impactent la QoS de l'application. En plus de ces deux sources d'erreurs, il peut y avoir d'autres sources de défaillances.

La Table 3.4 résume les types d'erreurs impactant la QoS de l'application de 'tracking', qui sont :

— **Vibration** : erreur de vibration pouvant être causée par la présence de vent par exemple.

— **Perte de points de tracking** : erreur de perte ou diminution des points caractéristiques détectés par le filtre HARRIS. La détection des points caractéristiques dépend du taux de luminosité.

— **Perte du vecteur de déplacement** : erreur de perte ou déviation du vecteur de déplacement entre deux images successives. Cette erreur peut apparaître lorsque le drone traque une cible potentielle et que la vitesse de la cible est supérieure à la vitesse du drone.

Le réseau Bayésien évaluant la QoS de l'application de 'tracking' peut être généré à partir de la table FMEA 3.4 en respectant le modèle et le principe proposé

Types d'erreurs	**Moniteurs possibles**	**Contexte d'apparition**	**Solution**
Vibration	Capteur IMU Capteur de vibration	Vent	Activer la stabilisation
Perte des points de tracking	Moniteur logiciel : nombre de points Harris détectés	Vitesse du drone Variation de la luminosité	Améliorer le contraste (pré-filtrage)
Perte du vecteur de déplacement	Moniteur logiciel : calcul du vecteur déplacement entre deux images	Vitesse de la cible Taille de la fenêtre de recherche (R.O.I)	Augmenter la taille de la R.O.I

Tableau 3.4: *Table FMEA pour l'application de 'tracking'.*

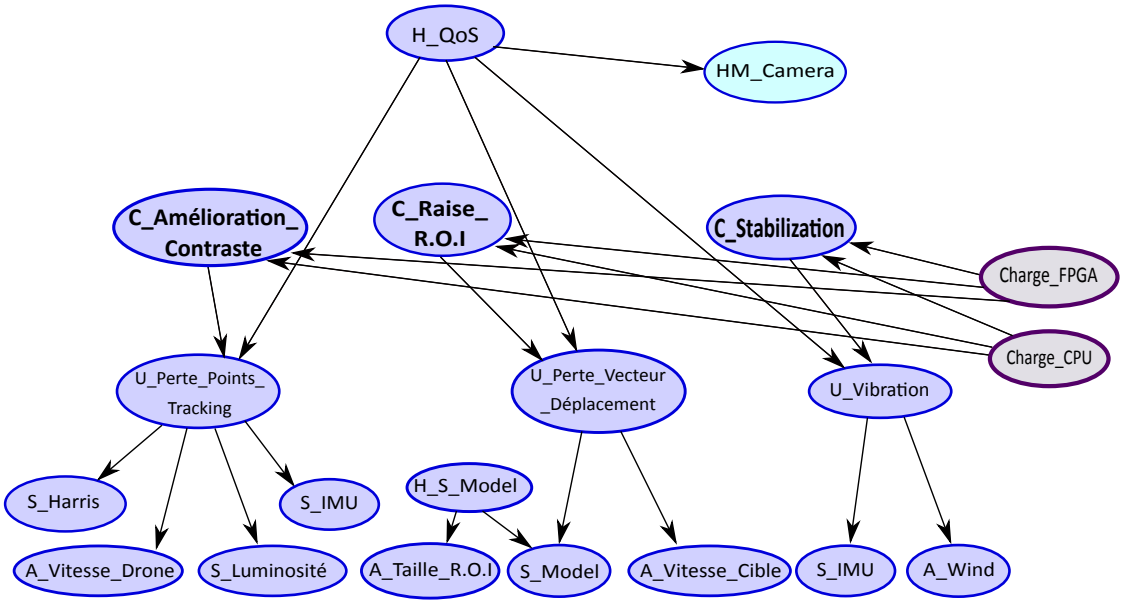


Figure 3.11: *Réseau Bayésien pour l'évaluation de la QoS de l'application 'tracking'.*

plus haut. Le BN-application évalue la QoS de l'application de 'tracking' et donne l'estimation en terme de probabilité de la solution (variations algorithmiques de l'application de tracking) à exécuter en fonction du type d'erreur identifié et de la disponibilité des ressources matérielles. Le réseau Bayésien de l'application de 'tracking' est donné par la Figure 3.11. Le nœud *H_QoS* représente la sortie du modèle et indique la probabilité de la QoS de l'application. La QoS de l'application varie selon les types d'erreurs représentés par *U_perte point tracking*, *U_perte vecteur déplacement* et *U_vibration*. Leurs monitoring respectifs sont donnés par des capteurs matériels (ex : IMU) ou logiciels (ex : métriques de l'application tel que la valeur du *Trust*) et sont représentés par les nœuds capteur et les nœuds contexte d'apparition. À chaque type d'erreur est associé une solution algorithmique (nœuds C_Vi) correspondant à une version (variante algorithmique) de l'application de tracking. Le nombre de ressources consommées varie d'une version du tracking à une autre. Le monitoring des ressources peut se faire directement à partir du support d'exécution, le nœud 'ressources' est utilisé comme nœud observable capteur. Dans notre cas, nous considérons deux nœuds 'ressources' car nous disposons de deux supports d'exécution ; un CPU et un FPGA. Enfin, toute application nécessite l'utilisation d'un capteur au minimum pour lui fournir des données en entrées, pour cela nous ajoutons un dernier nœud HM_capteur indiquant l'état de santé du capteur en question. L'état de santé HM_capteur peut être donné soit directement par monitoring du système ou bien calculé par un autre réseau Bayésien [Zermani et al., 2017b].

Dans la suite de ce chapitre, nous allons présenter comment l'évaluation de la QoS d'une application est calculée par le BN en utilisant un mécanisme d'inférence. Ensuite nous verrons comment déterminer les probabilités conditionnelles initiales

du BN par apprentissage.

3.3 Inférence AC pour BN-application

Dans cette section, nous allons présenter le calcul d'inférence après description du réseau Bayésien. Le calcul d'inférence se fait à travers une représentation interne du BN sous forme d'un arbre de Circuit Arithmétique. L'objectif de l'approche est de décrire l'AC de manière modulaire et hiérarchique pour pouvoir l'embarquer sur un système autonome et aussi faciliter les modifications dans le cas d'ajout et de suppression de nœuds du BN.

3.3.1 Description du réseau Bayésien

Il existe plusieurs outils permettant de décrire un réseau Bayésien, outils graphiques et textuels. Chaque description (graphique ou textuelle) comprend :

— nœuds : indiquent les variables aléatoires du réseau.

— Arcs : représentent les dépendances entre les nœuds du réseau.

— Tables de probabilités : regroupant les valeurs de probabilités conditionnelles à priori entre les nœuds.

Un BN peut être décrit et manipulé en utilisant soit une spécification textuelle en utilisant l'outil MATLAB, soit en utilisant GeNIe ou Hugin pour une spécification graphique du BN.

1. Description textuelle MATLAB (.m) : la description sous forme de fichier (.m) se fait en suivant les étapes suivantes :
   — définir le nombre de nœuds N du BN.
   — définir les dépendances entre les nœuds : en créant une matrice de taille N*N, initialement nulle. Si il y a une dépendance (cause à effet) entre deux nœuds, l'arc est représenté par la valeur 1 dans la matrice (dans la case correspondant aux deux nœuds).
   — définir le nombre d'états de chaque nœud en respectant l'ordre des nœuds : se fait via un vecteur de taille N.
   — renseigner les tables de probabilités associées à chaque nœud du BN. Cela se fait après la création de la structure du BN *bnet*.
2. Description graphique d'un BN (.net) : dans le (.net) les éléments important du BN sont visible graphiquement.
   — définition des nœuds du BN et leurs identifiant (nom du nœud).
   — définition des états de chaque nœud.
   — remplir la table de probabilité de chaque nœud du BN.

La description graphique en .net peut être traduite en description textuelle sous forme XML en utilisant des balises spécifiques pour la définition du nœud, ses états et sa table de probabilité.

3.3.2 Inférence AC

Dans cette partie nous allons utiliser une partie du BN-application pour illustrer l'inférence AC en utilisant une représentation du BN sous forme d'arbre contenant que des opérations d'additions (+) et de multiplications (*).

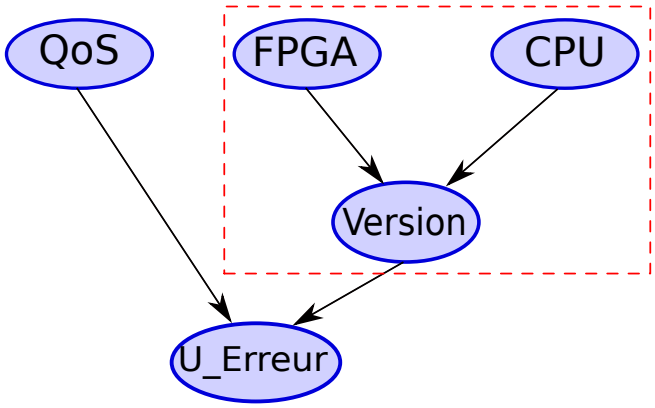


Figure 3.12: *Exemple de Réseau Bayésien*

La Figure 3.12 montre un pattern du BN de l'application de 'tracking' présenté dans la section précédente. Le BN se compose de cinq nœuds (QoS, FPGA, CPU, Version et U_Erreur). Le nœud QoS possède 3 états, qui sont 'high', 'medium' et 'low'. Le nœud CPU compte aussi 3 états, décomposés selon de taux d'occupation (en pourcentage) du CPU. Les nœuds Version et U_Erreur possèdent 2 états chacun, 'activer' et 'désactiver' pour le nœud Version, 'détectée' et 'non-détectée' pour le nœud U_erreur. Le nœud FPGA possède 3 états représentant le nombre de tuiles du FPGA. La Table 3.5 résume les identifiants d'évidences et les paramètres (ID probabilités) pour chaque nœud du BN. Pour montrer les étapes de création de l'AC, nous utilisons le sous-réseau de l'exemple constitué des trois nœuds (FPGA, CPU et Version).

1. **Définition de l'ordre des nœuds dans le BN** : commencer par ordonner les nœuds parents en premiers (nœuds n'ayant pas eux-même de parents), ensuite ajouter les nœuds fils. Ce qui donne l'ordre suivant pour notre exemple : QoS, FPGA, CPU, Version, U_Erreur.
2. **Polynôme associé au BN de l'exemple** : pour des raisons de lisibilité, nous allons définir le polynôme du sous-réseau Bayésien (FPGA, CPU et version) comme suit (équation 3.1) :

nœuds	ID d'évidences	ID paramètres
QoS	(λq0, λq1, λq2)	(θq0, θq1, θq2)
FPGA	(λf0, λf1, λf2)	(θf0, θf1, θf2)
CPU	(λc0, λc1, λc2)	(θc0, θc1, θc2)
Version	(λv0, λv1)	(θv0, θv1, ...)
U_Erreur	(λe0, λe1)	(θe0, θe1, ...)

Tableau 3.5: *Identifiant d'évidences et de paramètres du BN pour l'inférence AC.*

$$
\begin{aligned}
F \;=\;& \lambda_{f1}\theta_{f1}(\lambda_{c1}\theta_{c1}(\lambda_{v1}\theta_{v1}|f_1c_1 + \lambda_{v2}\theta_{v2}|f_1c_1) + \lambda_{c2}\theta_{c2}(\lambda_{v1}\theta_{v1}|f_1c_2 + \lambda_{v2}\theta_{v2}|f_1c_2)\\
+\;& \lambda_{c3}\theta_{c3}(\lambda_{v1}\theta_{v1}|f_1c_3 + \lambda_{v2}\theta_{v2}|f_1c_3)) + \lambda_{f2}\theta_{f2}(\lambda_{c1}\theta_{c1}(\lambda_{v1}\theta_{v1}|f_2c_1 + \lambda_{v2}\theta_{v2}|f_2c_1)\\
+\;& \lambda_{c2}\theta_{c2}(\lambda_{v1}\theta_{v1}|f_2c_2 + \lambda_{v2}\theta_{v2}|f_2c_2) + \lambda_{c3}\theta_{c3}(\lambda_{v1}\theta_{v1}|f_2c_3 + \lambda_{v2}\theta_{v2}|f_2c_3)) + \lambda_{f3}\theta_{f3}\\
(\;& \lambda_{c1}\theta_{c1}(\lambda_{v1}\theta_{v1}|f_3c_1 + \lambda_{v2}\theta_{v2}|f_3c_1) + \lambda_{c2}\theta_{c2}(\lambda_{v1}\theta_{v1}|f_3c_2 + \lambda_{v2}\theta_{v2}|f_3c_2) + \lambda_{c3}\theta_{c3}\\
(\;& \lambda_{v1}\theta_{v1}|f_3c_3 + \lambda_{v2}\theta_{v2}|f_3c_3)). \qquad (3.1)
\end{aligned}
$$

Après la définition du polynôme F factorisé du BN, l'arbre AC peut être construit en respectant le polynôme et en suivant les étapes de la compilation AC donnée par l'algorithme suivant :

Algorithme 1 : Compilation en AC

Ordonner les nœuds par ordre topologique et en ASAP
Créer un (+) pour le premier nœud, ayant N_0 fils (*) {N_0 est le nombre d'états du nœud}
A chaque (*) faire correspondre son indicateur d'évidence λ et son paramètre θ
pour tout les autres nœuds, par ordre **faire**
  si le nœud a au moins un parent **alors**
    Ajouter un (+) à tous les (*) du dernier père ayant N_i fils (*)
    A chaque (*) faire correspondre son indicateur d'évidence λ et son paramètre θ
  sinon
    Ajouter un (+) à tout les (*) du dernier nœud de la liste avant le nœud en question, ayant N_i fils (*) {N_i est le nombre d'états du nœud}
    A chaque (*) faire correspondre son indicateur d'évidence λ et son paramètre θ
  fin si
fin pour

Une fois l'arbre AC construit, il est décomposé en blocs élémentaires dans le but d'accélérer et de paralléliser les calculs d'inférence. De cette manière une version embarquable de l'inférence Bayésienne est envisagée (pour plus de détails voir les travaux de [Zermani, 2017]).

3.4 Apprentissage des paramètres de réseau Bayésien

3.4.1 Contexte et objectif de l'apprentissage

Dans le cas d'une mission de véhicule autonome, l'évaluation en ligne de l'état de santé du système autonome est nécessaire pour détecter les anomalies liées aux capteurs ou au système lui-même et par la suite décider de la tâche à exécuter en conséquence. L'évaluation de l'état de santé des composants d'un système est donnée par inférence (voir Sec 3.3) en se basant sur les tables de probabilités conditionnelles initiales des nœuds capteurs et contexte d'apparitions.

Ces tables de probabilités conditionnelles initiales sont en pratique soit données par expertise, simulation ou bien inconnues. C'est pourquoi, il est important d'introduire les méthodes d'apprentissage des paramètres des BNs. L'apprentissage permet d'une part le calcul des paramètres du BN (probabilités) dans le cas où ils sont inconnus et d'autre part améliorer ces paramètres dans le cas où l'expertise ne renseignerait pas toutes les données nécessaires pour l'évaluation de l'état de santé du système et de ses composants.

Dans ce qui suit, nous allons présenter l'algorithme le plus utilisé pour l'apprentissage des paramètres d'un BN ainsi que la version embarquée de l'algorithme que nous proposons pour pouvoir faire de l'apprentissage en ligne.

3.4.2 Algorithme d'apprentissage des paramètres d'un réseau Bayésien

Dans cette section, nous nous concentrons sur les réseaux Bayésiens statiques. L'algorithme le plus utilisé pour l'apprentissage des paramètres d'un BN est l'algorithme **EM** *(**Maximisation** et **Expectation**)*. Dans ce qui suit, nous allons rappeler le principe de l'algorithme **EM** puis nous abordons la version embarquée proposée pour l'algorithme EM.

EM est un algorithme classique utilisé pour l'estimation des paramètres θ (probabilités) d'un réseau Bayésien sachant que les données ont été observées soit de manière complète ou incomplète (manquantes). EM est un algorithme itératif, son fonctionnement est basé sur deux étapes principales : **l'étape E** et **l'étape M** [Naïm et al., 2007]. L'étape **E** estime les paramètres θ en s'appuyant sur les paramètres initiaux et la base de connaissance. L'étape **M** maximise les paramètres estimés à l'étape E.

3.4.3 L'algorithme EM revisité pour l'embarqué

Généralement, l'algorithme EM utilise l'inférence par arbre de jonction pour l'estimation des paramètres d'un BN. Notre objectif est de pouvoir faire de l'apprentissage des paramètres durant la mission du véhicule autonome (quand c'est nécessaire). Par conséquent, il est indispensable d'avoir un mécanisme d'inférence embarquable permettant de faire l'apprentissage en ligne.

Notre approche consiste à proposer une version de l'algorithme EM utilisant l'inférence par l'AC dont une version embarquable est possible contrairement à l'arbre de jonction. Dans ce qui suit, nous présentons le principe de la version proposée de l'algorithme EM-AC à travers un exemple de BN constitué de cinq nœuds. Chaque nœud contient deux états. Le BN utilisé comme exemple explicatif est représenté par la Figure 3.13.

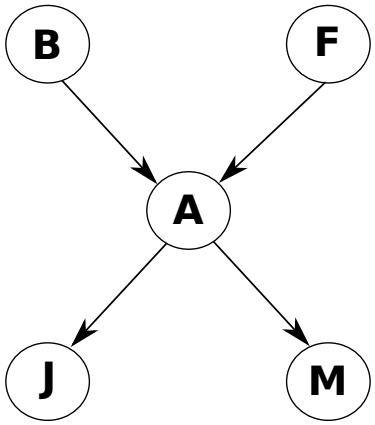


Figure 3.13: *Exemple de réseau bayésien*

nœuds	Probabilités initiales
F	P(f) = 0.002
B	P(b) = 0.001
J	P(j|a) = 0.95, P(j| $\overline{a}$) = 0.1
M	P(m|a) = 0.99, P(a| $\overline{a}$) = 0.3
A	P(a|b,f) = 0.99, P(a|b,$\overline{f}$) = 0.7 P(a| $\overline{b}$,f) = 0.94, P(a| $\overline{b}$,$\overline{f}$) = 0.05

Tableau 3.6: *Probabilités initiales des nœuds du BN.*

1. **Principe de EM-AC** :
   — Définir les différentes familles (ensemble de nœuds parents et nœuds fils) du réseau Bayésien.

	F	**B**	**A**	**J**	**M**
Observation 1	-	-	a	j	m
Observation 2	$\overline{f}$	$\overline{b}$	$\overline{a}$	-	-
Observation 3	f	$\overline{b}$	a	j	m

Tableau 3.7: *Base de connaissance avec des données manquantes.*

— Appliquer l'**étape E** de l'algorithme avec l'**AC** pour l'inférence.

— Appliquer l'**étape M** de l'algorithme avec l'**AC** pour l'inférence.

2. **Application de l'EM-AC sur l'exemple avec 5 nœuds** : nous allons détailler les différentes étapes de l'apprentissage des paramètres du BN en utilisant l'algorithme EM-AC.

— **Définition des différentes familles du BN** : une famille d'un BN est constituée des nœuds ascendants directs et de leur nœuds parents. Cinq familles sont déduites du BN de la Figure 3.13, à savoir : **B**, **F**, **ABF**, **JA** et **MA**.
Une fois les familles définies, nous traduisons la base de connaissances, Table 3.7, en évidences (e) pour qu'elles puissent être utilisées lors du calcul de l'inférence. Pour chaque couple (colonne, ligne) est associé une évidence selon l'état du nœud observé.

Observation1	**Observation2**	**Observation3**
e(1,1) = [1 1]	e(1,2) = [0 1]	e(1,3) = [1 0]
e(2,1) = [1 1]	e(2,2) = [0 1]	e(2,3) = [0 1]
e(3,1) = [1 0]	e(3,2) = [0 1]	e(3,3) = [1 0]
e(4,1) = [1 0]	e(4,2) = [1 1]	e(4,3) = [1 0]
e(5,1) = [1 0]	e(5,2) = [1 1]	e(5,3) = [1 0]

Comme chaque nœud du BN possède deux états, la valeur de chaque évidence est composée de '0' et/ou de '1'. Si l'état observé du nœud est 'état 1' alors la valeur de l'évidence est [1 0] ; si c'est 'l'état 2' qui a été observé alors la valeur de l'évidence vaut [0 1] ; par contre si aucun état du nœud n'a été observé, la valeur de l'évidence est [1 1].

— **Application de l'étape E** : avant de pouvoir estimer les paramètres du BN en appliquant l'étape E de l'algorithme EM, il faut d'abord construire l'AC du BN traité afin de permettre le calcul d'inférence.
Une fois l'AC construit, la première étape est de calculer la probabilité P(e). 'e' prend à chaque fois l'échantillon de données observé (un échantillon correspond à une ligne de la base de connaissance).

Pe1 = Inference_AC_Exemple(T_B, T_F, T_A, T_J, T_M, e11,e21,e31,e41,e51).
Pe2 = Inference_AC_Exemple(T_B, T_F, T_A, T_J, T_M, e12,e22,e32,e42,e52).
Pe3 = Inference_AC_Exemple(T_B, T_F, T_A, T_J, T_M, e13,e23,e33,e43,e53).

- Cas de famille composée d'un seul nœud (sans parents) : calcul de P(B = b).

Observation de l'état 'b' pour le nœud 'B' ⇒ b = [1 0] (valeur de l'évidence sur le nœud B).
Calcul de la probabilité en considérant la nouvelle évidence (b = [1 0]) qu'on souhaite calculer avec les données de la base de connaissance.

Nous remplaçons la valeur de l'évidence du nœud 'B' dans chaque échantillon (ligne) de la base de connaissance par b = [1 0] et nous calculons la nouvelle valeur de P(e), comme suit :

Pb1 = Inference_AC_Exemple(T_B, T_F, T_A, T_J, T_M, b,e21,e31,e41,e51).
Pb2 = Inference_AC_Exemple(T_B, T_F, T_A, T_J, T_M, b,e22,e32,e42,e52).
Pb3 = Inference_AC_Exemple(T_B, T_F, T_A, T_J, T_M, b,e23,e33,e43,e53).

Ensuite, la probabilité P(B = b) est calculée pour chaque observation répertoriée dans la base de connaissance.

P1 = Pb1/Pe1 ;
P2 = Pb2/Pe2 ;
P3 = Pb3/Pe3.

L'estimation finale de la valeur du nœud 'B' dans le cas (B = b) est : **P(B = b) = P1 + P2 + P3**.

- Cas de famille composée de nœuds parents et fils : calcul de P(a|b,f).

Les valeurs des évidences à modifier portent sur les nœuds B, F et A, tel que : b = [1 0], f = [1 0] et a = [1 0].
Comme précédemment, la nouvelle valeur de Pne = P(e) (Pne : Probabilité sachant la nouvelle évidence) est calculée en remplaçant la valeur des évidences des nœuds B, F et A par [1 0], comme suit :

Pa1 = Inference_AC_Exemple(T_B, T_F, T_A, T_J, T_M, b,f,a,e41,e51).
Pa2 = Inference_AC_Exemple(T_B, T_F, T_A, T_J, T_M, b,f,a,e42,e52).
Pa3 = Inference_AC_Exemple(T_B, T_F, T_A, T_J, T_M, b,f,a,e43,e53).

Une fois Pa1, Pa2 et Pa3 calculées, nous allons évaluer la probabilité P(a|b,f) pour chaque observation (échantillon) de la base de connaissance.

P1 = Pa1/Pe1 ;
P2 = Pa2/Pe2 ;
P3 = Pa3/Pe3.

L'estimation finale de la valeur du nœud 'A' dans le cas (B = b) et (F = f) est : **P(a|b,f) = P1 + P2 + P3**.

— **Application de l'étape M** : la maximisation des paramètres estimés dans l'étape E se fait de deux manières différentes selon le type de la famille à laquelle appartient le nœud pour lequel nous souhaitons estimer la probabilité.

- Cas de famille composée d'un seul nœud (sans parents) :
Dans le cas où le nœud ne possède pas de parents, la maximisation de la probabilité se fait en divisant la probabilité obtenue à l'estimation (étape E) par la taille de la base de connaissance. Ce qui donne pour l'exemple :

$$\mathbf{P(B = b) = (P1 + P2 + P3) / 3}.$$

- Cas de familles composée de nœuds parents et fils :
Lorsque nous calculons la probabilité d'un nœud sachant l'état de ses nœuds parents, il est nécessaire de calculer la probabilité du nœud parent sachant les données observées **P(Parent | Évidence)**. Ce qui est le cas de la famille **ABF** pour le calcul de P(a | b,f).
Calcul de P(Parent | Évidence) par inférence : les parents sont B=b et F=f. Les évidences sont [1 0] sur les nœuds B et F.

P_bc1 = Inférence_AC_Exemple(T_B, T_F, T_A, T_J, T_M, b,f,e31,e41,e51).
P_bc2 = Inférence_AC_Exemple(T_B, T_F, T_A, T_J, T_M, b,f,e32,e42,e52).
P_bc3 = Inférence_AC_Exemple(T_B, T_F, T_A, T_J, T_M, b,f,e33,e43,e53).

Une fois la probabilité P(Parent | Évidence) est obtenue pour chaque observation de la base de connaissance, P(a | b,f) vaut :

$$\mathbf{P(a \mid b,f) = (P1 + P2 + P3) / (P\_bc1 + P\_bc2 + P\_bc3)}.$$

Nous avons bien vu que la version proposée de la méthode d'apprentissage des paramètres d'un BN utilise de l'inférence embarquée basée sur l'inférence AC.

Une fois la méthode d'apprentissage avec EM-AC est mise en place, nous pouvons l'appliquer sur nos BNs (applications, capteurs) construits à partir d'une table FMEA. La Figure 3.14 montre un exemple de BN pour l'évaluation de l'état de santé d'un capteur GPS. Nous avons deux types de nœuds dans les BNs appelés : nœuds calculés et nœuds observables. Les nœuds observables sont des nœuds capteurs principalement et contexte qui capturent les informations liées au contexte environnemental et au système interne. Les nœuds calculés sont des nœuds internes au BN et sont non observables, leurs probabilités sont calculées par inférence sachant les observations des nœuds feuilles (nœuds observables).

Par conséquent, les données les plus importantes pour le calcul de l'état de santé sont celles des nœuds feuilles observables. Souvent, ces données sont incomplètes par manques de précision d'expertise sur la mission du système autonome. L'avantage de notre méthode d'apprentissage avec EM-AC comparativement à l'algorithme EM classique est que nous pouvons faire un apprentissage ciblé. Cela veut dire que nous pouvons apprendre seulement les paramètres des nœuds feuilles du BN au lieu de faire un apprentissage de tous les paramètres de tout les nœuds du BN comme c'est le cas avec les méthodes classiques.

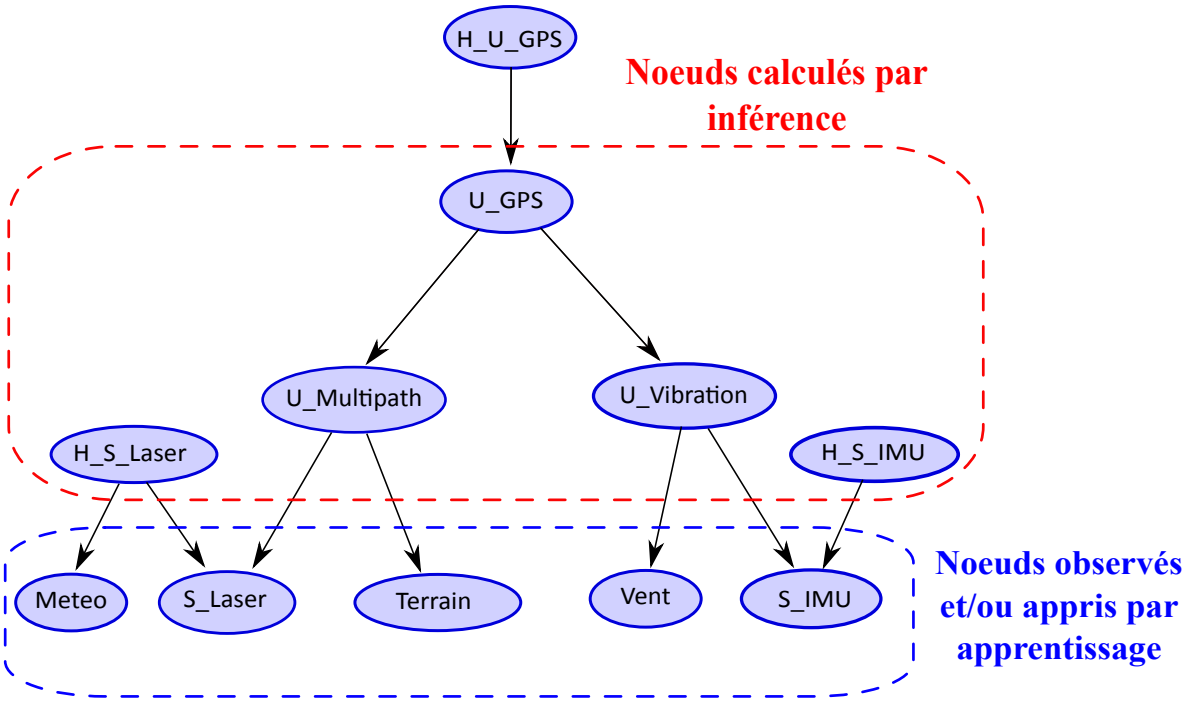


Figure 3.14: *BN capteur GPS*

3.5 Conclusion

Dans ce chapitre, nous avons décrit les contributions apportées à l'évaluation de l'état de santé d'un système autonome et de ses composants, plus particulièrement l'évaluation de la QoS des applications qui s'exécutent lors de scénarios de mission de système autonome et adaptatif. Nous avons proposé un modèle de réseau Bayésien statique pour le monitoring de la QoS des applications embarquées qui repose sur une analyse FMEA. Notre modèle Bayésien permet aussi d'estimer la variante algorithmique de l'application à exécuter en fonction du contexte externe (environnement) et de la disponibilité des ressources de traitement (CPU et FPGA). Nous

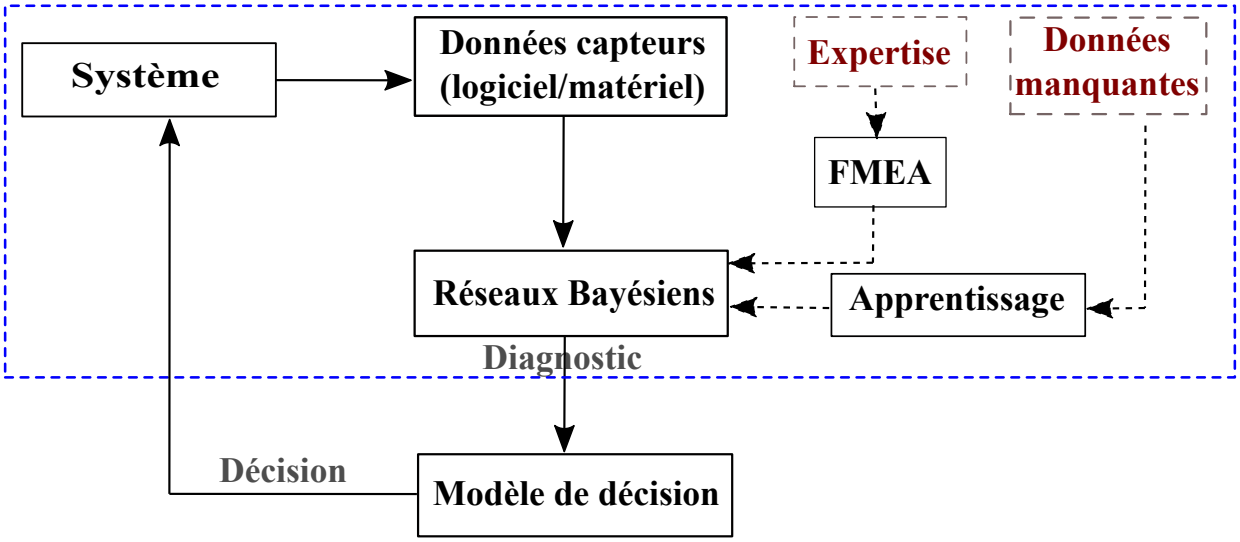


Figure 3.15: *Récapitulatif de la démarche d'évaluation du diagnostic pour un système autonome.*

avons complété cette étude par la proposition d'une version embarquable de l'apprentissage des paramètres d'un BN, avec l'algorithme EM, en utilisant de l'inférence AC. La Figure 3.15 récapitule la démarche suivie dans ce chapitre.

Cette contribution a abouti à des publications dans des conférences internationales [Hireche et al., 2018a] et [Hireche et al., 2017].

- Chapitre 4 -

Modèle de Décision : Markov Decision Process

Sommaire

4.1 Introduction

La montée en puissance de l'utilisation de véhicules autonomes nécessite une amélioration significative de la prise de décision embarquée. Cette prise de décision doit prendre en compte l'objectif de la mission du véhicule autonome et adapter le choix des actions à exécuter afin d'atteindre cet objectif.

La décision doit être prise en fonction des événements aléatoires liés au contexte dans lequel évolue la mission, l'état de santé de l'intégralité du système tout en considérant le niveau de risque autorisé pour la mission. L'autonomie d'une mission implique des calculs haute performance avec des ressources de traitement, communications et ressources de stockages limitées. Il est donc important que le processus de prise de décision inclut ces contraintes (ressources traitement, stockage, etc) afin d'éviter les choix conduisant à des erreurs causées par un manque de ressources de traitement. Le temps de réponse ou le débit d'une application peuvent en effet être dégradés par l'indisponibilité du système embarqué et provoqué des incidents majeurs comme la non détection d'un obstacle par exemple.

Dans ce chapitre, nous traitons la mise en œuvre de l'intelligence embarquée autonome pour faire face aux incertitudes pouvant survenir durant la mission. Nous proposons une nouvelle approche qui combine l'utilisation des BNs et des MDPs. Cette approche est divisée en deux parties, une partie diagnostic qui s'appuie sur les réseaux Bayésiens et une partie décision qui fait appel aux MDPs.

Dans la Section 4.2, nous présenterons l'architecture de notre modèle BFM pour la gestion de mission de systèmes autonomes. Ensuite, dans la Section 4.3 nous aborderons la démarche suivie afin de modéliser une mission avec un modèle BFM global monolithique. Dans la Section 4.4 nous exposons et expliquons la version concurrente du modèle BFM ainsi que la gestion de conflits de comportements et de ressources pouvant apparaître entre les différentes applications à exécuter. Enfin, nous concluons sur cette partie en récapitulant les travaux présentés.

4.2 Modèle de décision proposé : BFM

De précédents travaux ont déjà inclu les incertitudes liées à l'environnement dans la gestion de mission de drone. Par exemple, le modèle POMDP (MDP Partiellement Observable) [Vanegas et al., 2017] est un modèle utilisé pour la résolution de problèmes de gestion de mission en prenant des décisions efficaces durant le déroulement de la mission. L'inconvénient du modèle POMDP est qu'il n'est pas *scalable* et *modulable* dans le cas où nous avons une mission complexe à modéliser pour laquelle il faut générer des plans de mission afin de faire face aux différents événements qui peuvent conduire à l'échec de la mission du système autonome (ex : drone).

Dans le but de mener à bien la mission par le système autonome, le gestion-

naire de mission doit prendre en compte différentes contraintes (ressources, temps d'exécution, etc.) afin de générer une décision répondant à différents aléas (contexte environnemental et état interne du système). Pour cela, nous proposons un modèle de décision pour la gestion de mission nommé **BFM** pour ***réseau Bayésien construit à partir d'une table FMEA pour alimenter le modèle MDP***. Le modèle BFM a pour objectifs :

- Décider de l'application à exécuter à chaque étape de la mission.
- Décider de la version algorithmique de l'application à choisir si une application possède différentes versions algorithmiques en fonction du contexte dans lequel elle est exécutée.
- Intégrer l'état de santé des applications selon le contexte environnemental et l'état du système (occupation des ressources de calculs) (voir Chapitre 3. Section 3.2) et des composants du système (capteurs, etc.) dans le processus de prise de décision.
- Le choix de l'application est calculé en fonction de la stratégie suivie pour le déroulement de la mission. Deux stratégies sont définies : *stratégie de safety* et *stratégie mission first* qui sont détaillées dans la suite du chapitre.
- En plus des points cités ci-dessus, notre modèle de décision est modulable et *scalable* en termes de spécification de la mission et donne la possibilité de tirer profit du parallélisme offert par l'architecture hybride du support d'exécution (CPU/FPGA).

La Figure 4.1 montre l'architecture du modèle BFM. Le modèle BFM est composé de deux blocs, un premier bloc pour le diagnostic et un second bloc pour la prise de décision.

1. **Bloc Diagnostic** :
   Le bloc diagnostic a pour objectif d'évaluer l'état de santé des différents composants matériels (HW) et logiciels (SW) du système en considérant le contexte environnemental et les événements internes liés au système autonome. Nous considérons différentes catégories (modules) de Health Management (HM) pour le diagnostic afin d'alimenter le bloc de décision en probabilités. Les modules de HM utilisent les BNs pour l'évaluation des états de santé [Zermani et al., 2017b].
   - **HM capteur** : est composé de plusieurs HM capteurs, un HM par capteur ou groupe de capteurs permettant de calculer la probabilité de bon fonctionnement des capteurs embarqués en fonction de l'erreur détectée.
   - **HM application** : contient plusieurs HM pour évaluer la QoS des applications. Nous considérons un HM par application pouvant être exécuté sur le système embarqué (application de tracking, navigation, détection et évitement d'obstacle, etc).

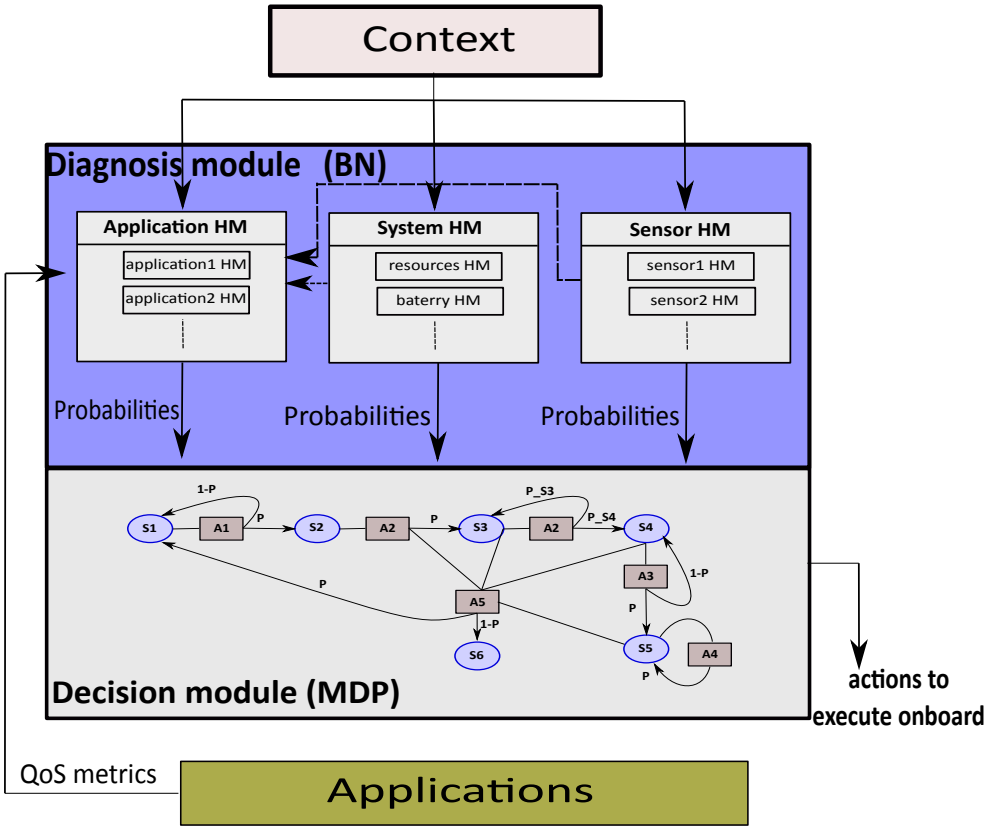


Figure 4.1: *Architecture du modèle BFM pour le planificateur de mission.*

— **HM système** : inclut les HM des composants restant comme les ressources (charge CPU, etc), la batterie, etc.

L'état de santé des capteurs et du système peuvent aussi impacter la QoS des applications. Pour cela, dans la Figure 4.1, le module HM application est d'une part relié au contexte, HM capteur et HM système et d'autre part à l'application elle-même qui peut renvoyer des métriques (indice de confiance) indiquant la présence possible d'erreurs au sein de l'application elle-même.

Tous les modules du bloc diagnostic sont élaborés à l'aide de BNs construits à partir de tables FMEA [Section 3.2.4]. Le diagnostic fournit en sortie les probabilités de bon fonctionnement ou pas du système, des capteurs et la probabilité de QoS des applications.

2. **Bloc Décision** :
Le bloc de décision repose sur le modèle MDP [Bethke et al., 2008] qui a pour but de produire la liste des actions à exécuter. Le modèle MDP prend en entrée

les probabilités calculées par les différents HM du bloc diagnostic du modèle BFM. Ces probabilités sont portées par les transitions du modèle MDP.

Les systèmes auto-adaptatifs de type SoC sont implémentés avec des architectures re-configurables et adaptent les configurations HW/SW en fonction de l'application à exécuter. Souvent, les approches de prise de décision existantent ne décident pas de la version algorithmique adéquate pour l'application choisie en considérant la QoS de l'application et/ou des contraintes de temps d'exécution.

Notre approche est complémentaire avec les travaux existants puisqu'elle s'appuie sur des architectures auto-adaptatives. De plus, le modèle BFM est conçu de façon à ce que le bloc de diagnostic puisse mettre à jour les probabilités de défaillances des capteurs et de la QoS des applications afin d'adapter la décision aux différents événements externes et internes. Le bloc diagnostic recalcule les probabilités des HMs, alimentant ainsi les transitions du MDP, à chaque fois qu'une anomalie est observée à l'aide des moniteurs (capteurs physiques ou logiciels).

Le modèle BFM peut être défini par deux versions possibles, monolithique ou concurrente selon le choix de représentation de la mission à l'aide du modèle MDP.

4.3 Représentation du module de décision du modèle BFM

Dans cette section, nous présentons la façon de décrire une mission à l'aide d'un MDP. Nous considérons une mission de drone dont le but est de traquer une cible potentielle. Le drone suit une trajectoire (ensemble de waypoints) prédéfinie à partir de la base (point de départ) jusqu'à la zone de recherche de la cible (point d'arrivée). Une fois arrivé à la zone de recherche, le drone se met en mode détection et suivi de cible.

Durant la mission, différentes erreurs peuvent apparaître dues aux perturbations environnementales et aux défaillances matérielles et/ou logicielles. L'apparition de ces erreurs peut faire échouer la mission du véhicule autonome ou bien ralentir l'accomplissement de la mission dans le cas où les erreurs ne sont pas traitées. Nous nous concentrons sur la partie suivi de la cible de la mission tracking effectuée par un drone. Dans la spécification de la mission, nous considérons une seule version pour l'application de 'tracking'. Cependant dans le cas où une erreur sur l'application 'tracking' est observée (causer par un manque de luminosité par exemple), la mission va être interrompue et la cible perdue. Pour y remédier, nous considérons cinq variations possibles de l'application 'tracking' à partir de la version initiale V0. La Figure 4.2 montre le MDP associé à la nouvelle spécification de la mission de

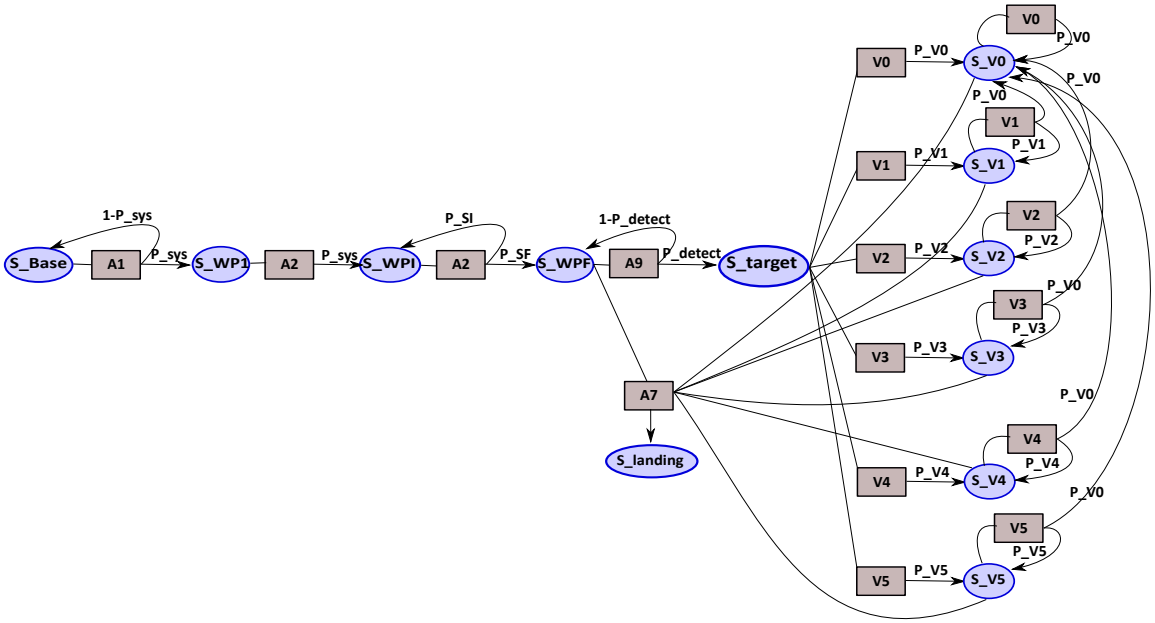


Figure 4.2: *MDP associé à la mission de tracking avec différentes versions de l'application 'tracking'.*

tracking.

- **Version 1 du 'tracking'** : ajout d'un pré-filtrage, à la version V0 de l'application 'tracking', utilisant un filtre à histogramme dans le but d'améliorer le contraste de l'image.
- **Version 2 du 'tracking'** : version du 'tracking' considérant une image de taille plus grande en faisant prendre de l'altitude au drone par exemple. Cela se traduit par une version plus performante de l'application de 'tracking' afin de détecter au mieux les points caractéristiques sur l'image traitée.
- **Version 3 du 'tracking'** : version initiale de l'application 'tracking' invoquant l'application 'extra-stabilisation' pour mieux stabiliser l'image.
- **Version 4 du 'tracking'** : version 1 du 'tracking' (pré-filtrage par histogramme) combinée avec l'application 'extra-stabilisation'.
- **Version 5 du 'tracking'** : version 2 du 'tracking' (changer la taille de la R.O.I) invoquée avec l'application 'extra-stabilisation'.

Maintenant, nous allons présenter la modélisation de la mission de tracking de deux manières différentes : une première modélisation avec un BFM monolithique et une seconde modélisation avec la version concurrente du modèle BFM.

4.3.1 Modèle BFM monolithique pour mission de drone

La version monolithique du modèle BFM consiste à décrire la totalité de la mission du drone sous forme d'un seul MDP global. Nous illustrons cette version

monolithique à travers la mission de tracking décrite ci-dessus.

Pour mener à bien la mission, nous avons besoin d'exécuter différentes actions. Certaines de ces actions assurent la partie navigation, d'autres la partie sécurité (safety) et enfin des actions pour la phase de suivi de la cible. Nous définissons par conséquent l'ensemble des actions de la mission.

1. Ensemble des actions :
   - A1 : décoller (take-off).
   - A2 : suivi de la trajectoire (ensemble des waypoints).
   - A3 : évitement d'obstacle.
   - A4 : détection d'obstacle par fusion de capteurs (Infra-rouge et Ultrason).
   - A5 : recherche de la zone 'T' pour atterrissage.
   - A6 : recherche de zone libre pour atterrissage d'urgence.
   - A7 : action d'atterrir utilisant l'asservissement par caméra.
   - A8 : action de re-planning et atterrissage.
   - A9 : détection de cible potentielle.
   - A10 : action de retour à la base.
   - A11 : détection d'obstacle par LIDAR.
   - A12 : version initiale de l'application 'tracking' (version V0).
   - A13 : version V1 de l'application 'tracking' (V0 + pré-filtrage par histogramme).
   - A14 : version V2 de l'application 'tracking' (V0 + re-dimension de l'image).
   - A15 : version V3 de l'application 'tracking' (V0 + extra-stabilisation).
   - A16 : version V4 de l'application 'tracking' (V1 + extra-stabilisation).
   - A17 : version V5 de l'application 'tracking' (V2 + extra-stabilisation). Pour la suite, nous allons nommer V_i les actions de A14 à A17.

   Nous définissons aussi l'ensemble des états décrivant les différents états de la mission. Ces états sont :

2. Ensemble des états de la mission :
   - S_base : état de départ à partir de la base.
   - S_WP1 : premier point de la trajectoire.
   - S_WPi : tous les points intermédiaires de la trajectoire.
   - S_WPf : dernier point de la trajectoire qui correspond à la zone de recherche de la cible à poursuivre.

— S_Obstacle_détecté : indique qu'un obstacle est détecté par LIDAR ou par fusion de capteurs.

— S_Fusion : indique que dans l'état détection d'obstacle, l'obstacle a été détecté par fusion de capteurs (capteur infra-rouge + capteur ultrason).

— S_WP_Évitement_Obstacle : représente l'ensemble des waypoints obtenus par l'action d'évitement d'obstacle.

— S_Zone_'T'_Ok : zone 'T' détectée pour l'atterrissage.

— S_Zone_'T'_Nok : zone 'T' non détectée.

— S_Zone_Ok : une zone atterrissage d'urgence est trouvée.

— S_Zone_Nok : zone atterrissage d'urgence non trouvée.

— S_Landing : représente l'état atterrissage.

— S_Cible : indique que la cible est détectée.

— S_Vi : représentent tout les états correspondant aux différentes versions de l'application 'tracking', pour i = 0...5.

Maintenant que l'ensemble des états et l'ensemble des actions de la mission sont définis, nous pouvons construire le modèle MDP représentant la mission de tracking en renseignant toutes les valeurs de probabilités des transitions nécessaires qui permettent de passer d'un état de la mission vers un autre en exécution une action a. Nous illustrons le modèle MDP de la mission en montrant des fragments du MDP à obtenir. La Figure 4.3 présente les différentes actions qui peuvent s'exécuter à partir des différents états constituant les points de trajectoire à suivre par le drone lors de la mission (WP1, WPi et WPf). Les actions A2 + A11 (respectivement suivi de la trajectoire et détection d'obstacle par LIDAR) et A2 + A4 (respectivement suivi de la trajectoire et détection d'obstacle par fusion de capteurs) sont utilisées pour la navigation, ce qui implique leurs exécutions à partir des états waypoints du modèle MDP. Selon la description de la mission à effectuer, nous ajoutons les actions adéquates à exécuter à partir de chaque état S_i du MDP global. Par exemple, l'action A5 (recherche de la zone 'T') peut être exécutée à partir des deux états WPi et WPf comparativement à l'action A9 (détection de la cible) qui ne peut être activée seulement une fois la zone de recherche de la cible atteinte (état WPf). Une fois la cible potentielle détectée, l'application 'tracking' est activée avec la version appropriée en fonction du contexte observé.

L'action de détection d'obstacle (avec LIDAR et avec fusion de capteurs) et l'action de recherche de zone pour atterrissage (zone 'T' et zone d'urgence) sont considérées comme des actions de sécurité (safety). Le fait de considérer les actions de détection d'obstacle et de recherche de zones d'atterrissage comme des actions de sécurité implique que la priorité donnée à la détection d'obstacle est plus élevée que la priorité de l'action recherche de zone d'atterrissage. L'objectif de la mission étant de suivre une cible le plus longtemps possible, deux stratégies de mission sont alors définies *'stratégie safety'* et *'stratégie mission first'*. Selon le choix de l'utilisateur

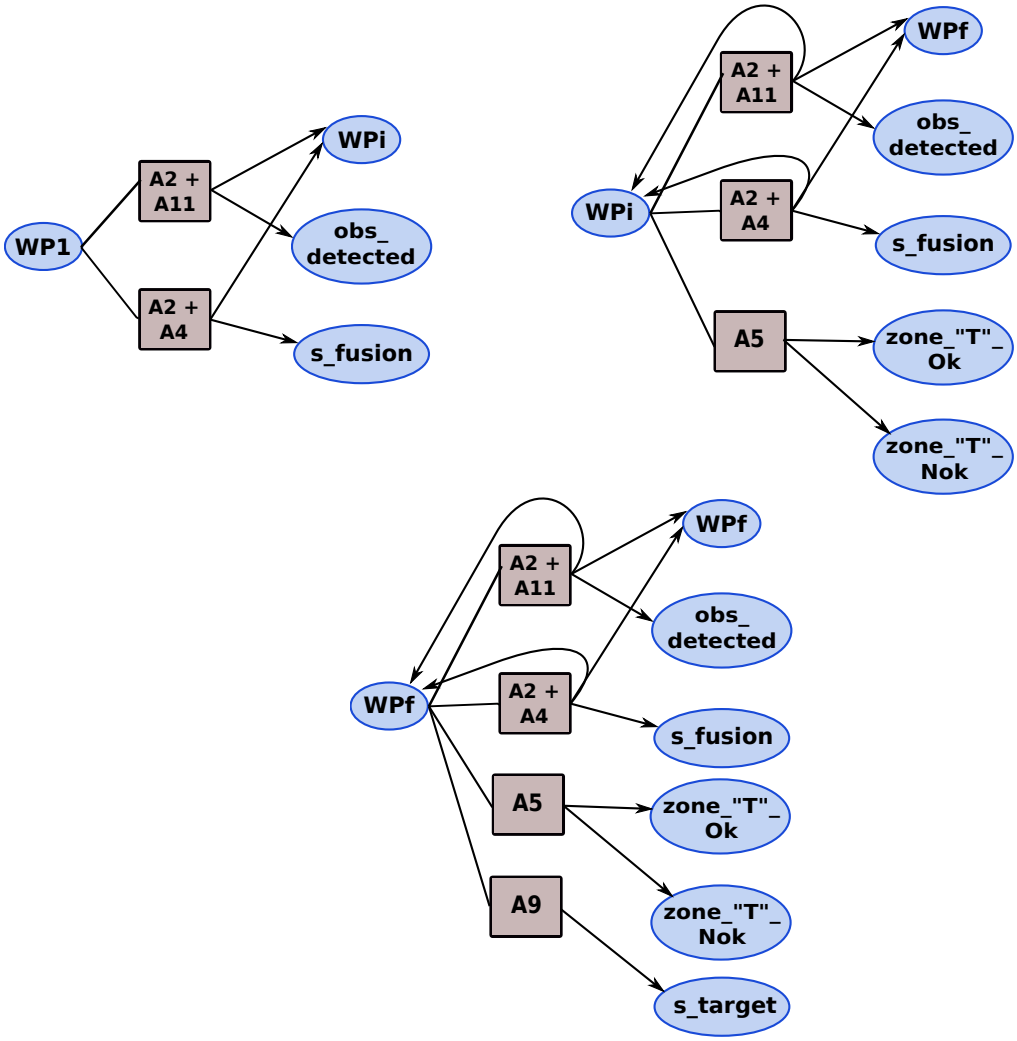
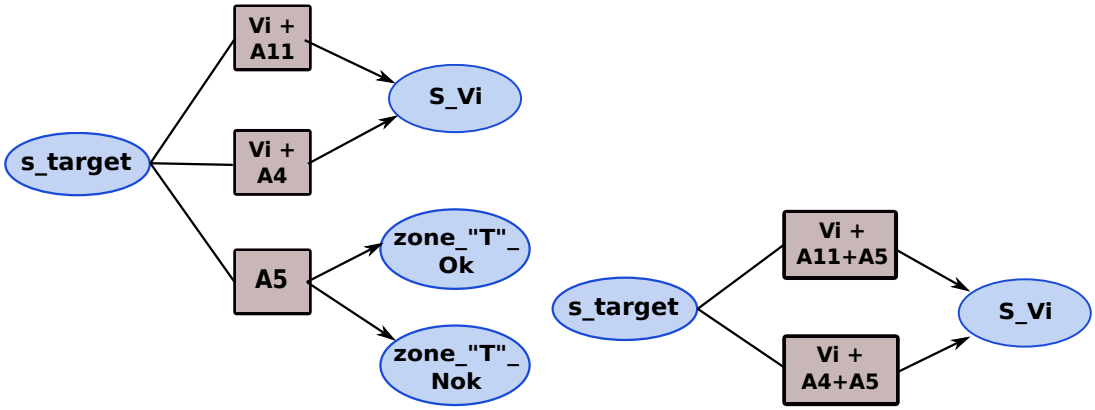


Figure 4.3: *Fragment du modèle MDP à partir des états waypoints de la mission de tracking.*

sur la stratégie du déroulement de la mission, deux représentations de la phase de tracking de la mission sont possibles, comme montré dans la Figure 4.4(a) et Figure 4.4(b). Avec la *'stratégie mission first'*, les applications 'tracking' et 'détection d'obstacle' sont exécutées simultanément ; contrairement au cas *'stratégie safety'* où les applications 'tracking', 'détection d'obstacle' et 'recherche de zone d'atterrissage' sont toutes les trois exécutées simultanément.

Les valeurs de probabilités portées par les transitions du modèle MDP sont produites par le module de diagnostic du modèle BFM (HM capteur, HM système et HM application) ou par les applications embarquées sur le drone en récupérant le facteur de confiance de l'application (ex : application 'tracking', probabilité de détection d'obstacle, etc). La valeur des rewards attribué à chaque application de l'ensemble



((a)) *Cas 'stratégie mission'.* ((b)) *Cas 'stratégie sécurité'.*

Figure 4.4: *Phase de tracking de la mission avec les deux stratégies définies.*

des actions du modèle MDP est fixée selon la stratégie choisie pour le déroulement de la mission.

La description de la mission à l'aide d'un modèle MDP a pour objectif de décider de l'action à exécuter à partir de chaque état de la mission. Dans le cas de la mission de suivi de cibles, l'objectif étant de maximiser le temps de tracking de la cible détectée, il nous faut donc résoudre le MDP monolithique de la mission constitué de 21 actions et de 20 états. Par conséquent, nous avons 21 matrices de transitions, où chaque matrice est de taille 20 x 20 (nbr_états x nbr_états), et une matrice rewards de taille 20 x 21 (nbr_états x nbr_actions). Nous pouvons constater que la version monolithique du modèle BFM est complexe que ce soit en termes de spécification ou en termes de résolution (nombre de matrices à manipuler). Dans la suite, nous verrons comment réduire la complexité du modèle BFM monolithique en introduisant les BFM concurrents afin de répondre aux objectifs suivants :

— Simplifier la conception au niveau expertise des concepteurs.

— Éviter les erreurs de conception.

— Tirer profit de l'expertise des concepteurs.

Nous abordons dans la section qui suit, la version concurrente du modèle BFM.

4.4 Modèle BFM concurrent pour une mission de véhicule autonome

Nous proposons dans cette section une méthodologie *scalable* et concurrente pour spécifier l'intégralité de la mission du véhicule autonome exécutant différentes applications (tracking, détection et évitement d'obstacle, recherche de zone d'atterrissage, etc). Comme vu avec la version monolithique de notre modèle BFM, l'inconvénient

du modèle MDP réside dans la dimension du problème à résoudre. En effet, plus le nombre d'états et d'actions décrivant le problème est grand, plus la complexité du MDP augmente. Cette complexité réside, d'un côté dans la spécification du problème sous forme d'un MDP et d'un autre côté dans la résolution du MDP. Car la résolution d'un MDP consiste à explorer la totalité de l'espace des états à chaque itération de l'algorithme de résolution dans le but de décider de l'action à exécuter à partir de chaque état de la mission.

Afin de réduire la complexité du modèle MDP, trois principales approches sont présentées dans la littérature [Corona-Xelhuantzi et al., 2009] : *factorisation*, *abstraction* et *décomposition*.

— *Factorisation* : consiste à représenter le MDP de manière factorisée et compacte. Les actions sont décrites comme ayant un effet sur des variables spécifiques (les états) dans certaines conditions. Cela induit implicitement à une fonction de transition représentée par un DBN (réseau Bayésien dynamique) [Ghahramani, 1997]. [Hoey et al., 1999] utilise des diagrammes de décision pour représenter les fonctions de transitions et de coûts. L'inconvénient de la représentation factorisée d'un MDP est qu'elle ne prend pas en compte l'exécution simultanée des actions.

— *Abstraction* : consiste à regrouper chaque ensemble d'états partageant le même comportement en un sous-ensemble et relier chaque sous-ensemble à un état abstrait. Cette approche s'apparente aux MDPs hiérarchiques.

— *Décomposition* : consiste à décomposer le problème à résoudre en sous-problèmes. Une première approche est de diviser l'ensemble des actions en sous-ensembles de tâches [Meuleau et al., 1998]. La politique optimale de chaque sous-problème est calculée hors-ligne, ensuite dans la phase en-ligne une heuristique est utilisée avec les valeurs de coûts des politiques optimales (de la phase hors-ligne) pour déduire les actions finales à exécuter. Une seconde approche consiste à diviser l'ensemble des états en sous-ensembles en fonction des étapes du problème (ex : phases d'une mission) [Laroche et al., 2001]. Chaque sous-ensemble d'états est décrit par un MDP. Les différents sous-ensembles sont reliés entre eux via des états abstrait appelés intersection. Cette approche de décomposition permet l'exécution des actions simultanément, par contre elle ne traite pas le cas des actions antagonistes.

L'idée ici est d'utiliser l'approche de décomposition et d'exprimer séparément les différents sous-ensembles avec différents BFMs en prenant en compte les défaillances potentielles des capteurs (HM capteur), anomalies au niveau du système (HM système) et les différentes versions possibles pour les applications embarquées (HM application). Cette décomposition rend la spécification plus maniable, mais l'utilisation de BFM (MDPs) concurrents peut conduire à la résolution de possible conflits dans le cas ou différents actions sont choisies à partir d'un même état.

Notre approche tire avantage de la formalisation de notre modèle BFM qui permet d'interagir avec des experts qui peuvent orienter sur la façon de résoudre les

conflits rencontrés en considérant la stratégie de déroulement de la mission (prudente ou risquer) et le contexte.

4.4.1 Principe décomposition en BFMs concurrents

Pour une mission donnée, nous définissons l'ensemble des actions garantissant le bon déroulement de la mission. Chaque action a un but particulier. Par conséquent nous avons la possibilité d'avoir des sous-ensembles d'actions répertoriées selon leurs fonctionnalités.

Le découpage des applications selon leurs fonctionnalités permet de décomposer la mission globale en différentes phases, ce qui facilite la description de la mission. Nous définissons trois catégories principales de fonctionnalités :

— **Fonctionnalité de base** : regroupe toutes les applications de base nécessaires au fonctionnement de départ du véhicule autonome. Dans le cas d'un drone, les applications de navigation (take-off, suivi et calcul de trajectoire, etc.) font partie des fonctions de base assurant le bon déplacement du drone.

— **Fonctionnalité de sécurité** : nous retrouvons toutes les applications qui assurent la sécurité du véhicule autonome et de la mission. Parmi ces applications, nous considérons la détection et l'évitement d'obstacle, la détection de zones libre ou spécifique pour atterrir (cas de drone), etc.

— **Fonctionnalité applicative** : correspond aux applications qui permettent de mettre en œuvre l'objectif global de la mission. Par exemple : si la mission du véhicule autonome est de poursuivre une cible, alors l'application de détection et tracking remplie la fonctionnalité applicative de la mission.
Chaque catégorie de fonctionnalité peut être décrite comme une phase de mission. Pour garantir la cohérence dans la description de la mission en plusieurs phases, nous pouvons intégrer les applications de sécurité dans la phase de navigation (fonctionnalité de base) ou dans la phase applicative.

Pour chaque sous-ensemble fonctionnel d'applications, nous construisons un modèle BFM afin d'obtenir une spécification la plus cohérente possible correspondant à la description globale de la mission. Chaque MDP (partie décision) du modèle BFM est construit en se basant sur des états observables de la mission et des actions permettant de passer d'un état à un autre. Une action peut correspondre à une commande de navigation ou à une application s'exécutant sur le système embarqué. La Figure 4.5 montre une vue globale de BFMs concurrents et de décomposition d'une mission en phases. Le 'BFM-Navigation' correspond aux fonctionnalités de base, regroupées dans un seul modèle BFM, permettant le fonctionnement de base du véhicule autonome tel que la navigation depuis la base jusqu'à son objectif. Le 'BFM-Safety' représente le BFM regroupant les fonctionnalités de sécurité qui assurent le bon déroulement de la mission en terme de sécurité du véhicule et de la

mission. Enfin, nous avons le 'BFM-Applicatif' qui représente la phase de réalisation de l'objectif de la mission qui nécessite des applications dédiées.

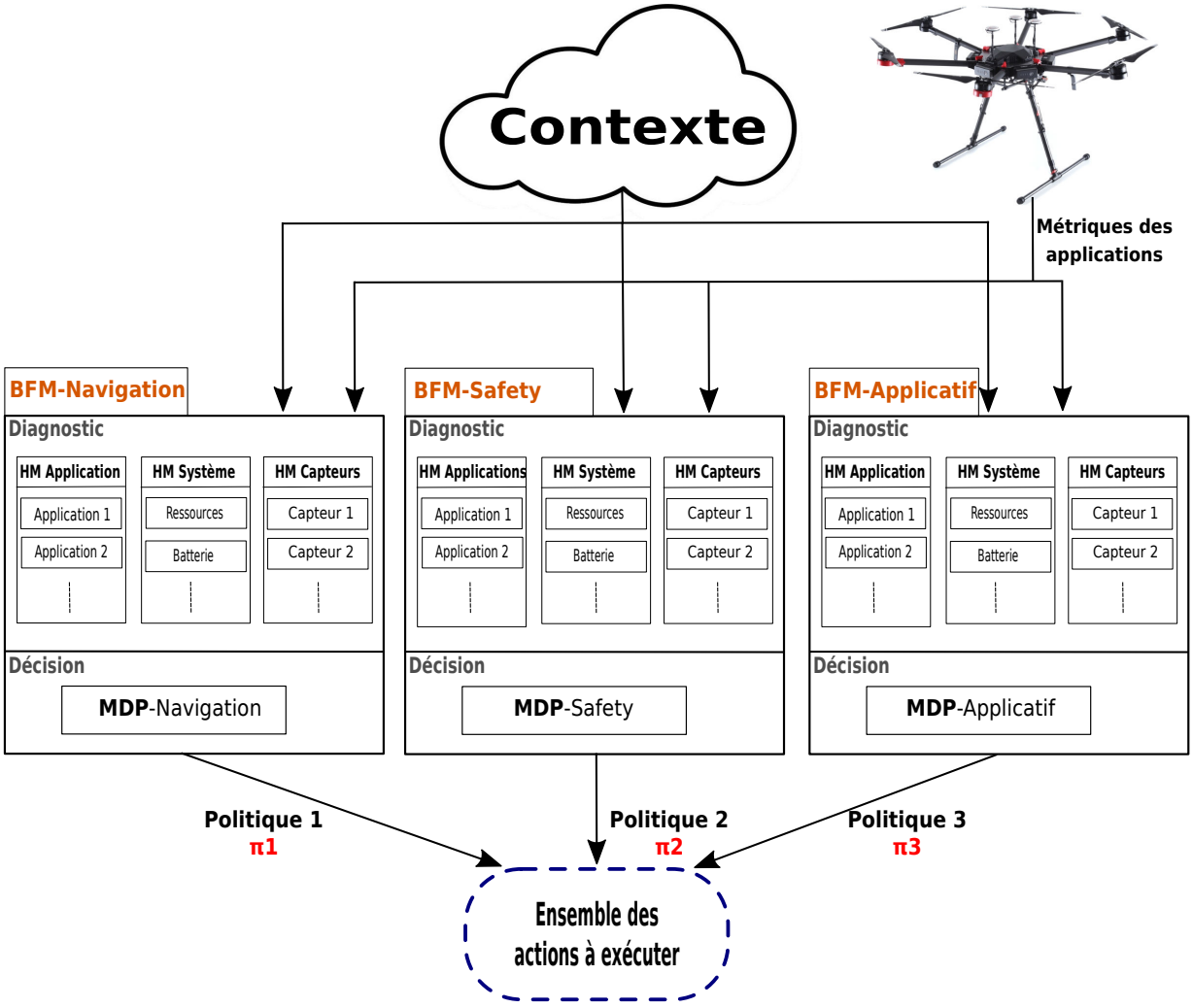


Figure 4.5: *Principe décomposition en BFMs concurrents.*

Les différents MDPs de la version concurrente du modèle BFM ne sont pas totalement indépendants et l'exécution concurrente impose que nous définissons certaines contraintes. Ces contraintes sont :

— Tout les modèles BFM concurrents ont le même état de départ et le même état final.

— Les différents BFMs doivent être synchronisés, ce qui implique des états communs aux modèles BFMs définis.

4.4.2 Exemple de BFMs concurrents

Reprenons la mission de tracking décrite précédemment. L'ensemble des actions permettant le succès de la mission contient des applications dédiées aux différentes fonctionnalités de la mission. Par conséquent, la mission peut-être décomposée en trois phases principales : navigation, atterrissage et tracking. Chaque phase de mission est décrite par un MDP.

L'état de départ et l'état final de chacun des MDPs sont : S_base (drone sur la base de départ) et S_landing (drone atterri). La synchronisation entre les différents MDPs est faite à l'aide des états communs suivant :

— S_WP1 : premier point de la trajectoire (en vol).

— S_WPi : points intermédiaires de la trajectoire.

— S_WPf : point final de la trajectoire (zone de recherche de la cible à traquer).

Les principaux MDPs définis pour la mission de tracking sont présentés dans ce qui suit :

MDP navigation :

Le modèle MDP décrit dans cette partie correspond à la phase de navigation de la mission de tracking. Dans cet MDP, nous retrouverons les actions de navigation et de sécurité, comme décrit dans le Figure 4.6.

1. Ensemble des états de la phase navigation :
   — Obstacle_Détecté : indique qu'un obstacle est détecté par le LIDAR ou bien par la fusion de capteurs.
   — S_Fusion : représente l'état de détection d'obstacle utilisant la fusion de plusieurs capteurs de type short/long infra-rouge (IR) et de type utrason (US).
   — WP_Évitement : points de trajectoire produits par l'action d'évitement d'obstacle.
   — S_Collision : représente l'état où le drone entre en collision avec un obstacle ou bien crash causé par une énergie insuffisante.
2. Ensemble des actions de la phase navigation :
   — A1 : take-off.
   — A2 : action de parcourir les points de la trajectoire.
   — A11 : action de détecter la présence d'obstacle à l'aide du LIDAR.
   — A3 : application d'évitement d'obstacle détecté.
   — A4 : application de détection d'obstacle utilisant la fusion de données de capteurs (IR et US).
   — A7 : atterrissage.

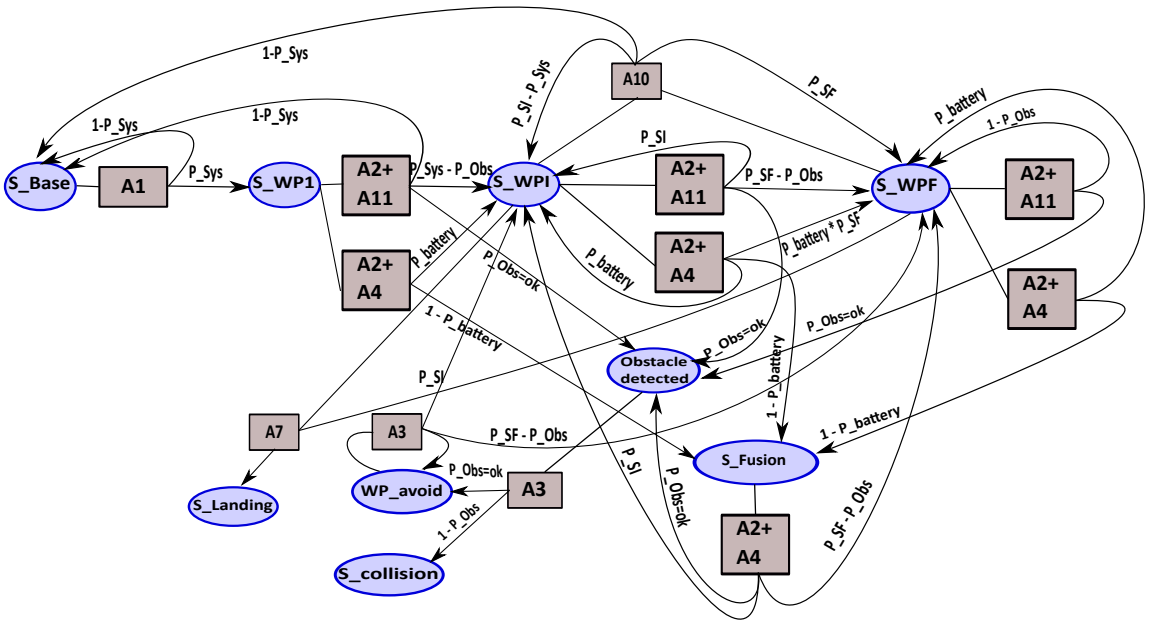


Figure 4.6: *Modèle MDP représentant la phase de navigation de la mission de tracking.*

— A10 : action de retour à la base.

3. Probabilités portées par les transitions :
   — P_Sys : représente la probabilité de bon fonctionnement du système.
   — P_Si : représente la probabilité de rester dans l'état WPi (parcourir tout les points intermédiare de la trajectoire).
   — P_Sf : indique la probabilité de quitter l'état WPi pour passer à l'état WPf.
   — P_Obs : renseigne sur la probabilité qu'un obstacle a été détecté.
   — P_Batterie : estimation du niveau de l'énergie restante.

MDP atterrissage

Le modèle MDP de la phase d'atterrissage contient essentiellement les actions nécessaire pour effectuer un atterrissage, la Figure 4.7 montre cela. Ces actions font partie aussi des actions assurant la sécurité du drone et de la mission, telle que : l'action de détection d'obstacle.

1. Ensemble d'états de la phase d'atterrissage :
   — Zone_T_Ok : indique qu'une zone 'T' est trouvée.
   — Zone_T_Nok : indique que la zone 'T' n'a pas été trouvée.
   — Zone_Ok : une zone libre est détectée pour un atterrissage d'urgence.
   — Zone_Nok : la zone pour atterrissage d'urgence n'a pas été trouvée.

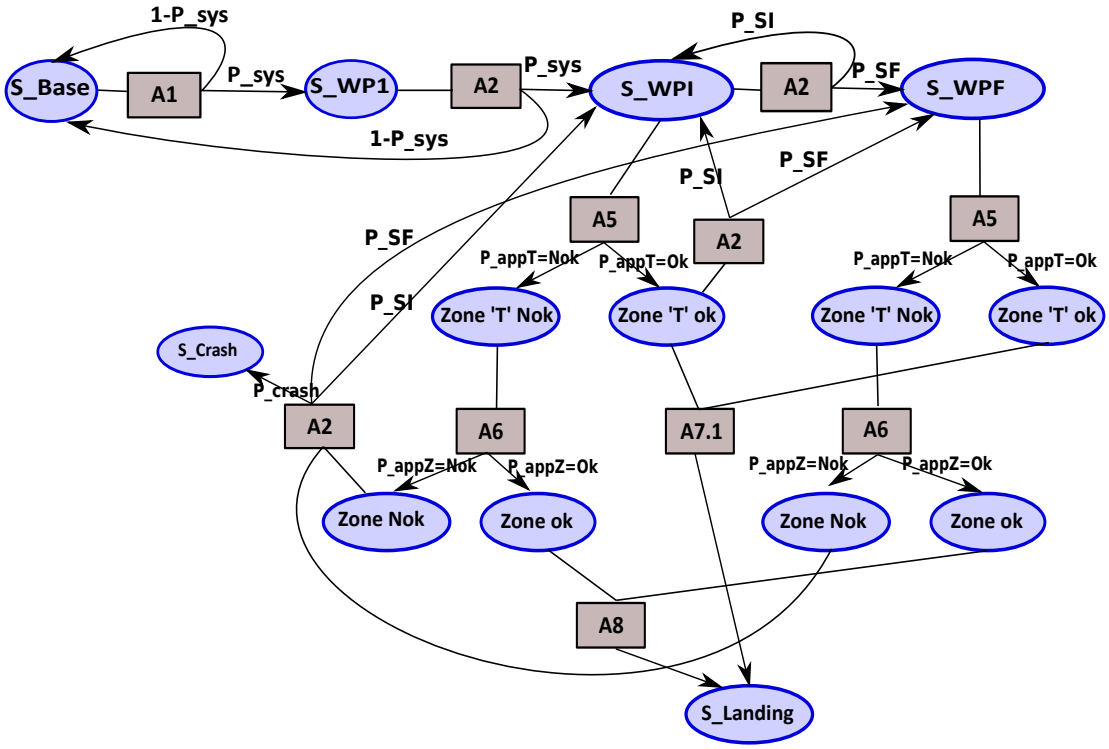


Figure 4.7: *Modèle MDP décrivant la phase d'atterrissage de la mission de tracking.*

2. Ensemble des actions de la phase de d'atterrissage :
   - A2 : correspond à l'action de parcours des points de la trajectoire.
   - A5 : représente l'application de recherche de zone 'T'.
   - A6 : application de recherche de zone libre pour atterrissage d'urgence.
   - A7 : correspond à l'action d'atterrir en utilisant l'asservissement caméra.
   - A8 : action de re-planning et d'atterrissage.
3. Fonction de transitions associées à la phase de d'atterrissage :
   - P_App_T : est la probabilité qu'une zone 'T' est trouvée.
   - P_App_Z : renseigne la probabilité qu'une zone libre pour atterrissage d'urgence est trouvée.
   - P_Crash : probabilité de crash.

MDP tracking

Le troisième et dernier MDP est celui de la phase de suivi de la cible. Le MDP de la phase tracking est illustré par la Figure 4.2. Pour cette partie de la mission, nous considérons plusieurs versions de l'application 'tracking' pour ne pas devoir arrêter la mission dès la première défaillance observée sur l'application de 'tracking' et de ce fait augmenter le temps de suivi de la cible détectée.

1. Ensemble d'états de la phase de tracking :
   En plus des états départ/arrivé, indiquant le début de la mission et l'arrivée à la zone de détection de la cible potentielle, des états communs entres les différents MDPs concurrents (états waypoints). nous définissons les états suivants spécifique aux suivi de la cible.
   - S_Cible : représente le fait que la cible soit détectée.
   - S_Vi : représente tout les états qui correspondent aux différentes versions de l'application 'tracking'.
2. Ensemble des actions :
   - A9 : action de détection de cible potentielle à traquer.
   - V0 : version initiale (nominale) de l'application de 'tracking'.
   - V1 : V0 + pré-filtrage par histogramme.
   - V2 : V0 + augmenter la taille de l'image.
   - V3 : V0 + extra-stabilisation.
   - V4 : V1 + extra-stabilisation.
   - V5 : V2 + extra-stabilisation.
3. fonctions de transitions du MDP :
   - P_Detect : probabilité que la cible à traquer soit détectée.
   - P_Vi : renseigne sur la probabilité d'activation des versions Vi dans un contexte donné en prenant en compte la disponibilité des ressources (CPU et FPGA) et les performance souhaitées.
   - Rajoutons à ces probabilités d'autres transitions qui garantissent de passer d'une version de 'tracking' à une autre.

Toutes les probabilités (fonctions de transitions) définies au sein des trois MDPs concurrents (navigation, atterrissage et tracking) sont calculées par : les différents modules de 'Health Management' du bloc diagnostic du modèle BFM (HM applications, HM système et HM capteurs) ; les métriques renvoyées par les différentes applications embarquées (détection d'obstacle, tracking, etc) ; etc.

Tous les états des MDPs définis plus haut sont observables par GPS pour les états points de la trajectoire, monitoring du niveau de la batterie pour l'état S_Fusion du MDP navigation et les autres états sont observables grâce aux métriques applications (tracking, détection et évitement d'obstacle, recherche de zone d'atterrissage).

La Figure 4.8 récapitule la version concurrente du modèle BFM appliqué à la mission de tracking. Chacune des phases de la mission de tracking est représentée par un modèle BFM. Chaque modèle BFM est constitué de la partie décision décrivant une phase de mission sous forme de MDP et de la partie diagnostic évaluant l'état de santé des capteurs, du système et la QoS des applications nécessaire au bon

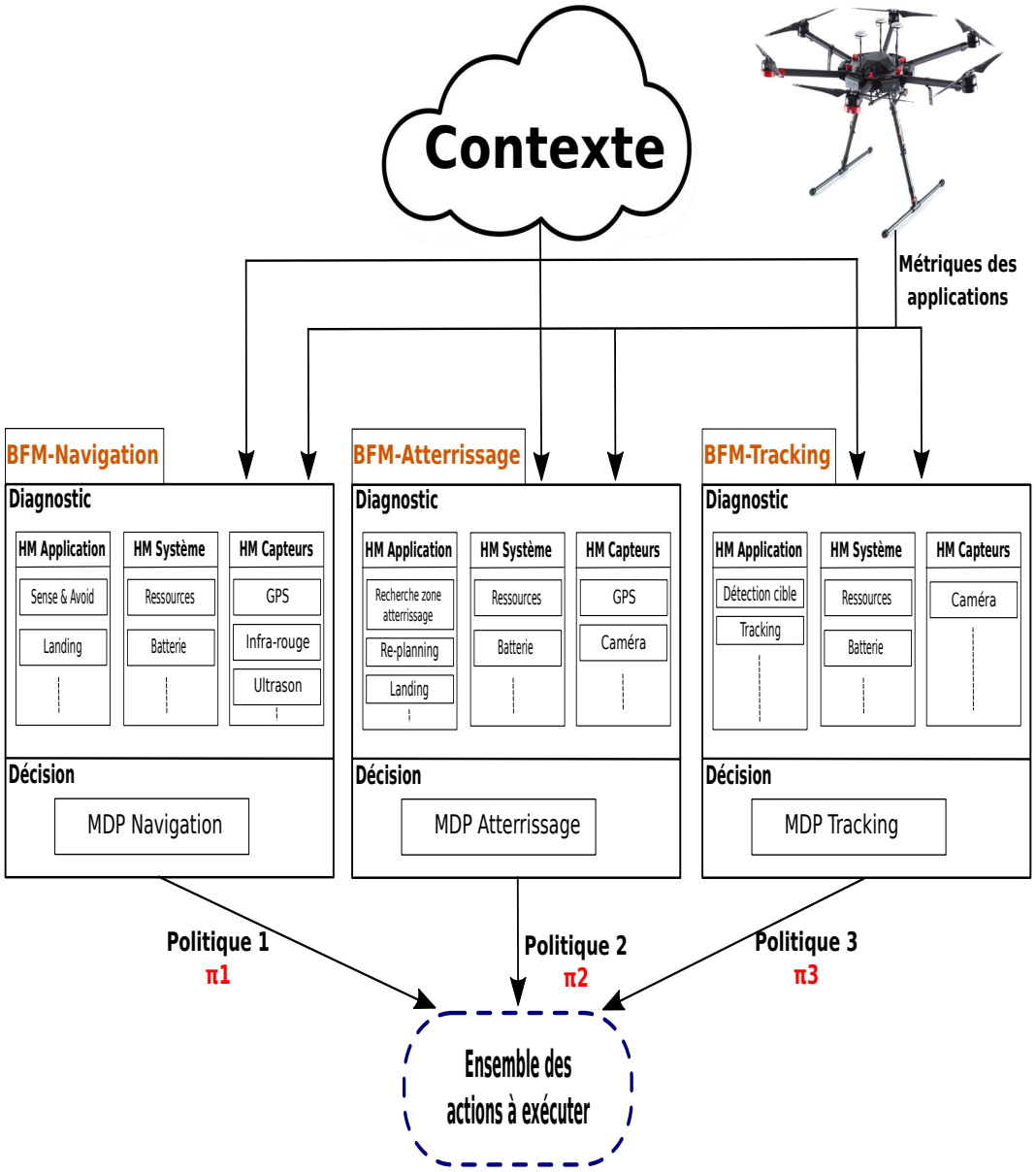


Figure 4.8: *BFM concurrents illustrant la mission de tracking.*

déroulement de la phase de mission. Par exemple, lors de la phase de navigation, les principales applications exécutées et pour lesquelles la QoS doit être évaluée par le module HM Application sont : 'détection d'obstacle', 'évitement d'obstacle' et 'atterrir'. L'ensemble des capteurs dont on aura besoin pour la phase de navigation sont : 'GPS' pour le suivi de la trajectoire, 'infra-rouge/ultrason' pour la détection d'obstacle par fusion de données de capteurs et 'LIDAR' pour la détection d'obstacle en utilisant le LIDAR ; l'état de santé de ces capteurs est calculé par le module HM Capteur. Enfin, le module HM Système nous renseigne sur le bon fonctionnement du système global incluant l'état et le niveau de la batterie ainsi que la consommation des ressources (CPU et FPGA).

Nous pouvons constater :

— La *scalabilité* apportée par le modèle BFM concurrent.

— La simplification de spécification de la mission à l'aide de MDP contrairement à la version monolithique.

— enfin la possibilité d'exécuter simultanément les différents MDPs décrivant la mission.

Le fait de décomposer la mission en plusieurs phases, selon l'objectif final de la mission, permet de réduire la complexité du BFM monolithique en termes de spécification de la mission (représentation de la mission avec un MDP) et en termes de résolution du modèle MDP, pour trouver l'ensemble des actions à exécuter selon le contexte observé, en exploitant le parallélisme offert par les architectures hybrides de type CPU/FPGA utilisées dans notre cas pour la résolution du modèle BFM.

4.4.3 Résolution de conflits

La représentation d'une mission de véhicule autonome à l'aide de BFM concurrent permet l'exécution en parallèle des différents MDPs concurrents et aussi faciliter la description de la mission en la décomposant en phases. Chaque phase de mission décrite par un MDP fournit une politique π (ensemble d'actions à exécuter) après résolution. Par conséquent, la résolution du BFM concurrent fournit autant de politiques que de MDPs définis, ce qui nécessite une gestion de conflits afin d'en déduire une seule politique pour la mission.

Pour des raisons de synchronisation des différents BFMs concurrents, nous avons défini des états communs (ex : états waypoints) entre les différents MDPs concurrents. Suite à cette synchronisation, des conflits peuvent apparaître entre les politiques obtenues par la résolution de chacun des MDPs (navigation, atterrissage et tracking). Par conséquent, nous définissons deux types de conflits possibles : *conflit de comportement* et *conflit de ressources*.

Conflit de comportement

Un conflit de type *comportement* apparaît entre deux ou plusieurs actions induisant des comportements antagonistes (contradictoires) lors de l'exécution.

Comme les différents BFMs peuvent être exécutés en parallèle, les actions décidées par chaque MDP peuvent être incompatibles en termes de comportement. Par exemple, si à partir d'un état, un premier MDP choisit l'action 'éviter l'obstacle' et qu'un second MDP choisit, pour le même état, l'action 'atterrir' alors les deux actions ne pourront pas s'exécuter simultanément. Pour résoudre le conflit de comportement, nous nous appuyons sur l'approche suivante :

— Lister les différents cas de conflit de comportement entre deux ou plusieurs actions.

— Pour chaque incompatibilité, proposer une solution respectant la stratégie choisie pour la mission et qui repose sur l'analyse d'un expert.

Cette approche est réalisable hors ligne puisque d'un côté le nombre de MDPs spécifiant une mission de véhicule autonome est connu et limité et d'un autre côté le nombre de stratégies de mission est aussi limité.

Dans le cas de la mission de tracking, nous avons identifié les principaux conflits de comportements, correspondants à des actions antagonistes, qui sont les suivants :

— Évitement d'obstacle et tracking ;

— Atterrissage et tracking ;

— Évitement d'obstacle et atterrissage ;

— Retour à la base et tracking ;

— Retour à la base et atterrissage.

Ces actions sont antagonistes parce qu'elles provoquent des mouvements différents et incompatibles pour le drone. Pour résoudre ce conflit de comportement, nous définissons, avec l'aide d'un expert de la mission, les priorités des actions qui s'exécutent durant la mission en considérant le choix de la stratégie de la mission. Par exemple, dans le cas où la mission est exécutée avec une stratégie favorisant la sécurité, les actions de sécurité sont sélectionnées en premier. Par conséquent, si à partir d'un même état S, le MDP1 choisit l'action d'évitement d'obstacle et que le MDP2 choisit l'action tracking alors l'action à retenir et à exécuter est l'évitement d'obstacle car la mission est exécutée avec la stratégie 'safety' et dans ce cas l'application d'évitement d'obstacle est prioritaire à l'application 'tracking'. Dans le cas contraire où la mission est réalisée avec la stratégie favorisant la continuité de la mission (suivre la cible le plus longtemps possible), l'action de tracking aurait été plus prioritaire que les autres actions.

Dans le processus de décision visant à résoudre ces conflits, plusieurs critères peuvent être pris en compte tel que : le risque de perdre le drone (crash ou panne),

risque lié à l'environnement de la mission ainsi que la réalisation de la mission. Dans ce cas, la décision prise par le modèle de décision (MDP) est dépendante du contexte et donc le modèle de décision doit être résolu en ligne après avoir répertorier et intégrer hors ligne les données d'expertise (connaissance d'un expert des applications réalisant la mission).

Le nombre de stratégies peut être étendu à N stratégies différentes selon la mission étudiée. Ce qui rendra probablement plus complexe la gestion de conflit que nous proposons. Dans ce cas, d'autres approches peuvent être utilisées telle que les approches multi-critères incluant des préférences humaines.

Conflit de ressources

Nous nous concentrons ici sur le conflit dit '*conflit de ressources*' concernant les applications embarquées sur un système autonome. Dans notre cas d'étude, un conflit de ressource est un conflit où une ressource est utilisée par plusieurs applications en même temps. La ressource concerne, dans notre cas, les ressources de calculs (CPU et FPGA). Cependant, la ressource peut être étendue à d'autres types de ressources comme les capteurs. La proposition de gérer les conflits de ressources est originale et importante ; car souvent les processus de gestion de mission pour systèmes autonomes ne prennent pas en comptes les ressources interne du système.

Pour résoudre le conflit de ressources de calculs, nous évaluons la possibilité de partager les ressources de calculs sur lesquelles s'exécutent les différentes versions algorithmiques des applications en se basant sur l'estimation de la consommation des ressources selon l'implémentation matérielle ou logicielle des différentes applications.

Reprenons la mission de suivi de cible, nous considérons deux types de variantes concernant les applications nécessaires à la réalisation de la mission. Ces variantes sont :

— **Variantes algorithmiques** : une application peut avoir différentes versions algorithmiques selon les différentes implémentations (matérielles ou logicielles) possibles. Les priorités entre les différentes versions algorithmiques sont définies en évaluant la QoS associée à l'application selon le contexte et l'occupation des ressources de calcul. Cette QoS est fournie par le module HM application du bloc diagnostic du modèle BFM.
Par exemple, pour l'application de tracking, différentes versions de tracking sont disponibles. Chaque version nécessite un taux de ressources (pourcentage charge CPU et nombre de tuiles FPGA) disponibles pour s'exécuter correctement. La consommation de ressources diffère d'une version à une autre d'une même application.

— **Variantes liées au support d'exécution** : différents supports tel que le CPU pour les versions logicielles et le FPGA pour les versions matérielles, peuvent être utilisés pour l'exécution des versions algorithmiques conduisant à de multiple configurations. Ces configurations sont estimées en termes d'oc-

cupations des ressources (nombre de tuiles occupées sur le FPGA et la charge du CPU) et aussi en termes de performances (ex : nombre de millisecondes nécessaire pour le traitement d'une frame pour une application de traitement d'images).

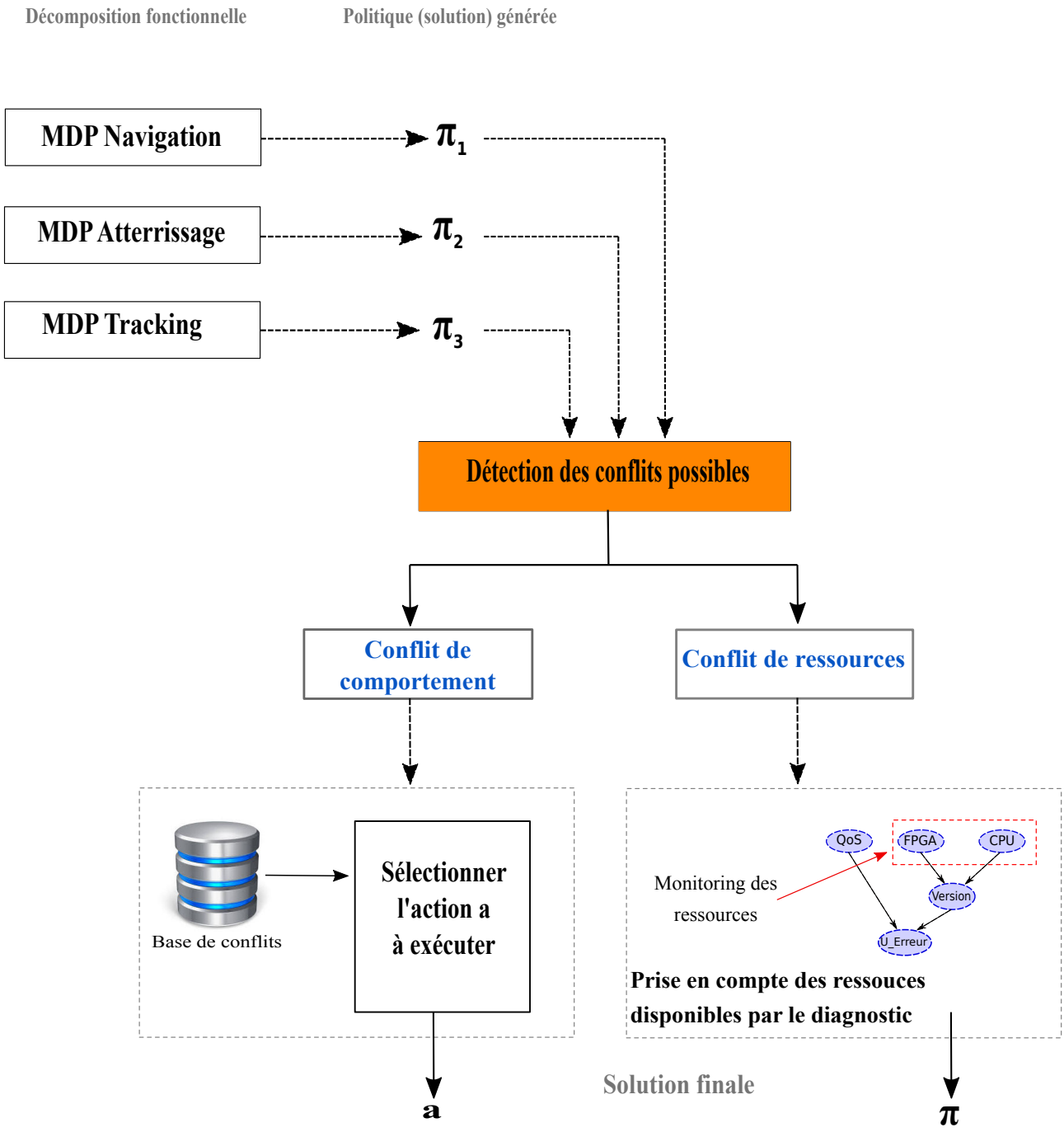


Figure 4.9: *Architecture de la gestion de conflits de comportement et de ressources.*

La Figure 4.9 résume comment la détection et la résolution de conflits est intégrée au modèle BFM présenté plus haut. La détection de conflit intervient après le calcul de la politique (π) à exécuter.

Après la décomposition fonctionnelle de la mission où chaque phase de mission est représentée par un MDP. Chaque MDP est résolu indépendemment des autres et génère une politique ($\pi 1$, $\pi 2$, $\pi 3$) par MDP. Les différentes politiques obtenues vont être comparées entre elles afin de détecter des possibles conflits. Par exemple, la mission de suivi de cible est décomposée en trois phases décrites chacune par un MDP (MDP navigation, MDP atterrissage et MDP tracking). Après résolution de ces trois MDPs, nous obtenons trois politiques. Le but de notre modèle étant d'obtenir à la fin qu'une seule et unique décision (politique) à propos des applications à exécuter pour la mission. Nous devons nous assurer qu'il n'y est pas de conflit entre les différentes applications des différentes politiques avant d'en déduire la politique finale.

Dans le cas de conflit de comportement, pour un même état du MDP, on vérifie si les applications à exécuter générées par les différents MDPs conduisent à un comportement contradictoire. Cette vérification est faite en s'appuyant sur la base de conflits de comportement définie hors ligne par un expert. Parmi les applications en conflit détectées, une seule application sera sélectionnée selon les priorités définit entre les différentes applications et selon la stratégie de la mission (safety ou mission first).

Dans le cas de conflit de ressources, l'apparition des conflits entre les applications ou bien les versions algorithmiques des applications a été anticipé et traité au niveau du module de diagnostic. Comme exposé dans le Chapitre 3, lors de l'évaluation de la QoS d'une application embarquée, nous prenons en compte non seulement les incertitudes liées au contexte mais aussi la consommation des ressources de calculs tel que la charge CPU et l'occupation du FPGA en termes de tuiles.

Une fois les conflits de comportement et de ressources détectés et corrigés, le modèle de décision renvoie une seule politique globale π à exécuter pour mener à bien la mission.

4.4.4 Illustration de la *scalabilité* du modèle BFM

Un véhicule autonome peut effectué différentes missions et avoir différents objectifs. De même une mission peut évoluer en lui ajoutant de nouvelles fonctionnalités à celles de base ou bien ajouter de nouveaux capteurs.

Dans cette section, nous allons montrer la *scalabilité* de notre modèle BFM ainsi que son adaptabilité à tout type de mission. Nous allons illustrer cela à travers trois points importants : changement de mission, ajout de capteurs et ajout de nouvelles fonctionnalités.

— **Changer de mission** : est définie par le changement de l'objectif de la mission et donc de la fonctionnalité applicative de la mission. Par exemple, changer la mission de tracking (vu précédemment) par une mission de monitoring d'in-

frastructure. Ce qui change alors c'est l'application réalisant l'objectif de la mission, ce qui implique de remplacer l'application de 'tracking' par une application de monitoring.
La modification à opérer sur le modèle BFM se situe au niveau de la fonctionnalité applicative vu que les autres fonctionnalités (de base et de sécurité) restent inchangées. Il faudra adapter la partie diagnostic du modèle BFM en renseignant les capteurs nécessaires à la réalisation du nouvel objectif de la mission. Enfin, mettre à jour la partie décision du modèle BFM en utilisant le MDP associé à la nouvelles fonctionnalité.

— **Ajouter des capteurs** : définit par l'ajout d'un ou plusieurs capteurs au système autonome dans le but de réaliser sa mission. Par exemple, ajout d'un LIDAR. Dans ce cas, nous mettons à jour le modèle BFM en incrémentant la partie diagnostic par l'ajout de l'évaluation de l'état de santé du LIDAR dans la catégorie 'HM Capteurs'.

— **Ajout de méthodes de sécurité** : si une nouvelle application de sécurité est ajoutée à la liste des applications de sécurité existante alors le modèle BFM sera adapté en conséquence. Deux changement sont alors requis au niveau du modèle BFM :

  — Ajouter un 'HM application' dans la partie diagnostic du BFM associé à l'application ajoutée. Le 'HM application' servira à évaluer la QoS de l'application et si nécessaire estimer la version de l'application à exécuter.

  — Pour la partie décision du modèle BFM, intégrer la nouvelle méthode de sécurité dans le MDP 'safety'. Cependant, il faudra s'assurer que la synchronisation entre les différents MDPs du BFM concurrent est toujours vérifiée. Il n'est pas impossible d'ajouter cette nouvelle méthode de sécurité au 'MDP navigation' (MDP avec les fonctionnalités de base) afin de garantir la cohérence de la mission et la synchronisation des différents BFMs concurrents.

La Figure 4.10 illustre la scalabilité du modèle BFM et la démarche à suivre pour adapter le modèle BFM aux modifications qui peuvent être apportées à la mission de base (modification de l'objectif de la mission, ajout de capteurs et ajout d'applications de safety).

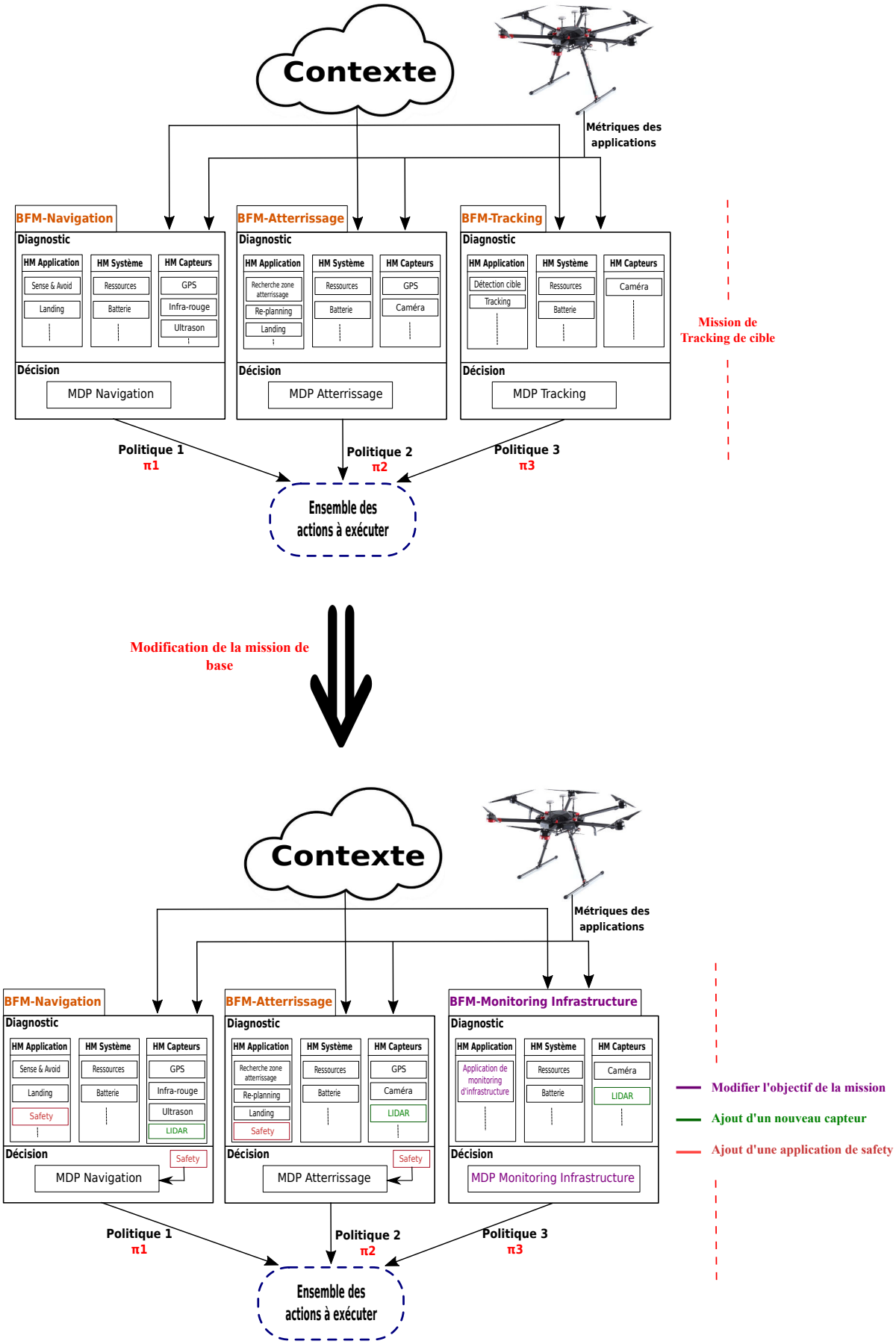


Figure 4.10: *Illustration de la scalabilité du modèle BFM.*

4.5 Conclusion

Dans ce chapitre, nous avons exposé les contributions apportées à la prise de décision par un système autonome. Nous avons proposé un modèle BFM qui s'appuie sur le processus de décision markovien (MDP) pour la prise de décision combinée avec un modèle Bayésien pour le diagnostic. Souvent, la difficulté de l'utilisation de modèles de décision probabilistes est de déterminer et définir les valeurs de probabilités des transitions d'un état à un autre. La première contribution consiste à alimenter les transitions du MDP par les probabilités calculées par le module de diagnostic du modèle BFM.

La prise de décision dans le cas d'un système autonome et adaptatif doit être réactive aux événements externes ou internes, ce qui implique l'importance de la complexité du modèle de décision en terme de spécification de la mission du véhicule autonome. La deuxième contribution est de proposer deux versions possibles du modèle BFM ; une version monolithique et une version concurrente facilitant ainsi la spécification de la mission et offrant une exécution en parallèle des BFMs concurrents. La version concurrente du modèle BFM inclut une gestion de conflits qui peuvent apparaître entre les actions des politiques générées par les BFMs concurrents.

Dans les travaux décrits dans la littérature [Vanegas et al., 2017], les probabilités sont obtenues à l'aide de lois et fonctions de probabilités à définir selon le problème étudié. La particularité de notre travail réside dans l'utilisation des BNs pour alimenter le modèle MDP de probabilités calculées par inférence. Aussi, nous proposons une version concurrente de notre modèle BFM qui permet de générer un plan de mission tout en résolvant les conflits de ressources et de comportements possibles. Ce travail a abouti à une publication en revue [Hireche et al., 2018b].

Nous avons fait une comparaison en terme de temps d'exécution entre le BFM monolithique et concurrent. La Table 4.1 suivante montre les temps obtenus. Nous constatons que le BFM-monolithique prend beaucoup plus de temps que n'importe quel BFMs concurrents (navigation, atterrissage et tracking). Le BFM-monolithique prend 5 fois plus de temps que le BFM-navigation et 3 fois plus de temps que le BFM-atterrissage et BFM-tracking pour produire une décision.

	Nombre d'états	Nombre d'actions	temps d'exécution (ms)
BFM-monolithique	20	21	10.159
BFM-navigation	9	7	2.004
BFM-atterrissage	10	6	2.781
BFM-tracking	12	10	3.537

Tableau 4.1: *Comparaison entre le temps d'exécution du BFM monolithique et les BFMs concurrents.*

- Chapitre 5 -

Implémentation embarquée du modèle BFM

Sommaire

5.1 Introduction

Nous présentons dans ce chapitre, nos contributions apportées au niveau de l'implémentation du modèle BFM proposé dans le Chapitre 4 sur une architecture SoC hybride. Ces contributions sont :

— Implémentation embarquée du modèle BFM : implémentation, logicielle (SW) et/ou matérielle (HW), embarquée du modèle BFM et principalement la prise de décision embarquée à l'aide de MDP. Étude, choix et implémentation de l'algorithme de résolution, d'un MDP, le plus adapté à notre cas d'étude.

— Implémentation d'une méthode de compilation : pour le diagnostic avec les BNs. Implémentation SW/HW de l'apprentissage des paramètres d'un BN d'une part et l'inférence Bayésienne pour le BN−application (voir Chapitre 3) d'autre part.

— Atelier de conception : proposition d'un atelier logiciel permettant d'automatiser la construction des modèles BNs et MDPs, la génération de la méthode de compilation pour les BNs (inférence et apprentissage) et la résolution de MDPs (produire la décision à exécuter). Cette chaîne d'automatisation permet la spécification de la mission à travers une interface utilisateur.

Le choix de faire une implémentation embarquée SW/HW du modèle BFM a été motivée par différents points. Permettre d'avoir un mécanisme de prise de décision pouvant produire une décision avec un temps de réponse court afin de faire face aux événements (obstacle, etc.) ; décharger le processeur dans le cas où il n'y a plus d'espace pour exécuter toutes les applications embarquées. Le modèle BFM peut être, alors, exécuté sur la partie FPGA de la carte embarquée en sollicitant l'implémentation HW du BFM. et ainsi, libérer de la place pour l'exécution des autres applications. Le modèle BFM, qui représente le mission manager, peut alors s'exécuter hors-ligne ou bien en-ligne :

1. Phase hors-ligne : pour cette phase, nous commençons par spécifier les réseaux Bayésiens (sensors, système et applications) du niveau diagnostic du modèle BFM, nous nous concentrons sur le BN-application. L'inférence des valeurs de probabilités est faite à l'aide de la compilation AC du BN. Ensuite, nous décrivons la mission à réaliser par le véhicule autonome à l'aide de MDP. Une fois la structure du BN posée, la description C de haut niveau de la compilation AC est générée afin de permettre le calcul du diagnostic. Pareillement, le code C de l'algorithme choisi pour la résolution du MDP est généré dans le but de produire une décision. Une fois les spécifications respectives du BN et du MDP faites, nous utilisons les outils de synthèse de haut niveau offerts par Vivado HLS de Xilinx pour optimiser la description C proposée du diagnostic et de la décision à l'aide des directives de synthèse qui répondent à différents besoins tels que : le parallélisme, la latence et les contraintes de ressources. La synthèse

de haut niveau, nous permet aussi de définir les interfaces de communications à utiliser pour l'envoi des paramètres du BN et du MDP et ainsi produire la description VHDL du C haut niveau qui sera synthétisé et implémenté sur le FPGA du SoC pour la version HW.

2. Phase En-ligne : cette phase est exécutée pendant la réalisation de la mission par le véhicule autonome. Comme le modèle BFM prend en entrée des données de capteurs, elles peuvent être récupérées et exploitées directement durant la mission ainsi que les métriques des applications embarquées dans le but de calculer le diagnostic et la décision.

La Figure 5.1 montre un aperçu de l'architecture suivie pour l'implémentation du modèle BFM. Le module de décision du modèle BFM est alimenté par le module diagnostic en probabilités. Ces probabilités se reportent sur les valeurs de transitions du MDP décrivant la mission dans le but de produire le plan de mission à exécuter. Le BN du module de diagnostic prend en entrée les données des capteurs et des applications afin d'évaluer la probabilité de l'état de santé du système et ses composants. Ces données de capteurs sont mises à jour à chaque fois qu'un événement (vibration, obstacle, etc.) est détecté. Par conséquent, le module de décision génère un nouveau plan de mission seulement lorsqu'un événement est détecté conduisant à une mise à jour des valeurs de probabilités calculées par le BN diagnostic.

Dans ce chapitre, nous présenterons l'implémentation du modèle BFM proposé en nous concentrons en premier lieu sur la phase hors-ligne. Dans la Section 5.2.1 nous détaillons la génération du BitStream pour l'apprentissage des paramètres du BN et pour l'évaluation de l'état de santé du système à l'aide de la compilation AC. Dans la Section 5.2.2 nous présenterons les différents algorithmes permettant de résoudre un MDP et ainsi produire le plan de mission à exécuter. La Section 5.3.2 montre la génération du BitStream pour la partie prise de décision ainsi que la validation de l'implémentation HW du modèle BFM. Enfin, nous concluons ce chapitre.

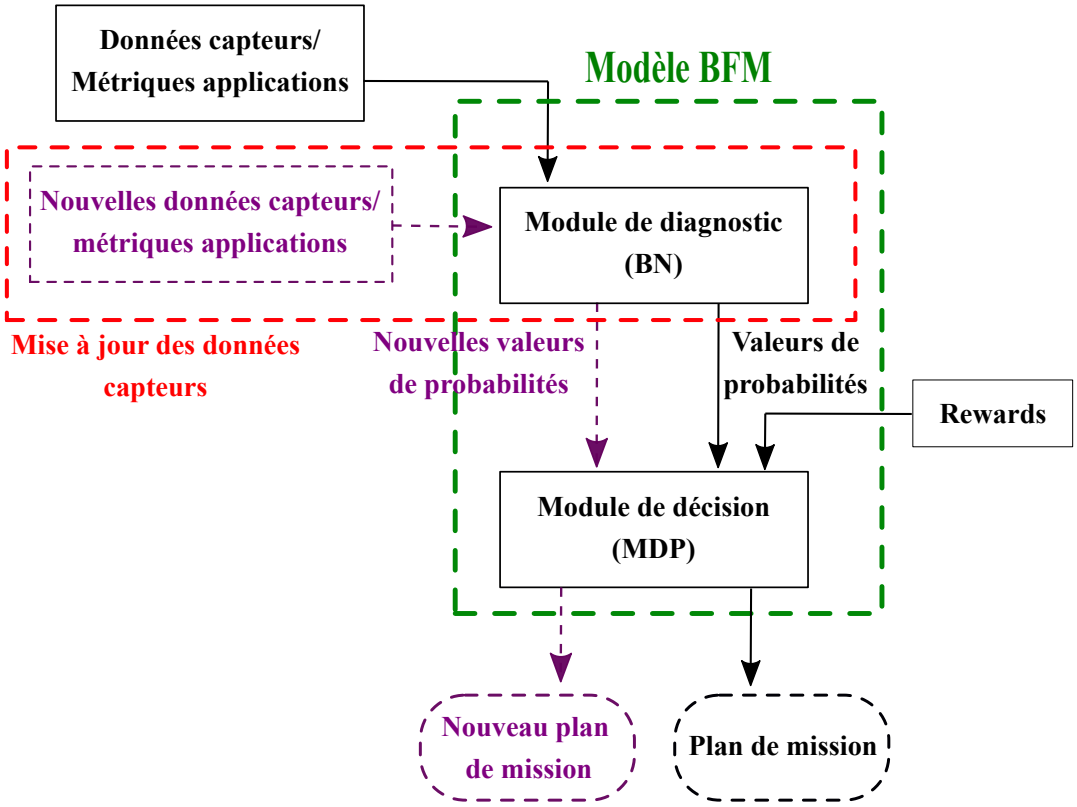


Figure 5.1: *Démarche pour l'implémentation HW/SW pour le modèle BFM*

5.2 Implémentation logicielle embarquée du modèle BFM

5.2.1 Implémentation logicielle du modèle Bayésien

Dans cette section, nous allons présenter les premiers résultats obtenus sur l'apprentissage des paramètres d'un BN. Le principe de l'apprentissage des paramètres de BN à l'aide de l'algorithme EM a été présenté dans le Chapitre 2. L'inférence par arbre de jonction est classiquement utilisée avec EM pour apprendre les valeurs des paramètres du BN dans le cas de données manquantes. L'inconvénient avec l'arbre de jonction est qu'une version embarquable de ce dernier est difficilement envisageable. Pour cela, nous proposons une version de EM embarquable en utilisant l'inférence par circuit arithmétique. Une version de l'inférence par AC a été proposée et validée par les travaux de [Zermani et al., 2017b].

Notre approche consiste donc à remplacer l'inférence par arbre de jonction par l'inférence AC. Cependant, nous exécutons sous Matlab l'algorithme EM avec les deux types d'inférences, arbre de jonction et AC, afin d'évaluer et comparer l'accélération obtenue avec chaque version. Pour nos BNs construits à partir d'une table

Taille base données	100	500	1000	2000	5000
Temps d'exécution EM-JT (s)	0.749	3.947	6.435	8.666	21.28
Temps d'exécution EM-EC (s)	0.028	0.068	0.127	0.152	0.364
Accélération	26	57	50	56	58

Tableau 5.1: *Accélération obtenue avec le BN simple et avec 50% de données manquantes dans le base de données.*

FMEA spécifique, nous n'avons besoin d'apprendre que les probabilités des nœuds contexte d'apparition du type d'erreur. Les autres types de nœuds sont observables à l'aide de capteurs ou bien calculés par l'inférence (voir Figure 5.2). Par conséquent, à l'aide de la version EM-AC embarquable nous avons la possibilité d'inférer que les probabilités des nœuds que l'on souhaite.

Nous utilisons l'exemple du BN-GPS pour apprendre les paramètres des nœuds contexte d'apparitions (nœuds feuilles) seulement à l'aide EM-JT (EM avec arbre de jonction) et EM-AC (EM avec circuit arithmétique). Ensuite nous comparons le facteur d'accélération obtenu. L'apprentissage avec l'algorithme EM s'applique sur une base de données manquantes. EM prend les évidences (observations) depuis la base de données manquantes et calcul la probabilité du nœud souhaité. Nous précisons que nous avons fixé le nombre d'itérations maximum de EM à 10 itérations.

Avant de présenter les résultats d'accélération obtenus avec l'exemple BN-GPS, nous allons évaluer le temps d'exécution ainsi que l'accélération obtenus en appliquant EM avec différentes tailles de base de données sur un BN simple composé de trois nœuds (BN avec 3 nœuds). La Table 5.1 montre les résultats obtenus. Nous constatons que le facteur d'accélération varie de 26 à 58 en fonction de la taille de la base de données. Les temps d'exécution obtenus montre que la version EM-AC est plus rapide que la version EM-JT. Les tables 5.2 et 5.3 quant à elles montrent l'accélération et le taux de précision sur les paramètres calculés en utilisant le BN-GPS. Nous constatons que la version EM-AC est plus rapide que la version EM-JT avec une accélération de plus de 55 fois. L'accélération dans le cas d'une base de données avec 80% de données manquantes est moindre comparativement au cas avec 50% de données manquantes. Cela s'explique par le fait que plus il y a de données manquantes dans la base plus le nombres de valeurs à initialiser avant l'application de EM est important. La précision varie d'un cas à un autre en fonction de la taille de la base de données. Plus le nombres de données est grand plus la valeur des paramètres calculés est précise et s'approche des paramètres initiaux.

=

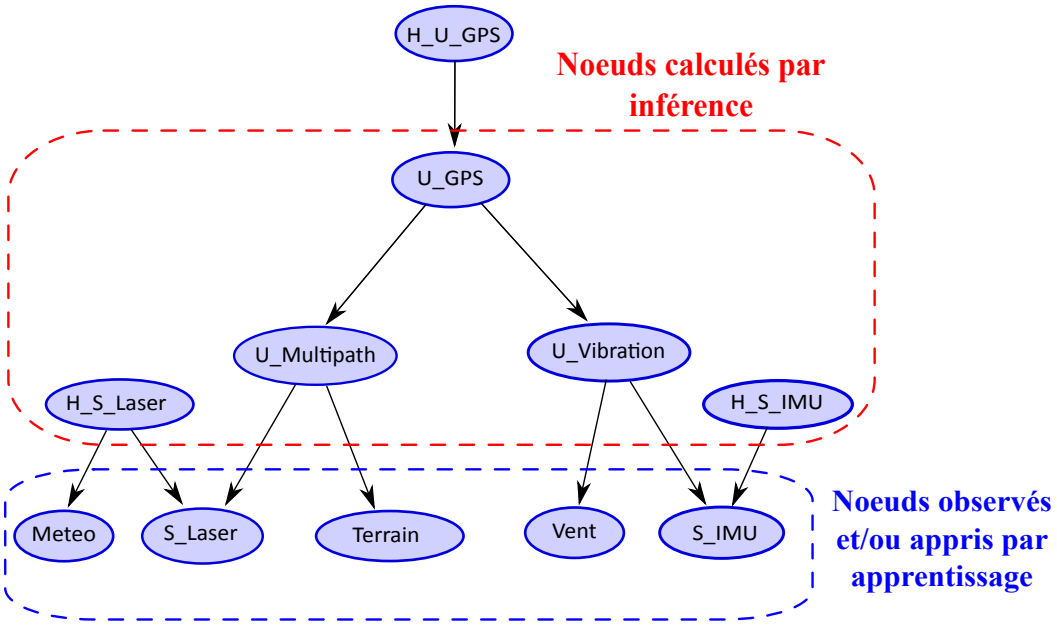


Figure 5.2: *BN du capteur GPS.*

Taille base données	500	2000	5000
Précision EM-JT	4%	2%	2%
Précision EM-AC	4%	4%	2%
Accélération	42 fois	45 fois	44 fois

Tableau 5.2: *Accélération et précision obtenus avec 80% de données manquantes dans la base de données.*

Taille base données	500	2000	5000
Précision EM-JT	3%	1%	1%
Précision EM-AC	6%	3%	2%
Accélération	55 fois	58 fois	58 fois

Tableau 5.3: *Accélération et précision obtenus avec 50% de données manquantes dans la base de données.*

Bilan

Les premiers résultats obtenus concernant l'apprentissage de paramètres de BN en termes d'accélération en SW (exécution réalisée avec Matlab) montrent que :

- La version EM-AC est plus performante en termes d'accélération et de temps d'exécution en comparaison avec la version classique EM-JT.
- La version embarquable de l'inférence AC a été proposée et prouvée dans les travaux de [Zermani et al., 2017b], par conséquent, l'apprentissage embarqué des paramètres d'un BN est aussi embarquable avec l'utilisation de la version AC de l'algorithme EM.
- EM-AC permet de cibler l'apprentissage sur les nœuds utiles que l'utilisateur souhaite calculer et par conséquent réduire le temps d'exécution de l'algorithme et gagner en accélération.

Ces résultats ont aboutis avec un article publié dans la conférence MECO'2017 (Mediterranean Conference on Embedded Computing) [Hireche et al., 2017].

5.2.2 Méthodes de résolution du modèle MDP

Comme décrit dans le Chapitre 4, un MDP est un modèle stochastique, probabiliste qui permet de décrire une mission réalisée par un véhicule autonome. Le but du MDP est de produire la décision à appliquer par le véhicule autonome, cette décision consiste à trouver l'action 'A' à exécuter à partir de chaque l'état 'S' de la mission. Cette décision est appelée politique π et représente l'ensemble des actions à exécuter par le véhicule autonome.

Bien que la définition de la politique π puisse être arbitraire, la plupart des approches limitent le choix des actions en considérant l'action dépendante uniquement de l'état. Par exemple, lorsque nous sommes dans l'état S(i), le choix de l'action A(i) dépend uniquement de l'état S(i). Résoudre un modèle MDP consiste alors à choisir les actions optimales qui maximiseront ou bien minimiseront un critère de coût (récompense). Classiquement, plusieurs critères de coût sont utilisés.

Les critères typiques de récompenses comprennent les coûts instantanés ou moyens escomptés et non escomptés, sur des horizons temporels finis ou infinis.

Il existe deux principaux algorithmes permettant de résoudre un MDP. Ces deux algorithmes sont : '***Policy Iteration***' et '***Value Iteration***'. Les deux algorithmes reposent sur l'équation de '**Bellman**' sous une forme récursive exprimant la récompense à partir d'un état sous la forme d'une récompense instantanée plus une récompense résiduelle. L'eq 5.1 définit l'équation de Bellman à partir du coût non escompté sur un horizon fini :

$$\begin{aligned} V_\pi(S(k)) &= R(S(k), A(k)) \\ &+ \sum_{j=0}^{N} (P(S(j), S(k), A(k))V(S(j))) \end{aligned} \tag{5.1}$$

Où $P(S(j), S(k), A(k))$ représente la probabilité de transition permettant le passage de l'état S(k) vers l'état S(j) avec l'action A(k), $R(S(k), A(k))$ correspond au coût généré par l'action A(k) en étant dans l'état S(k) et V(S(j)) est le coût résiduel à effectuer après avoir exécuté l'action A(k). L'algorithme '***Value Iteration***' construira la politique optimale tout en calculant récursivement les récompenses générées à partir de chaque état. À chaque étape, l'action la plus appropriée est choisie afin de maximiser la récompense exprimée dans l'eq 5.1 comme suit :

$$\begin{aligned} V^*(S(k)) &= max_A R(S(k), A(k)) \\ &+ \sum_{j=0}^{N} \left(P(S(j), S(k), A(k)) V^*(S(j))\right), \end{aligned} \tag{5.2}$$

Où V^* représente la récompense maximisée.

L'algorithme '***Policy Iteration***' quant à lui calcule en premier une politique initiale, ensuite résout l'équation de '**Bellman**'.

$$\begin{aligned} V_\pi(S(k)) &= R(S(k), A(k)) \\ &+ \sum_{j=0}^{N} \left(P(S(j), S(k), A(k)) V_\pi(S(j))\right), \end{aligned} \tag{5.3}$$

Finalement, l'algorithme calcule la politique optimale pour chaque état comme suit :

$$\begin{aligned} \pi^*(S(k)) &= arg\_max_A \left(R(S(k), A(k))\right. \\ &+ \left.\sum_{j=0}^{N} \left(P(S(j), S(k), A(k)) V_\pi(S(j))\right)\right) \end{aligned} \tag{5.4}$$

.

La Figure 5.3 montre dans le détail le fonctionnement de l'algorithme '*Policy Iteration*'. L'algorithme *Policy Iteration* s'appuie sur deux phases :

1. **Phase d'initialisation** : consiste à initialiser la politique de départ en prenant comme entrée les matrices de transitions P, la matrice des récompenses (matrice de coût) R et le facteur discount γ. Cette phase d'initialisation utilise l'équation de Bellman pour calculer la politique de départ π_0 et les valeurs de fonctions V (value function V_0). Les valeurs de fonctions V représentent l'espérance de coût futur attendu à partir de l'état S en choisissant l'action A.
2. **Calcul de la prochaine politique** : consiste à itérer récursivement sur la politique π_0 en utilisant l'équation de Bellman. La Figure 5.4(b) illustre les

différentes étapes de la boucle d'exécution qui consiste à générer une politique optimale. Une nouvelle matrice de transition et une matrice de récompenses sont alors construites à partir de la politique trouvée à l'itération précédente. Ces nouvelles matrices vont servir à calculer les valeurs de fonctions V associées à la politique précédente. Ensuite une nouvelle politique est calculée en appliquant l'équation de Bellman sur les nouvelles valeurs de fonctions V.

Ces deux phases sont répétées jusqu'à convergence de la politique entre deux itérations successives.

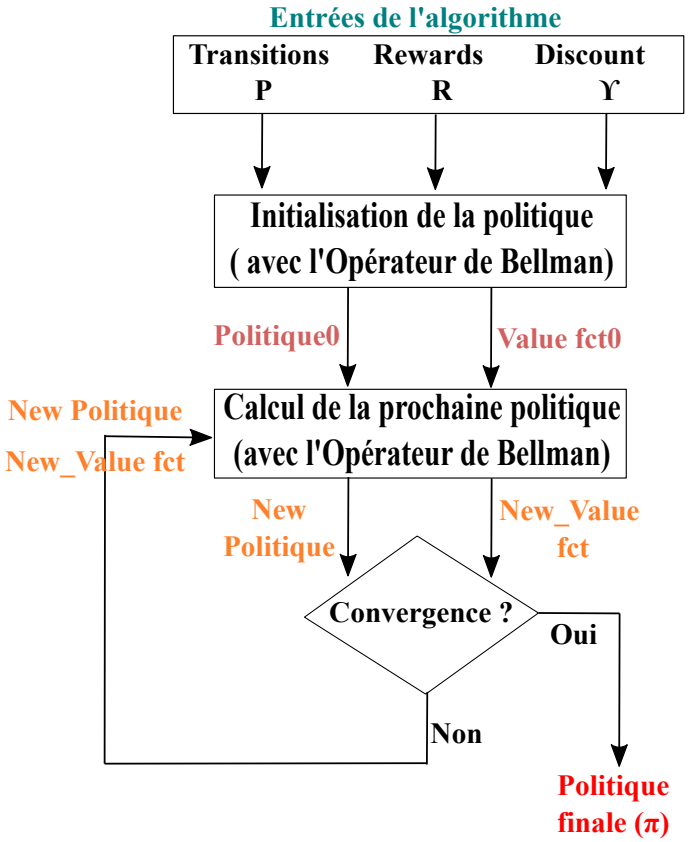
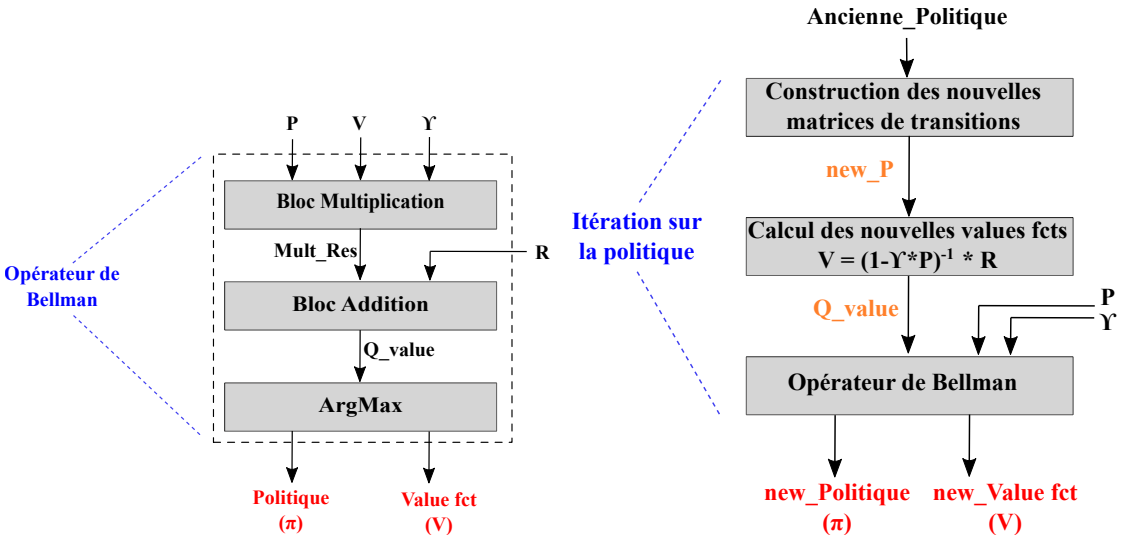


Figure 5.3: *Étapes de l'algorithme 'Policy Iteration'.*

5.2.3 Évaluation des performances des méthodes de résolutions de MDP

Les deux algorithmes *Policy Iteration* et *Value Iteration* s'appuient sur la résolution de l'équation de **Bellman**. La différence réside dans le fait que *Policy Iteration* itère sur une politique initiale et que *Value Iteration* itère sur les valeurs de fonctions escomptées générées à chaque itération et à chaque état.



((a)) *Opérations effectuées par l'opérateur de Bellman*

((b)) *Opérations effectuées par l'étape d'itération sur la politique*

Figure 5.4: *Fonctionnement de l'algorithme Policy Iteration (PI).*

Dans cette section, nous allons étudier ces deux algorithmes et évaluer leurs performances afin de déterminer l'algorithme le plus efficace à utiliser pour notre cas d'étude qui consiste à générer des plans pour la mission de tracking. Afin d'évaluer les performances des deux algorithmes, nous allons prendre un exemple simple et pédagogique. L'exemple est illustré par la Figure 5.5 qui représente une grille sur laquelle un robot terrestre doit se déplacer à partir de la case de départ dans le but d'atteindre la case objectif représentée par la pierre bleu (case en haut à droite) [Russell and Norvig, 2016]. Si le robot atteint l'objectif, un coût de (+1) sera généré. Cette grille possède deux obstacles de nature différentes, le premier obstacle est un mur (case (2,2)) et le deuxième obstacle est un feu (case (2,4)) qui engendre un coût de (-1) si le robot passe par la case feu. La formulation de cet exemple de grille à l'aide d'un modèle MDP est la suivante :

— **Ensemble des états S** : représente les différentes cases de la grille.

— **Ensemble des actions A** : le robot possède trois actions de commandes, à savoir : "aller en face", "aller à droite" et "aller à gauche".

— **Fonction de transitions** : chaque action a une probabilité d'être choisie. La distribution de probabilité pour cet exemple est comme suit :
Probabilité de 80% pour l'action choisie et 10% si une action n'est pas choisie. Dans le cas ou aucune action n'est choisie, le robot reste dans l'état (case) courant.

— **Fonction de coût (récompense)** : atteindre la case d'arrivée génère un coût

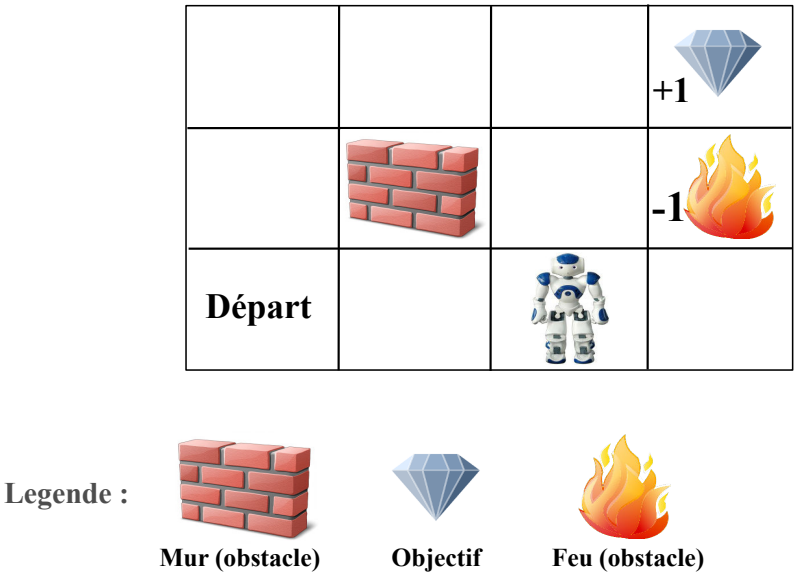


Figure 5.5: *Exemple de mission de robot terrestre.*

(+1), être dans la case feu génère un coût de (-1).

Dans ce qui suit, nous utilisons l'exemple du robot se déplaçant sur une grille (Figure 5.5) pour évaluer les performances en termes de temps d'exécution et de latence pour les deux algorithmes 'Policy Iteration' (PI) et 'Value Iteration' (VI).

1. **Temps d'exécution**
   Nous avons évalué le temps d'exécution nécessaire pour le calcul de la politique optimale π en fonction de la taille de la grille en utilisant la MDPToolbox fourni par MATLAB d'une part et en utilisant notre version en code C des deux algorithmes d'autre part. Les deux méthodes de résolution, PI et VI, sont itératives. Pour cela, nous allons nous intéresser dans un premier temps au nombre d'itérations nécessaires pour chacune des méthodes afin de produire la politique π.
   La Table 5.4 montre le nombre d'itérations nécessaires pour la résolution d'un problème MDP avec les deux algorithmes PI et VI. Nous remarquons facilement que l'algorithme VI a besoin de faire un grand nombre d'itérations comparativement à l'algorithme PI afin de converger vers une politique optimale.
   La raison pour laquelle l'algorithme VI prend plus d'itérations que l'algorithme PI est la suivante :
   — L'algorithme VI repose sur l'itération sur les valeurs de fonctions 'V' des récompenses futures.
   — Entre chaque deux itérations successives de VI, les valeurs de fonctions 'V' sont comparées afin de déterminer si l'algorithme a convergé selon un critère d'arrêt.

Taille grille (SxS)	Nombre d'itération de PI	Nombre d'itération de VI
16x16	5 itérations	21 itérations
25x25	5 itérations	22 itérations
36x36	5 itérations	24 itérations
49x49	5 itérations	26 itérations
64x64	6 itérations	29 itérations
81x81	7 itérations	32 itérations
100x100	9 itérations	34 itérations

Tableau 5.4: *Nombre d'itérations de PI et VI selon le nombre d'états de la grille.*

— Si la valeur absolue de V(i)-V(i-1) (ième itération) est inférieure à la valeur du critère d'arrêt, fixée comme paramètre d'entrée de VI, alors l'algorithme VI a trouvé la politique optimale.

Par conséquent, l'algorithme VI effectue un grand nombre d'itérations par rapport à l'algorithme PI. Le nombre d'itérations de VI est lié à la valeur du critère d'arrêt souhaitée. Plus cette valeur est petite plus le nombre d'itération est grand et inversement.

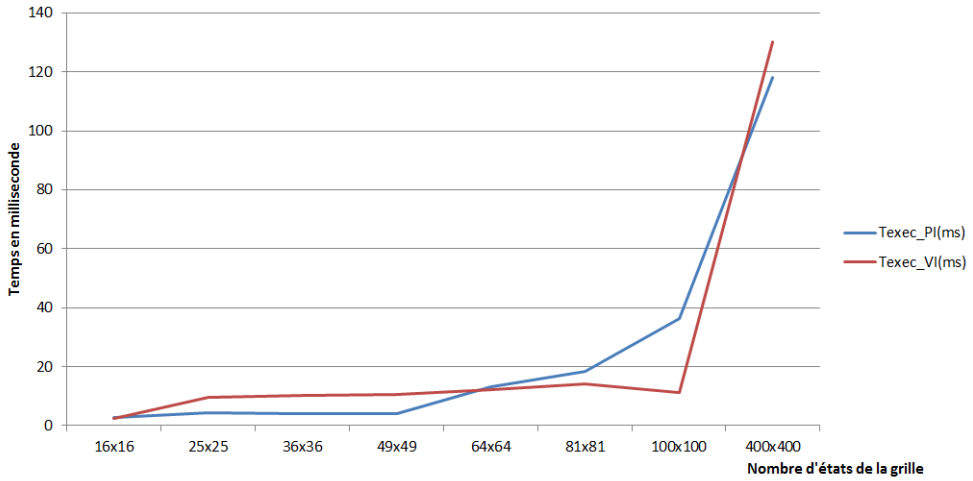


Figure 5.6: *Temps d'exécution de PI et VI*

La Figure 5.6 représente l'évolution du temps d'exécution des algorithmes PI et VI en fonction du nombre d'états du MDP décrivant l'exemple de la grille. Nous constatons deux parties dans ce graphique. Avec un nombre d'états du MDP inférieur à 64x64, l'algorithme VI prend plus de temps pour s'exécuter que l'algorithme PI. Lorsque le nombre d'états du MDP dépasse 64 états, la courbe s'inverse avec l'algorithme PI qui prend plus de temps pour résoudre le MDP. Ces résultats peuvent s'expliquer par le fait que l'algorithme PI utilise l'inversion de matrices à chaque itération pour le calcul des valeurs de

Nombre d'états (S)	Latence de PI (cycles)	Latence de VI (cycles)
16	324.290	794.760
25	784.442	1.923.306
36	1.618.550	3.969.240
49	2.989.634	7.332.522
64	5.090.354	12.485.832
81	8.143.010	19.974.570
100	12.399.542	30.416.856
225	62.888.422	153.656.106
400	198.601.447	485.266.056
625	517.510.247	1.234.714.206

Tableau 5.5: *Latence (nombre de cycles) de PI et VI selon le nombre d'états de la grille.*

fonctions 'V' (Figure 5.4(b)). Par conséquent, lorsque le nombre d'états est très grand, l'opération d'inversion de matrices entre autre prend plus de temps pour s'effectuer comparativement au cas où le nombre d'états est relativement petit. C'est pour cette raison que nous observons le changement de tendance du temps d'exécution sur le graphique de la Figure 5.6.

2. **Latence**
Continuons avec l'exemple de la grille afin de comparer la latence entre les deux algorithmes PI et VI. Pour cette expérimentation, nous avons utilisé notre code C pour PI et VI que nous avons synthétisé à l'aide de l'outil Vivado HLS (High Level Synthesis) dans le but d'évaluer la latence. Nous avons défini plusieurs tailles pour une grille allant de 16 cases à 625 cases. La Table 5.5 regroupe les valeurs de latence obtenues pour chacun de PI et de VI en fonction de la taille de la grille. La taille de la grille correspond au nombre d'états S du MDP. La Figure 5.7 montre l'évolution de la latence obtenue. Nous pouvons déduire que les deux algorithmes suivent une tendance plutôt linéaire. Plus la taille du problème à résoudre est grande plus la résolution prend un nombre important de cycles (latence) pour générer la solution optimale (politique π). Cependant, nous constatons que la latence de l'algorithme PI est moindre comparativement à celle de VI pour le même nombre d'états.

Analyse et synthèse

Ces résultats présentent une étude comparative entre l'algorithme PI et l'algorithme VI en termes de temps d'exécution et de latence. Vu que nous sommes dans un contexte où le véhicule autonome doit prendre une décision autonome face à l'apparition d'un événement, le temps de calcul de cette décision et donc le temps de réponse est important. Les résultats obtenus plus haut montrent que l'algorithme *Policy Iteration* est le plus approprié pour générer une décision le plus rapidement possible.

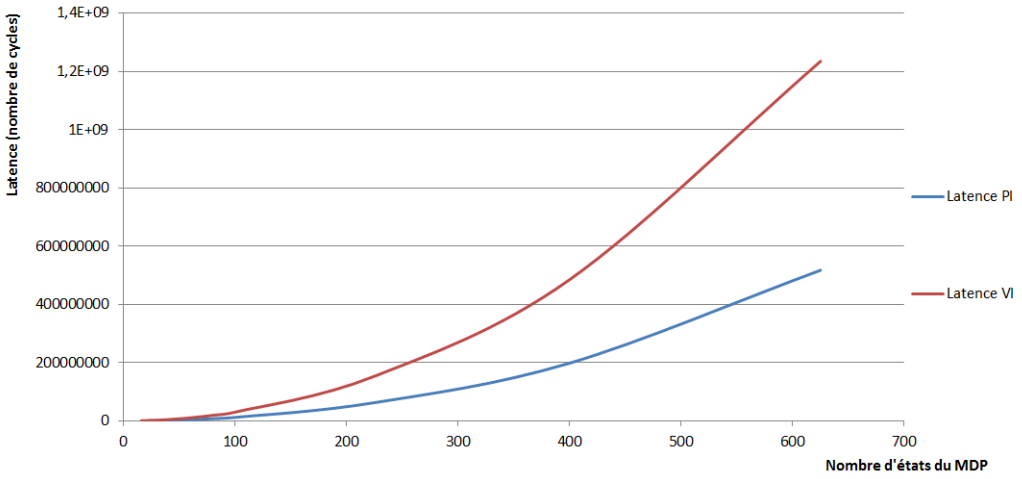


Figure 5.7: *Évolution de la latence pour PI et VI en fonction du nombre d'états de la grille.*

5.3 Implémentation matérielle du modèle BFM

5.3.1 Présentation du framework

Avant de présenter les résultats de l'implémentation matérielle du modèle BFM, nous allons d'abord décrire et présenter le framework proposé qui s'appuie sur le modèle BFM. La Figure 5.8 montre la démarche suivie. Le but de ce framework est d'avoir un outil permettant de faciliter à un utilisateur non expert la description d'une mission d'un système autonome d'une part et de contrôler cette mission en résolvant le modèle BFM adapté à la mission décrite.

Dans les chapitres précédents, (Chapitre 3 et Chapitre 4), nous avons bien vu que la construction des modèle BNs et MDPs n'était pas une tâche évidente. Afin de faciliter cette tâche, nous proposons les étapes suivantes pour le framework :

— **Spécification de la mission** : cette étape se doit d'être la plus intuitive et accessible pour tout utilisateur (expert ou non expert de la mission). Deux interfaces seront utilisées pour indiquer tous les éléments importants permettant de décrire au mieux la mission et ainsi construire un modèle BFM le plus complet possible. Étant donné que notre modèle BFM est constitué de deux niveaux, diagnostic et décision, nous devons récolter le plus d'informations possibles pour construire les BNs et les MDPs.
La première interface concerne la partie diagnostic. L'interface sera sous forme de table FMEA dans laquelle l'utilisateur rentrera, pour un système donné, les informations sur les types d'erreurs, leurs monitoring possible, le contexte d'apparition du type d'erreur et la solution envisagée pour corriger l'erreur observée.

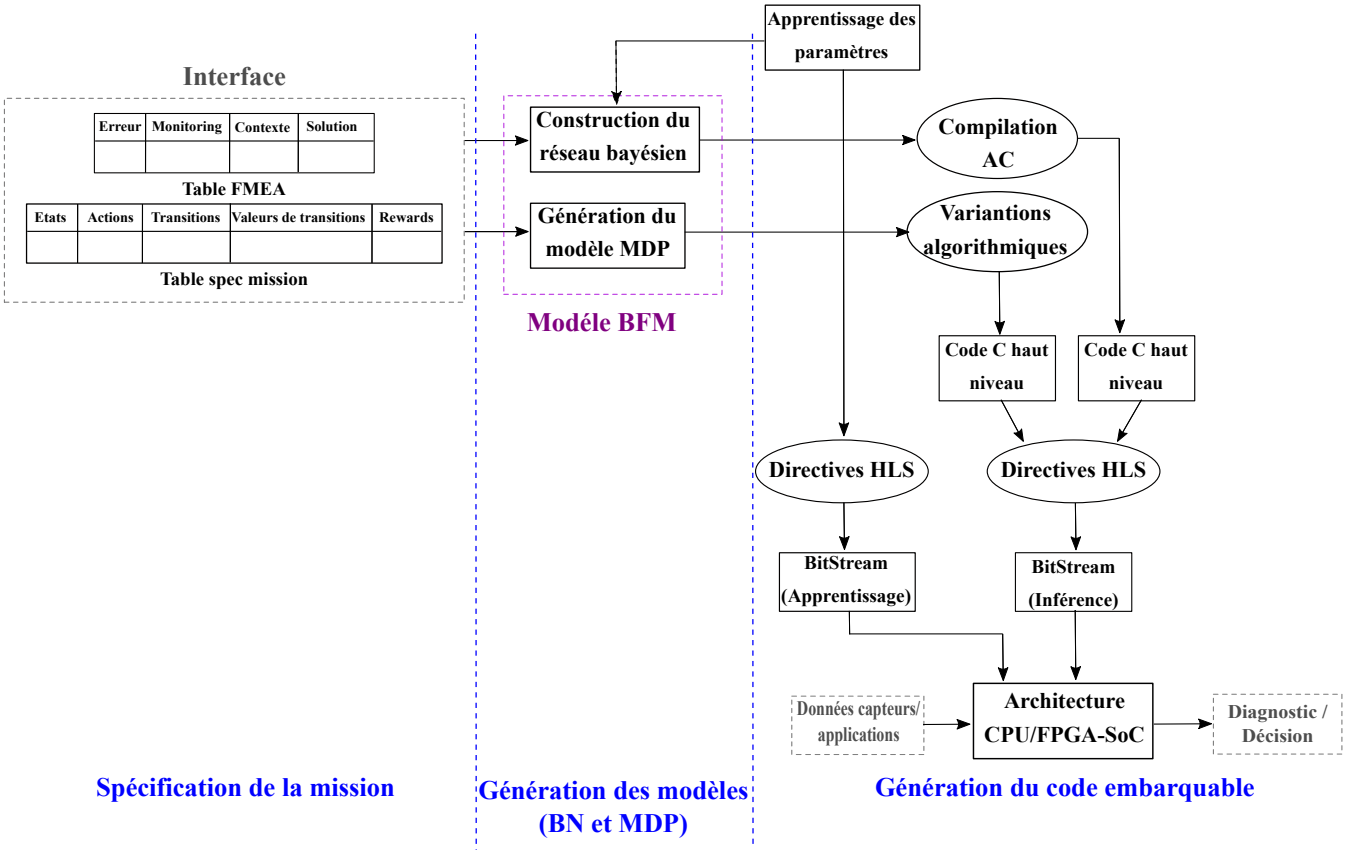


Figure 5.8: *Framework proposé.*

La seconde interface est associée à la partie décision du modèle BFM. Cela consiste en une table de spécification de la mission. Cette table contient les différents états que la mission peut avoir, les actions permettant de mener à bien la mission, les transitions représentées par un tuple *(état(i), action(a), état(i+1))*, la valeur de probabilité portée par la transition et enfin le reward (récompense) associé à chaque action.

— **Génération des modèles probabilistes** : Une fois la table FMEA et la table de spécification de la mission remplies, les modèles associés sont alors générés. Le BN correspondant au composant à monitorer est alors construit à partir de la table FMEA renseignée par l'utilisateur (voir Section 3.2.4). Le MDP décrivant la mission est construit à partir de la table de spécification de la mission de la façon suivante : graphiquement, les états seront représentés par des nœuds états, les actions par des carrés et les transitions par des arcs qui relit deux états avec une action. Les valeurs de probabilités seront portées par les arcs de transitions. La table de spécification de la mission nous aidera tout particulièrement dans la construction des différentes matrices de transitions et de rewards. Pour cela, la taille (SxS) des matrices de transitions est indiquée

par le nombre d'états (S) renseigné par la table de spécification. Le nombre de matrices de transitions à générer est égale au nombre d'actions indiqué dans la table de spécification. Les valeurs de transitions seront reportées sur la case de la matrice indiquée par le tuple transition *(état(i), action(a), état(i+1))*. Quant à la matrice rewards, sa taille (SxA) est définie par le nombre d'états (S) et le nombre d'actions (A) indiqués par la table de spécification de mission. Les valeurs de récompenses sont reportées sur la matrice rewards aux emplacements indiqués par le couple *(état(i), action(a))*.

— **Génération du code embarquable** : Une fois les modèles (BN et MDP) construits correctement, vient l'étape de résolution du modèle BFM.
Pour le diagnostic, le modèle Bayésien va être décrit sous forme d'arbre AC pour lequel le code C de haut niveau est généré. Ensuite, le code va passer par la synthèse de haut niveau (HLS) afin d'optimiser le code à l'aide de directives HLS. Après optimisation, le BitStream associé à l'AC est généré afin d'être embarqué sur l'architecture SoC-FPGA. Le fait de proposer une implémentation matérielle pour l'inférence Bayésienne à l'aide de l'AC permet d'accélérer les calculs et d'évaluer la QoS des applications embarquées et l'état de santé des capteurs en ligne.
Cependant, dans le cas où les données de capteurs sont manquantes, nous avons la possibilité d'apprendre les paramètres du BN, avec l'algorithme EM, pour lequel une implémentation matérielle sera proposée.
Pour la partie décision du modèle BFM, l'implémentation de l'algorithme de résolution d'un MDP est générée (code C) avec les entrées spécifiques à la description de la mission à gérer. Ces entrées sont explicitement, les matrices de transitions contenant les probabilités produites par le BN diagnostic et la matrice de rewards. De la même manière que pour les BNs, l'algorithme de résolution du MDP est optimisé à l'aide des directives HLS pour ensuite générer le BitStream pour configurer le FPGA.

5.3.2 Validation de la version matérielle du modèle BFM

Génération du code C pour la synthèse de haut niveau

La génération du code C se base sur la décomposition fonctionnelle de l'algorithme de résolution choisi. Dans notre cas, c'est l'algorithme *Policy Iteration* qui est choisi, nous respectons la décomposition fonctionnelle décrite dans la Figure 5.3.

Une fois le code C généré, nous optimisons le code pour qu'il soit plus efficace, ces optimisations se basent sur les points suivants :

1. La représentation des données :
Les deux méthodes (PI et VI) de résolution de modèle MDP prennent toutes les deux les même types de données en entrées, à savoir :

— Les matrices de transitions P : qui contiennent les probabilités de passage d'un état S(i) à un état S(j) avec une même action A. Ces valeurs de probabilités sont inférieures ou égales à 1. Donc, les matrices de transitions peuvent être représentées par des flottants pour un maximum de précision ou bien utiliser une représentation en virgule fixe (ou à l'aide d'entiers) en précisant la taille globale de la données ainsi que la taille de la partie réelle.

— Matrice de récompenses R : contient les valeurs de récompenses (coûts) immédiates générées par une action A(i) en ayant été choisie à l'état S(i). Ces valeurs de récompenses peuvent être représentées par des flottants ou des entiers.

— Facteur γ : qui est une valeur flottante.

Le choix de représenter les données en flottants est porté par le fait que l'algorithme PI peut s'exécuter soit en SW ou bien en HW pour décharger le CPU. Aussi, comme nous le verrons dans la suite de ce chapitre, PI ne consomme pas énormément de ressources (LUT, DSP, etc.).

2. Organisation des données en mémoire :
Le nombre et la taille des matrices de transitions peuvent être importants et augmentent respectivement avec le nombre d'actions et le nombre d'états du MDP. La matrice de coûts augmente aussi avec le nombre d'actions et d'états du modèle MDP. Pour cela, l'organisation des données en mémoire est importante. Nous utilisons des directives de synthèse permettant un stockage externe des données avec l'utilisation des AXI Stream ou bien un stockage en interne en utilisant des BRAM.

— Avec AXI Stream : les données des paramètres sont envoyés via le Stream depuis le processeur vers l'accélérateur. Les données sont stockées en mémoire interne dans des registres ou bien des BRAM et lu par l'accélérateur. Une fois le traitement terminé, les résultats sont envoyés via le Stream au processeur.

— Avec mémoire partagée : les données sont stockées dans une BRAM. La BRAM est connectée d'un côté à l'accélérateur via un port GP et d'un autre côté au processeur via un contrôleur BRAM afin d'accéder aux adresses des données. Pour ce faire, il suffit de rajouter les directives suivantes dans la fonction globale.

```
#include <hls_stream.h>
#include <ap_axi_sdata.h>
typedef ap_axis<32,2,5,6> intSdCH;  //  les paramètres du stream

float NomF(hls::stream<intSdCH>&inStream,
hls::stream<intSdCH>&outStream)

#pragma HLS INTERFACE axis port=inStream   // directive axi stream
#pragma HLS INTERFACE axis port=outStream

// L'envoi des données et la récupération via le Stream
for (int i=0; i<size;c++)
{
#pragma HLS PIPELINE
 #pragma HLS UNROLL
intSdCH InT1= inStream.read();       // lire  les données
union {    unsigned int ival; float oval; } convertert1;
convertert1.ival = InT1.data;          //convertir du uint à float
 T1[c] = convertert1.oval;    // stockage des données

// Appel de la fonction  (resultat Res)

// L'envoi du résultat   via le Stream
union {    unsigned int oval; float ival; } converter1;
     converter1.ival=Res;
     valOut1.data=converter1.oval;
// pour indiquer la fin
     valOut1.last = 0;
     valOut1.strb = -1;
     valOut1.keep = 15;
     valOut1.user = 0;
     valOut1.id = 0;
     valOut1.dest = 0;
     outStream.write(valOut1);   // écriture du résultat
```

```
#pragma HLS INTERFACE bram port=T1 //  variable T1 en  BRAM
#pragma HLS RESOURCE variable=T1 core=RAM_1P_BRAM
```

Validation de l'implémentation matérielle du MDP

Après la génération du code C de l'algorithme PI, nous utilisons l'outil Vivado de Xilinx pour l'implémentation HW sur la carte ZedBoard. Avant l'exécution sur la carte, nous devons effectuer les étapes suivantes :

1. Synthèse de haut niveau et génération du RTL :
   Pour cette étape, nous utilisons l'outil Vivado HLS pour l'optimisation, du code généré, en utilisant les différentes directives et la préparation des interfaces (AXI Stream, BRAM, etc.). L'outil HLS nous permet aussi de synthétiser notre code dans le but d'obtenir une estimation de la latence et des ressources

utilisées en termes de DSP, FF, BRAM et LUT. Après que la synthèse soit terminée, nous générons le RTL.

2. Génération du BitStream :
   Après la génération du RTL, l'outil Vivado est utilisé pour construire l'architecture qui contiendra l'IP HLS du *solver* et les différents blocs HW afin d'assurer la communication entre le CPU et l'IP. Les blocs utilisés, dans notre cas, sont : des DMA, Timer, des BRAM et contrôleur de BRAM. Une fois l'architecture validée, le BitStream peut être généré.
3. Exécution sur la carte ZedBoard :
   Maintenant que le BitStream est généré, nous allons l'exporter vers l'outil Xilinx SDK afin d'exécuter la version SW et la version HW du *solver* PI et évaluer l'accélération obtenue.

La Figure 5.9 résume les étapes à suivre en utilisant la chaîne d'outils Xilinx afin d'exécuter nos approches sur la carte.

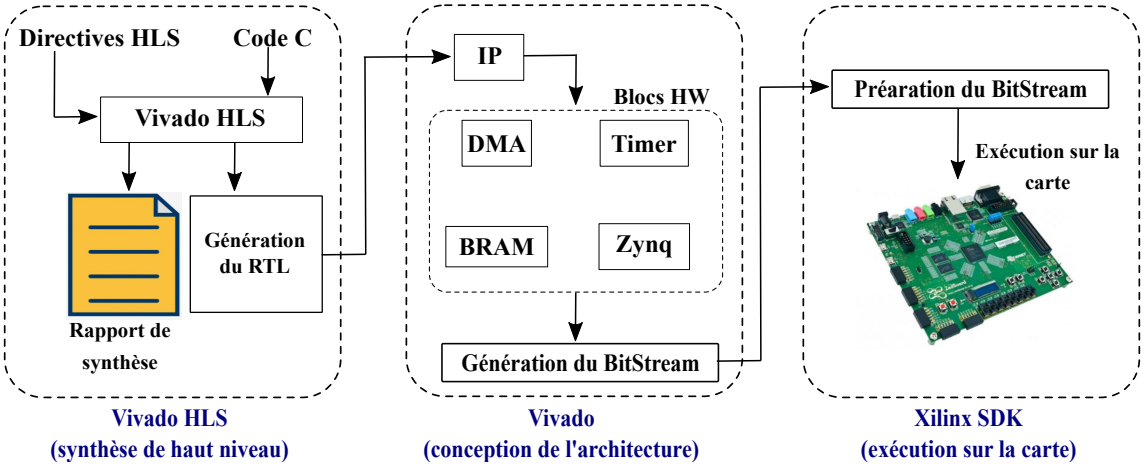


Figure 5.9: *Étapes de l'implémentation sur la carte.*

Dans ce qui suit, nous reprenons la mission de tracking présentée dans le Chapitre 4. La mission de tracking a été décrite à l'aide du modèle BFM avec les deux versions proposées : la version monolithique et la version concurrente. Dans cette section, nous allons comparer ces deux versions en termes de latence et consommation de ressources.

Version monolithique

La mission de tracking est représentée dans son intégralité à l'aide de la version monolithique du modèle BFM. Ce qui signifie que la mission est décrite par un seul MDP global (voir Section 4.3.1). La Figure 5.10 montre une vue globale de la décomposition fonctionnelle mise en place pour l'implémentation de notre solution.

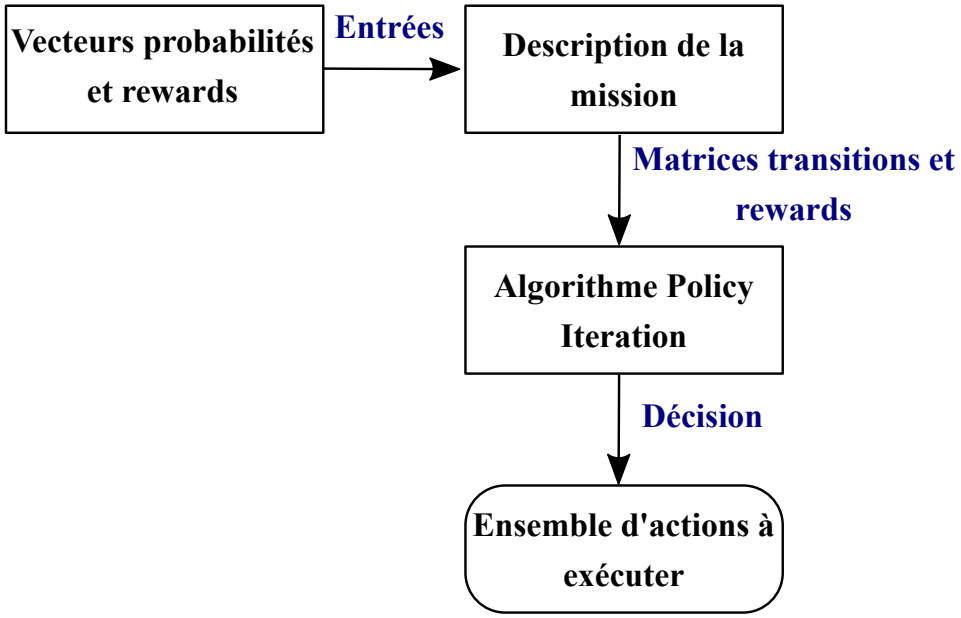


Figure 5.10: *Décomposition fonctionnelle de l'implémentation de la mission*

Notre approche prend en entrées deux vecteurs, un vecteur de probabilités et un vecteur de rewards. Ces vecteurs sont en entrées de la fonction de spécification de la mission. Cette dernière utilise les valeurs des vecteurs de probabilités et de rewards afin de générer les matrices de transitions et la matrice de rewards associées à la mission. Une fois les différentes matrices générées, elles sont injectées, en plus du facteur d'atténuation (discount), à l'algorithme PI qui produira la liste des actions à exécuter selon l'événement observé. L'observation ou non d'un événement est introduite par les valeurs de transitions.

1. Résultats de la synthèse haut niveau (sans pragma) :
   Nous commençons par présenter les résultats de synthèse obtenus avec l'outil Vivado HLS sur le code C de l'algorithme PI sans l'application de directives HLS. Le MDP global de la mission de tracking est composé de 20 états et de 21 actions. La taille des vecteurs de probabilités et de rewards est respectivement 13 et 21. Nous pouvons déduire assez facilement que la fonction de spécification de la mission, de la Figure 5.10, générera 21 matrices de transitions de taille 20x20 (états x états) et un matrice rewards de taille 20x21 (états x actions).
   La Table 5.6 contient les résultats HLS de la version monolithique du modèle BFM appliquée à la mission de tracking. Nous constatons que, malgré la taille de la mission à résoudre (20 états et 21 actions), le taux de ressources utilisé est raisonnable. Notons que seulement 23% de LUT et 15% de BRAM sont utilisées. Cependant, nous remarquons que la latence en nombre de cycles nécessaire pour la résolution de la mission est élevée.

	Latence (cycles)	BRAM	DSP	FF	LUT
Sans pragma	16.084.780	15%	14%	8%	23%

Tableau 5.6: *Version monolithique : latence et ressources utilisées (sans optimisation).*

Comme présenté dans la Section 5.2.2, l'algorithme PI s'appuie sur la résolution de l'équation de Bellman. Nous proposons de regarder de plus près la latence (nombre de cycles) consommée par les différentes fonctions constituant l'algorithme PI (équation de Bellman, etc.). La Table 5.7 résume la latence obtenue avec vivado HLS. L'équation de Bellman utilise deux fonctions qui sont le 'produit de matrices' et 'argmax'. Aussi, la fonction de calcul des valeurs de fonctions 'V' (récompenses futures à partir d'un état S) fait appel à deux fonctions qui sont 'inversion de matrices' et 'construction des matrices internes'. La construction des matrices internes correspond à la construction d'une matrice de transition et une matrice reward associée à la politique π trouvée à l'itération précédente de PI. Nous constatons, à partir des résultats de la Table 5.7, que les fonctions qui consomment le plus en latence sont le 'produit de matrices' et 'inversion de matrices'.

Algorithme	Fonctions principales	Sous Fonctions	Latence (cycles)
Policy Iteration	Equation de Bellman	Produit de matrices	126.883
		Argmax	5.561
	Calcul des valeurs de fonctions (V)	Inversion de matrices	216.584
		Constructions des matrices internes	27.343

Tableau 5.7: *Profiling des différentes fonctions de l'algorithme PI.*

2. Optimisation et résultats de la synthèse haut niveau :
   Le but des optimisations apportées est de réduire la latence de PI. Pour cela, nous utilisons les directives offertes par vivado HLS. Les résultats du profiling ont montré que le 'produit' et 'inversion' de matrices sont les fonctions les plus coûteuses en latence. Pour cela, nous avons fait le choix de mettre les directives sur ces fonctions (produit et inversion) afin de réduire la latence de l'algorithme PI. Les directives utilisées sont 'PIPELINE', 'INLINE' La Figure 5.11 montre les résultats en terme de consommation de ressources obtenus.

   Les résultats montrent qu'avec l'utilisation de la directive 'INLINE' sur la fonction de résolution global de la mission aucune amélioration n'est apportée. Cependant, lorsque nous appliquons la directive 'PIPELINE', nous constatons une baisse au niveau de la latence et une hausse du taux de ressources utilisé (Table 5.8). L'application de la directive 'PIPELINE' sur uniquement la fonction 'produit de matrice' montre une diminution de la latence et 29% de

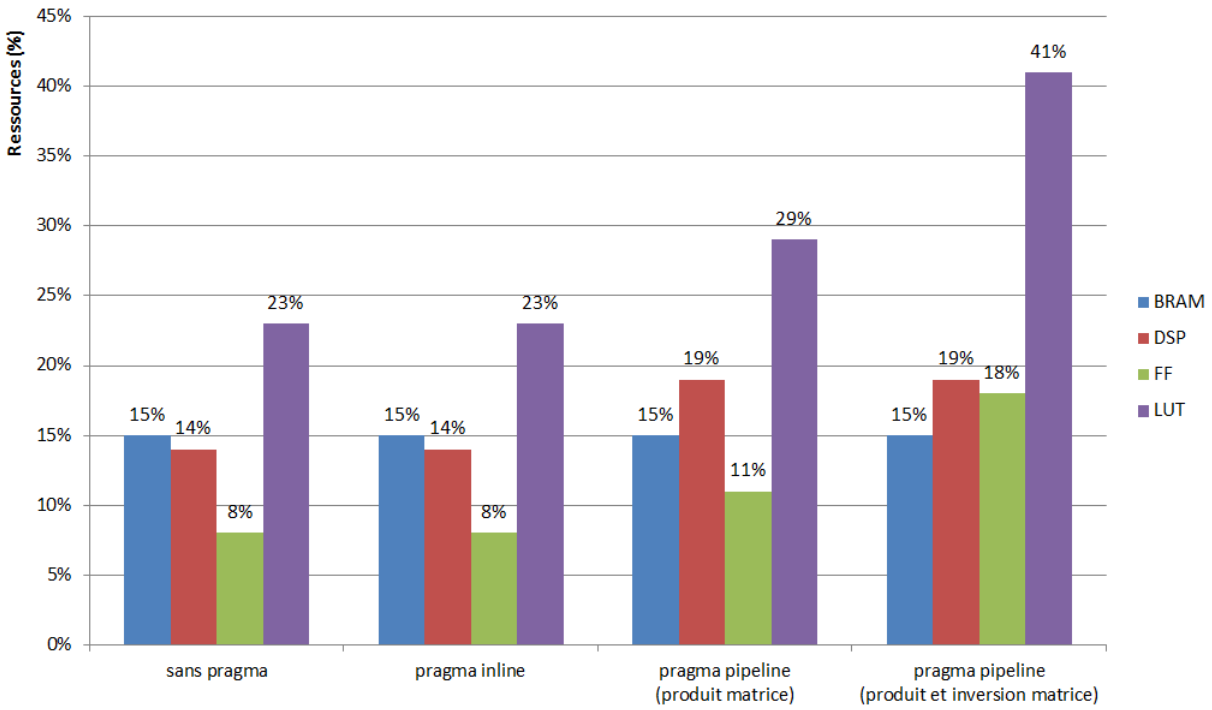


Figure 5.11: *Ressources utilisées par le BFM monolithique.*

LUT utilisées contre 23% de LUT utilisées dans le cas sans optimisation. Par la suite, nous avons appliqué la même directive (pipeline) sur le 'produit de matrices' et 'inversion de matrice'. Dans ce cas nous avons observé une diminution de la latence de 816225 cycles par rapport à la latence obtenue dans le cas sans optimisation. Par contre, le taux de LUT utilisées a augmenté avec 41%, ce qui reste logique et raisonnable.

	Latence (cycles)	**BRAM**	**DSP**	**FF**	**LUT**
Sans pragma (rappel)	16.084.780	15%	14%	8%	23%
Inline (fonction globale)	16.084.780	15%	14%	8%	23%
Pipeline (produit de matrices)	15.349.330	15%	19%	11%	29%
Pipeline (produit et inversion matrices)	15.268.555	15%	19%	18%	41%

Tableau 5.8: *Version monolithique : latence et ressources utilisées (avec optimisation).*

Analyse et synthèse

Ces résultats montrent que l'implémentation de la version monolithique du modèle BFM est réalisable et peut être embarquée sur le véhicule autonome. Cependant, l'utilisation de la version monolithique pour la prise de décision, en cas d'aléas critiques, requière une latence importante et avec les optimisations, la consommation des ressources (LUT) augmente considérablement. Pour un problème de très grande taille, la version monolithique du modèle BFM ne serait probablement pas approprié. Cela dépendra de la latence et des ressources nécessaires pour la résolution du problème.

Version concurrente

La version concurrente consiste à décomposer la mission en différentes phases fonctionnelles (voir Section 4.4). La mission de tracking est donc décomposée en 3 phases : navigation, atterrissage et tracking. Chaque phase est décrite par un modèle BFM avec un MDP pour la décision décrivant la phase en question. Les différents BFM peuvent être résolus indépendemment les uns des autres. Comme pour la version monolithique, chaque MDP du modèle BFM prend en entrée un vecteur de probabilités et un vecteur de rewards.
Caractéristiques des différents BFMs concurrents :

— **BFM navigation** : la phase de navigation est constituée de 9 états et 7 actions ce qui implique 7 matrices de transitions de taille 9x9 et un matrice rewards de taille 9x7. La taille des vecteurs de probabilités et de rewards utilisés en entrée est respectivement 5 et 6.

— **BFM atterrissage** : contient 10 états et 6 actions. Les vecteurs de probabilités et de rewards ont une taille respectivement de 5. Ce qui donne 6 matrices de transitions de taille 10x10 et une matrice rewards de taille 10x6 générées pour la phase d'atterrissage. Ces matrices sont les données d'entrées pour résoudre le MDP avec l'algorithme PI.

— **BFM tracking** : la phase de tracking est représentée par 12 états et 10 actions. La taille des vecteurs de probabilités et de rewards, qui serviront à générer les matrices transitions et rewards, est respectivement 9 et 10. Par conséquent, nous aurons à résoudre un problème MDP avec 10 matrices de transitions de 12x12 plus une matrice rewards de 12x10.

Une fois les différentes phases de la mission décrites avec un modèle BFM, nous utilisons l'outil vivado HLS afin d'avoir l'estimation de la latence et des ressources utilisées à la suite de la synthèse de haut niveau. La Figure 5.12 traduit la latence utilisée (Table 5.9) par chacun MDP des BFMs concurrents décrivant la mission

de tracking. La courbe bleue représente la latence obtenue pour chaque BFM (navigation, atterrissage et tracking) dans le cas sans optimisation. La courbe rouge indique la latence obtenue après applications des directives HLS (pipeline et inline) sur l'algorithme PI. Nous constatons qu'avec les directives utilisées la latence baisse de 17.80% pour le BFM-navigation, 17.90% pour le BFM-atterrissage et enfin de 13.10% pour le BFM-tracking.

	Latence (cycles)	
	Sans optimisation	Avec optimisation
MDP Navigation	171.250	140.762
MDP Atterrissage	210.738	173.014
MDP Tracking	690.860	600.323

Tableau 5.9: *Latence BFMs concurrents.*

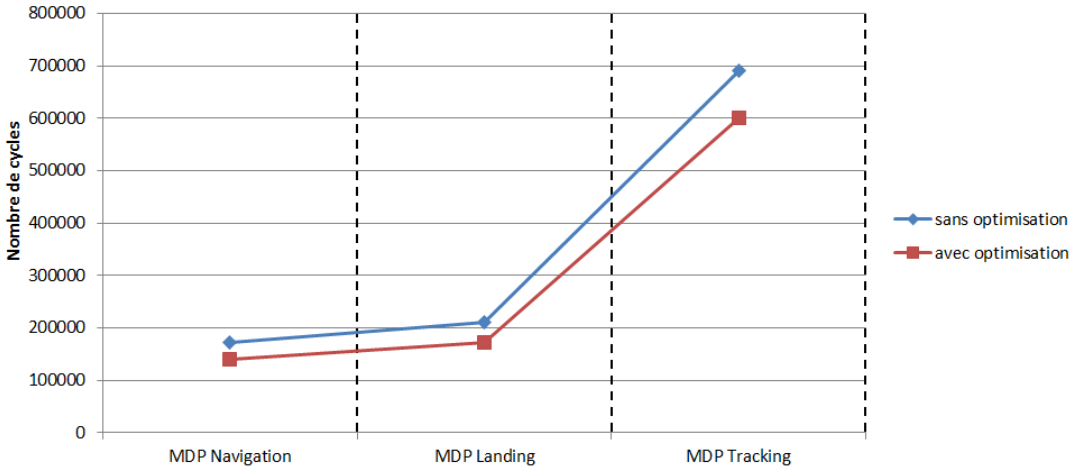


Figure 5.12: *Latence utilisée par chacun des BFMs concurrents de la mission tracking.*

Nous allons, maintenant, présenter l'évolution des ressources utilisées par les BFMs concurrents avec et sans utilisation des directives HLS. Les Figures 5.13, 5.14 et 5.15 montrent les résultats de la synthèse haut niveau en terme de ressources utilisées (BRAM, DSP, FF et LUT). Les graphiques montrent qu'après utilisation des directives (pipeline) le pourcentage de ressources utilisées augmente. Cependant, le parallélisme permet de diminuer la latence (Figure 5.12). Malgré l'augmentation du nombre de ressources utilisées, celui-ci reste raisonnable ; le plus grand BFM, dans le cas de la mission de tracking, ne consomme que 23% de LUT.

Enfin, la Table 5.10 montre les temps d'exécution SW/HW ainsi que l'accélération obtenus en exécutant l'algorithme PI avec les différents MDP navigation, atterrissage et tracking sur la carte ZedBoard de Xilinx avec une fréquence d'horloge de 100 MHz. Le temps d'exécution augmente légèrement en fonction de la taille

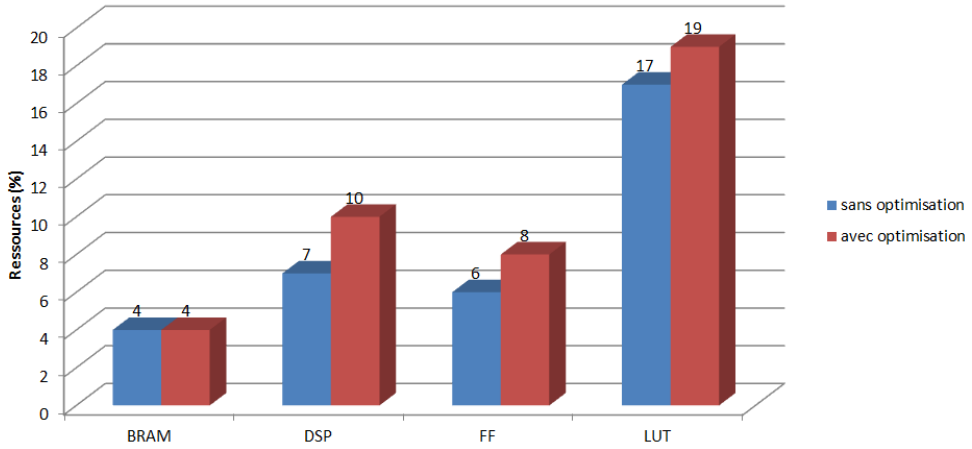


Figure 5.13: *Ressources utilisées par le BFM-navigation.*

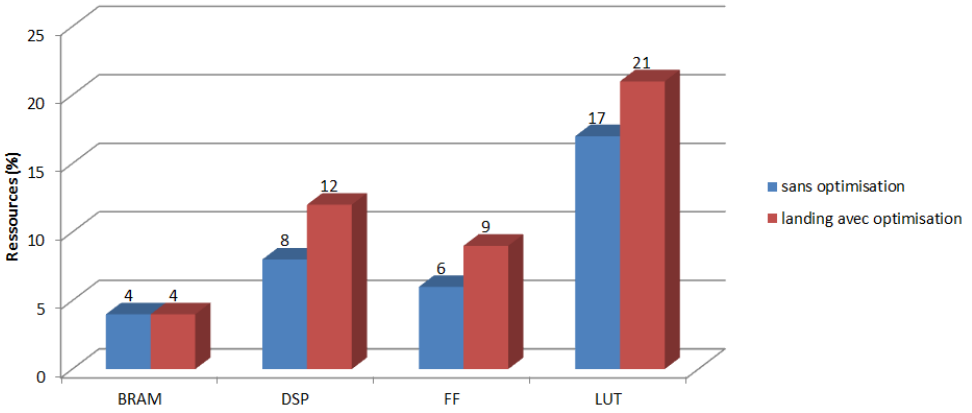


Figure 5.14: *Ressources utilisées par le BFM-landing.*

du MDP mais pour notre cas d'étude un temps de réponse inférieur à 10 ms est suffisant.

Analyse et synthèse

Ces résultats montrent que la version concurrente du modèle BFM est meilleure que la version monolithique. La version concurrente offre de simplifier la mission en la décomposant en différentes phases. Par conséquent, le fait de pouvoir exécuter chaque BFM de façon indépendante des autres BFMs ou bien d'exécuter l'ensemble des BFMs concurrents en parallèle offre de meilleurs résultats en terme de latence et de ressources consommées. Ce qui rend le modèle plus réactif et adaptatif aux événements et produit une politique π de mission (décision) assez rapidement.

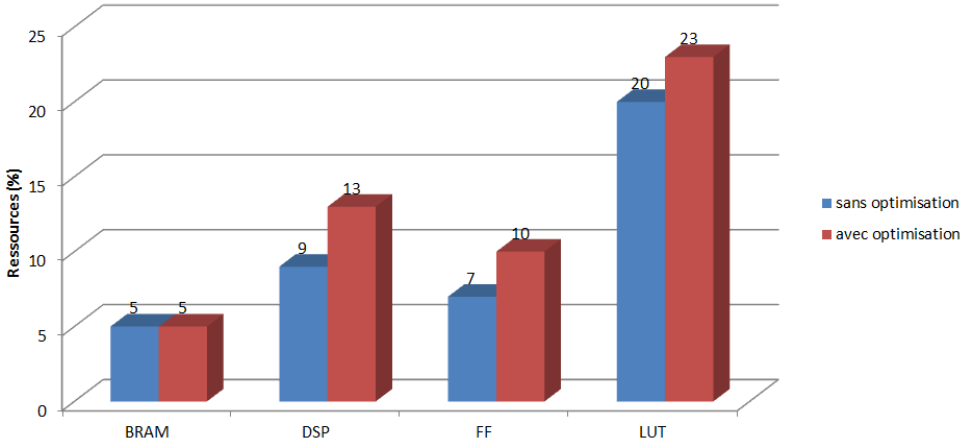


Figure 5.15: *Ressources utilisées par le BFM-tracking.*

	MDP-navigation	MDP-landing	MDP-tracking
Speed up	2.86	1.22	1.14
Execution time (SW clock frequency 1 GHz)	2.7 ms	3.1 ms	8.6 ms
Execution time (HW clock frequency 100 MHz)	0.9 ms	2.6 ms	7.6 ms

Tableau 5.10: *Accélération et temps d'exécution SW/HW du MDP.*

5.4 Conclusion

Dans ce chapitre, nous avons présenté l'implémentation embarquée (HW/SW) du BFM modèle. Nous nous sommes concentrés sur l'implémentation du modèle MDP pour la décision. Pour cela, nous avons dans un premier temps étudié les algorithmes existant pour la résolution d'un MDP et fait un comparatif en termes de temps d'exécution et de latence sur un exemple de grille afin de tester les deux algorithmes sur un grand nombre d'états. Ensuite, nous avons choisi l'algorithme qui correspond le plus à notre cas d'étude, notre choix était d'utiliser *policy iteration*. Nous avons proposé une implémentation matérielle de l'algorithme PI et montré les résultats obtenus par la synthèse de haut niveau en appliquant l'algorithme sur le BFM monolithique et concurrent. Nous avons aussi proposé un frameforwk permettant de décrire et de générer de manière automatique le modèle BFM. L'approche et les résultats présentés dans ce chapitre ont été soumis pour un article à la conférence DATE'2020.

- Chapitre 6 -

Validation du mission manager dans HPeC

Sommaire

6.1 Introduction

Nous abordons dans ce chapitre l'intégration des différents modules dans une architecture globale dédiée à un projet composé de plusieurs niveaux dans le but de contrôler des missions de drone. Le contrôle d'une mission peut se faire à différents niveaux :

— Au niveau mission : à l'aide d'un planificateur de mission, qui inclut la prise de décision sur le plan à exécuter en fonction du contexte environnemental et des événements aléatoires.

— Au niveau architecture : à l'aide d'automate de re-configuration [Gueye et al., 2017], qui permet d'ordonnancer les séquences d'applications, du plan de mission, en choisissant de lancer l'exécution d'une application dans sa version SW ou HW.

L'objectif de ce chapitre est de montrer l'intégration de notre modèle BFM, qui est le planificateur de mission, dans le cadre du projet HPeC et valider les résultats à travers des scénarios de mission définis dans HPeC.

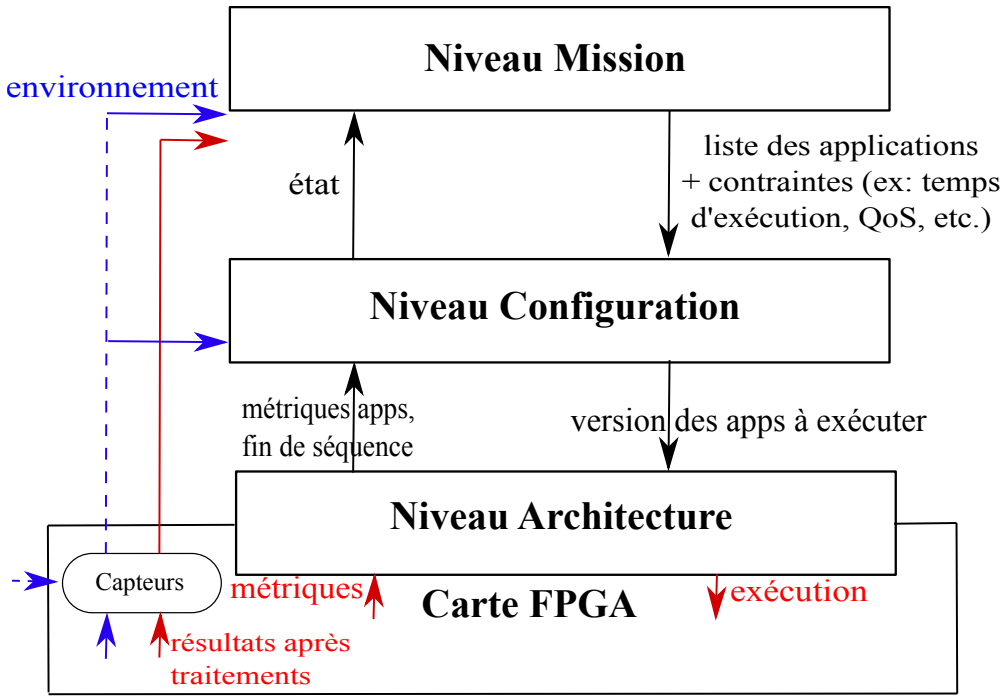


Figure 6.1: *Architecture globale du projet HPeC.*

La Figure 6.1 montre une vue globale de l'architecture de mise en œuvre du projet HPeC. Nous retrouvons dans la figure les différents niveaux HPeC ainsi que l'interaction entre ces différents niveaux. Le niveau mission reçoit les informations

sur l'environnement et produit la décision à exécuter. Ces informations sur l'environnement sont collectées par les différents capteurs embarqués sur le drone. Le niveau mission envoie donc la liste des applications à exécuter au niveau configuration qui choisit la version (HW ou SW) dans laquelle sera exécutée l'application en fonction des performances souhaitées. Le niveau configuration récupère depuis le niveau architecture les métriques des applications (trust, fin de l'exécution de l'application, etc.) et transmet l'état d'exécution des applications au niveau mission. Le niveau mission à son tour améliore la décision qu'il produit en fonction du retour du niveau configuration.

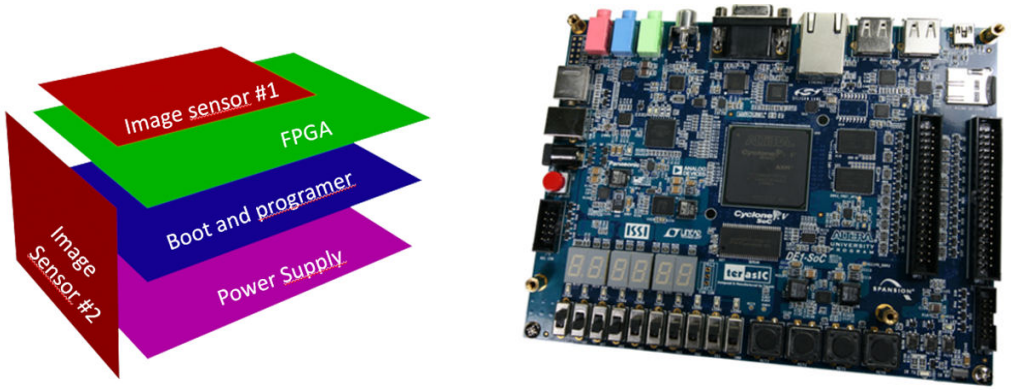
Le chapitre est structuré comme suit, nous commençons par présenter le projet HPeC et son objectif (Section 6.2). Ensuite, dans la Section 6.3, nous définissons la mission étudiée dans le cadre du projet HPeC ainsi que les scénarios mis en place pour valider le comportement du modèle BFM, les capteurs utilisés pour la mission HPeC et nous concluons cette section en présentant les résultats obtenus (en terme de décision sur le plan de mission à exécuter) par le modèle BFM. La Section 6.4 montre d'un côté l'architecture ROS établie pour le projet HPeC et l'intégration de notre planificateur de mission dans le projet, et d'un autre côté, les outils de simulations et l'architecture autour dans le but de simuler les scénarios de mission définis dans la Section 6.3. Enfin, nous présentons la validation du prototype proposé par le projet HPeC. Nous clôturons ce chapitre avec une conclusion.

6.2 Contexte de validation

L'ensemble des travaux de recherche présentés dans les chapitres précédents font partis du projet HPeC (ANR 2015-2019). L'objectif du projet HPeC est de démontrer la capacité d'un système auto-adaptatif de faire face aux demandes de calcul haute performance des systèmes autonomes tel que les drones. Le projet HPeC propose de re-configurer dynamiquement une carte embarquée de type SoC-FPGA Altera pour exécuter les applications embarquées afin de réaliser une mission de drone.

Dans le but de démontrer cet aspect auto-adaptatif des systèmes autonomes, le projet HPeC propose une solution qui repose sur trois différents niveaux d'adaptation :

— **Niveau mission** : consiste à proposer et mettre en œuvre un planificateur de mission dans le but de gérer la mission du drone et proposer des plans de mission. Le planificateur de mission s'appuie sur des modèles probabilistes (BNs et MDPs) car ils prennent en compte les incertitudes liées à l'environnement ou au système lui-même. Le modèle BFM proposé répond aux attentes du niveau mission décrit par le projet HPeC. L'objectif est de planifier la mission du drone en fonction d'événements aléatoires. Ces événements peuvent être liés au contexte externe (obstacle, manque de luminosité, etc) ou interne (défaillance capteurs, ressources insuffisantes, QoS applications, etc.).



((a)) *Illustration des couches de la carte HPeC.*

((b)) *Carte Altera utilisée pour les simulations.*

Figure 6.2: *Cartes embarquées utilisées dans le cadre du projet HPeC.*

— **Niveau configuration** : proposer un modèle pour ordonnancer les séquences d'applications ou de versions d'applications dans le but de configurer ou reconfigurer dynamiquement l'exécution des applications sur la carte embarquée SoC-FPGA. Les applications seront exécutées soit dans leurs versions SW ou HW. Le contrôleur de configuration est mis en œuvre à l'aide d'un automate obtenu par synthèse [Gueye et al., 2017] [Delaval et al., 2015].

— **Niveau architecture** : configuration dynamique d'une architecture SoC-FPGA. Conception d'une carte SoC-FPGA pour le projet HPeC sur laquelle vont s'exécuter les applications embarquées sur le drone.

Dans la suite de ce chapitre, nous allons présenter la mission et les scénarios de validation du modèle BFM pour la planification de la mission ainsi que l'interaction entre le contrôleur de configuration et le planificateur de mission.

Le projet HPeC a pour objectif non seulement la validation individuelle des différents niveaux le constituant, mais aussi proposer un prototype de démonstration pour le contrôle d'une mission de drone en utilisant la simulation hardware avec HIL (Hardware In The Loop) et la carte embarquée conçue dans le cadre du projet HPeC. La Figure 6.2(a) présente une illustration des différentes cartes filles composant le système (smartcam/calculateur) HPeC. Elle contient quatre cartes, une carte supérieure pour les entrées/sorties capteurs et images, une carte inférieure pour l'alimentation, une carte avec le FPGA et enfin une carte de boot avec le processeur. La Figure 6.2(b) montre une carte DE1-SoC Altera utilisée pour les simulations des scénarios de mission HPeC notamment avec HIL. Le choix d'utiliser une carte Altera s'appuie sur le fait qu'elle supporte la configuration (re-configuration) dynamique des applications embarquées ce qui entre dans les objectifs du projet HPeC.

6.3 Validation du modèle BFM avec les scénarios HPeC

Dans les chapitres précédents, nous avons détaillé le principe et fonctionnement de notre modèle BFM pour la planification de mission de véhicule autonome ainsi que la validation de ce dernier. Dans cette section, nous allons présenter la mission considérée dans le cadre du projet HPeC, les différents scénarios définis pour valider les points importants (apport de l'évaluation de la QoS des applications, adaptation de la décision au changement durant la mission) du modèle BFM appliqué à la mission HPeC, ainsi que la liste des capteurs embarqués sur le drone et utilisés pour la mission HPeC.

6.3.1 Mission et scénarios envisagés

Dans le cadre du projet HPeC, le véhicule autonome considéré est un drone de type hexacopter (drone avec six hélices) effectuant une mission de tracking. La mission de tracking consiste à suivre une trajectoire prédéfinie avec un ensemble de waypoints (points de trajectoire) reliant la base (départ) à une zone de recherche (arrivée). Une fois arrivé à la zone de recherche, le drone cherche à détecter la cible à suivre. Lorsque la cible est détectée par le drone, elle est traquée par ce dernier, en essayant de ne pas la perdre et en faisant face aux différents aléas externes (contexte environnemental) et interne (état du système).

Le planificateur de mission, présenté dans le Chapitre 4, qui s'appuie sur le modèle BFM, a pour objectif de gérer la mission de tracking réalisée par le drone. Dans ce qui suit, nous allons présenter les différents scénarios utilisés pour la validation du modèle BFM au sein du projet HPeC.

La particularité du bloc de diagnostic, BN, du modèle BFM est l'évaluation de la QoS des applications embarquées en fonction du contexte environnemental observé et du taux d'occupation des ressources de calculs (CPU et FPGA). D'autre part, le bloc diagnostic nous renvoie une estimation de la version algorithmique de l'application à exécuter en fonction du contexte externe et interne et de la QoS souhaitée pour l'application en question. Les scénarios décrit ci-après seront utilisés pour la validation de l'avantage de l'ajustement de la QoS à l'aide des BNs dans le cas de la mission de tracking.

— Scénario 1 : cas nominal, déroulement de la mission sans la présence d'éventuelles erreurs liées à l'environnement ou bien au système.

— Scénario 2 : détection d'un problème de vibrations en utilisant le capteur IMU. L'observation de l'erreur de vibration est reportée sur le BN de l'application de tracking, utilisée pour la mission, comme entrée (évidence).

— Scénario 3 : la vitesse de la cible suivie est beaucoup plus rapide que la vitesse de déplacement du drone. Cela implique une forte probabilité de perdre la cible

de vue sur l'image à traiter.

— Scénario 4 : correspond à une extension du scénario 3 avec l'observation de présence de vent fort en plus de la cible qui se déplace à grande vitesse par rapport au drone.

Pour la validation de la partie prise de décision du modèle BFM, nous avons défini deux stratégies pour le déroulement de la mission de tracking. Une stratégie dite 'safety' qui favorise la sécurité au niveau de la mission et du drone. Par conséquent, la stratégie 'safety' met la priorité sur les applications de sûreté comme la détection et évitement d'obstacle, la recherche de zone libre pour un atterrissage en cas d'urgence, etc. La deuxième stratégie est appelée 'mission first', elle consiste à donner la priorité à la réalisation de la mission. Dans le cas de la mission de tracking, donner la priorité à l'application de détection et suivi de la cible tant en gardant un minimum de sécurité (avec la détection d'obstacle). Nous décrivons les deux scénarios suivants pour la validation de la prise de décision du modèle BFM.

— Scénario 5 : contexte avec présence de vibrations auquel nous ajoutons un obstacle sur la trajectoire du drone.

— Scénario 6 : la cible se déplace à une vitesse supérieure à la vitesse du drone et nous ajoutons la présence d'obstacle sur la trajectoire.

Nous déroulerons les deux scénarios 5 et 6 une fois avec la stratégie 'safety' et une fois avec la stratégie 'mission first', ensuite nous observons la décision prise par le modèle BFM pour chacun de ces scénarios.

Avant de décrire les résultats de l'exécution des scénarios, nous allons décrire les différents capteurs utilisés pour la mission de tracking du projet HPeC. Ces capteurs seront utilisés pour récolter les informations nécessaires au déroulement des scénarios que nous avons défini. Ensuite, dans la Section 6.3.3, nous présenterons les résultats obtenus pour la validation du modèle BFM.

6.3.2 Capteurs utilisés dans HPeC pour le MM

La gestion de la mission revient à générer une décision sur les applications à exécuter et par conséquent un nouveau plan de mission. Afin de produire une décision, le planificateur de mission a besoin des données de capteurs embarqués sur le drone. Ces données de capteurs sont les suivantes :

— **Données des capteurs physiques** :
  — IMU (Unité de Mesure Inertielle) : utilisé pour détecter la présence de vibrations causées soit par un déséquilibre du drone, un problème d'hélice se traduisant par un vent irrégulier.
  — Capteur de luminosité : indique si l'intensité de la luminosité diminue ou augmente.
  — Altimètre (PX4) : indique l'altitude à laquelle vole le drone.

  - Auto-pilote : pour récupérer la vitesse du drone.
  - Batterie : monitoring de la consommation d'énergie de la batterie.
- **Monitoring des ressources** :
  - Ressource FPGA : indique le nombre de tuiles du FPGA utilisées.
  - Ressource CPU : indique la charge CPU utilisée en pourcentage (%).
- **Métriques renvoyées par les applications embarquées** :
  - Détection d'obstacle : indique si un obstacle est détecté ou pas.
  - Recherche de zone d'atterrissage : métrique indiquant si une zone libre est détectée ou non afin d'atterrir en cas d'urgence.
  - Détection de cible : indique si la cible à traquer est détectée ou non, ou bien perdue.
  - NCC-tracking : métrique de confiance du suivi de la cible [Kalal et al., 2011]. Si la valeur du NCC est supérieure à 0.8 (NCC>0.8) alors le suivi de la cible se déroule bien sans problème. Une seconde métrique est combinée avec la métrique NCC, cette métrique 'track' indique si le suivi de la cible (tracking) est valide ou non. Si la valeur du track est '0' alors le tracking n'est plus valide. Par contre, si la valeur du track est égale à '1' et que la valeur du NCC est inférieure à 0.8 ; cela vaudra dire que la cible est potentiellement perdue.
- **Retour du contrôleur de configuration** :
  - 'Achievable' : indique si la solution (décision) proposée par le planificateur de mission a pu être implémentée par le contrôleur de configuration ou non.

6.3.3 Résultats de validation

Maintenant que les différents scénarios de la mission de tracking sont définis, nous allons les dérouler. Les scénarios de 1 à 4 (de la section 6.3.1) vont aider à montrer l'avantage de l'évaluation de la QoS des applications par le modèle BFM ainsi que l'impact sur la prise de décision. Les scénarios 5 et 6 (Section 6.3.1) vont être utilisés pour montrer l'évolution de la prise de décision en fonction de différentes contraintes : stratégie suivi pour la mission, taux d'occupation de ressources de calculs (CPU et FPGA) et contexte environnemental.

Nous utilisons le logiciel Matlab qui offre une toolbox pour la résolution des MDP. Nous exécutons le modèle BFM pour la décision durant 100 steps avec une résolution sur horizon fini du MDP de la mission. Une nouvelle décision est calculée à chaque fois que les valeurs de probabilités, calculées par les BNs, changent de manière significative. Ce changement de probabilités indique la présence d'un événement (ex : obstacle). Les résultats obtenus après simulation sous Matlab sont présentés ci-dessous.

Intérêt de l'évaluation de la QoS des applications

Le MDP décrivant la mission de tracking est régulièrement évalué et prend comme entrée les probabilités produites par le bloc diagnostic. Pour les scénarios définis ci-dessus, le bloc diagnostic contient le BN de l'application 'tracking' avec toutes les versions algorithmiques possibles.

Nous comparons les résultats obtenus entre la mission de tracking de référence et la mission de tracking adaptative en terme de temps de suivi de la cible (temps de tracking). La mission de référence correspond à la mission de tracking (définie plus haut) pour laquelle une seule et unique version de l'application de tracking est considérée. Quant à la mission adaptative, elle consiste en la mission de tracking avec la proposition de différentes versions algorithmiques de l'application de tracking en fonction du contexte environnemental de la mission. La Table 6.1 montre les résultats obtenus.

Scénarios	temps de tracking (nbr de steps)		# Cycles (10^6)
	Référence	BFM	Version du tracking
Nominal (scénario 1)	51	51	103 (320x240 frame)
Présence de Vibrations (scénario 2)	40	65	292 (nominal + stabilization)
Vitesse de la cible (scénario 3)	40	72	264 (640x480 frame)
Vent (scénario 4)	35	56	264

Tableau 6.1: *comparaison du temps de tracking entre la mission de référence et sa description avec le modèle BFM.*

Les résultats montrent que le temps de suivi de la cible varie d'un scénario à un autre.

— Scénario 1 : avec le scénario nominal, le temps de suivi de la cible obtenu avec le modèle BFM est similaire au temps obtenu avec la mission de référence. La mission de tracking est réalisée avec succès dans les deux cas, si aucune défaillance n'est détectée.

— Scénario 2 : observation de vibrations durant la phase de suivi de la cible. nous remarquons que dans le cas de la mission de référence, la mission a échouée à cause des vibrations. Cependant, avec la mission adaptative, le cible continue d'être traquée par le drone en activant une nouvelle version algorithmique de

l'application 'tracking'. Cette nouvelle version consiste à exécuter l'application 'tracking' avec l'activation de l'application 'extra-stabilisation' d'image.

— Scénario 3 : scénario avec vitesse de la cible supérieure à celle du drone. Dans ce cas de figure, le drone peut perdre la cible. Avec la mission adaptative (modèle BFM), le problème est corrigé en prenant de l'altitude avec le drone. Ce qui va faire en sorte d'augmenter la R.O.I (Region Of Interest) et relancer la détection de la cible. Les résultats montrent qu'avec le modèle BFM de la mission adaptative, la cible est traquée plus longtemps qu'avec la mission de référence.

— Scénario 4 : comme c'est une extension du scénario 3, la solution appropriée est déjà activée (prendre de l'altitude avec le drone). Cependant, avec la présence du vent, l'énergie de la batterie est consommée rapidement. Le temps de suivi de la cible reste supérieur au temps de suivi avec la mission de référence. Néanmoins, la cible est poursuivie moins longtemps que dans le scénario 3. Cela est due à l'augmentation de la consommation de l'énergie de la batterie.

Les résultats de l'expérimentation ont montré que nous pouvons prolonger le temps de suivi de la cible en considérant différentes versions algorithmiques pour l'application de 'tracking'. L'activation de ces versions du 'tracking' va faire en sorte de corriger ou diminuer la défaillance observée. Par conséquent, augmenter le temps de tracking de la cible.

En plus du temps de tracking de la cible, nous donnons aussi quelques estimations du temps d'exécution des différentes versions de l'application de 'tracking' afin d'illustrer la variations de l'utilisation des ressources. Les temps d'exécution sur le processeur embarqué Cortex A9 sont obtenus avec les optimisations du co-processeur NEON 625. Les applications utilisées sont : TLD pour le 'tracking' [Kalal et al., 2012] appliqué sur deux d'images de taille 320x240 et 640x480, l'extra-stabilisation et la recherche de zone d'atterrissage.

Comme présenté dans la Table 6.1, la version algorithmique initiale de l'application 'tracking' peut tourner à 9 fps (soit 103.10^6 cycles), si le CPU est utilisé en intégralité. Avec le scénario 2, la nouvelle version choisie pour le 'tracking' (tracking initiale + extra-stabilisation) est plus lente et a besoin de 292.10^6 cycles. Pour le scénario 3 (vitesse cible $>>$ vitesse drone), la version du 'tracking' adoptée est évidement plus lente et nécessite 264.10^6 cycles pour s'exécuter. Donc, toutes ces versions algorithmiques de l'application 'tracking' requièrent un nombre de ressources différents pour s'exécuter sur le CPU. Par conséquent, si l'activation d'une application, en plus de celles qui sont en exécution, surcharge la capacité du CPU ; alors la QoS demandée ne pourra pas être assurée. Elle doit donc être prise en compte dans la gestion globale de la mission ; d'où l'intérêt de l'évaluation de la QoS d'une application par notre modèle BFM.

Applications	Version algorithmique	Ressources
Tracking	V0 : nominal	S(SW :285ms) M(HW :10ms) F(HW : 7ms)
	V1 : V0+ pré-filtrage	S(HW : 286ms) M(SW-HW :10+1ms) F(HW :7ms)
	V2 : V0+ re-dimensionner R.O.I	S(SW : 730ms) M(HW :10ms) F(HW :7ms)
	V3 : V0 + stabilisation	S(SW :540 ms) M(SW-HW :285+10ms) F(HW :10 ms)
	V4 : V1 + stabilisation	S(HW : 540ms) M(SW-HW :255+10ms) F(HW :10ms)
	V5 : V2 + stabilisation	S(SW-HW :410+10ms) M(SW-HW : 30 +10ms) F(HW :10ms)
Recherche zone atterrissage	recherche zone 'T'	S(SW :394 ms) F(HW : 7ms)
	recherche zone d'urgence	S(SW : 255 ms) F(HW : 7ms)
Détection d'obstacle	Lidar	5ms
	Fusion	7ms
Évitement d'obstacle	V0 : tourner à droite	1ms
	V1 : path-planning	SW :1s
Path-planning ?	GA version	SW :1s

Tableau 6.2: *Consommation ressources pour les différentes versions des applications de la mission.*

Validation de la prise de décision avec le modèle BFM

La mission de tracking étudiée pour le projet HPeC peut être décrite par la version monolithique ou bien par la version concurrente du modèle BFM. Dans cette partie, nous allons présenter les résultats obtenus sur de temps de suivi de la cible et des décisions prises pour faire face aux différents événements rencontrés. Avant d'exécuter les scénarios 5 et 6 (voir Section 6.3.1), nous définissons deux stratégies de déroulement de la mission, une stratégie 'safety' et une stratégie 'mission first'. Les scénarios 5 et 6 sont exécutés une première fois en suivant la stratégie 'safety' et une seconde fois en suivant la stratégie 'mission first'.

Dans notre cas d'étude (mission de tracking), nous évaluons en premier les différentes variantes des applications exécutées par le mission. La Table 6.2 donne la liste de ces variantes applicatives. Pour une version algorithmique, nous considérons trois variantes d'implémentation. Ces variantes correspondent à une vitesse d'exécution *rapide* (F), *lente* (S) ou *moyenne* (M). Les temps d'exécution estimés s'appuient sur des implémentations HW/SW sur une carte TeraSIC SoCKit qui tourne sur un Cyclone V Altera. Le Cortex A9 tourne avec Linux à une fréquence d'horloge de 925MHz. La carte SoCKit communique avec une carte auto-pilote Pixhawk. Le FPGA contient 4 tuiles qui peuvent être re-configurées dynamiquement.

La Table 6.3 et la Table 6.4 représentent les résultats obtenus en exécutant le scénario avec vibrations (scénario 5) avec les deux stratégies de réalisation de la mission (safety et mission first). Avec la stratégie 'safety', nous constatons que durant la phase de tracking, les actions 'détection d'obstacle' et 'recherche de zone' pour l'atterrissage sont plus prioritaires que l'action de 'tracking'. Lorsqu'un manque de ressources de calculs (CPU ou FPGA) est observé, la version actuelle du tracking (tracking + extra-stabilisation) est dégradée à la version initiale du 'tracking' car la version initiale consomme moins de ressources. Avec la stratégie 'mission first', la priorité est donnée aux différentes versions de l'application de 'tracking' tout en

États	**Ensemble des actions**	**Évènements**	**Ressources**
Base	tacke-off	pas d'événement	-
WayPoint 1	suivi de trajectoire détection d'obstacle	pas d'événement	5ms
WayPoint Intermédiaire	suivi de trajectoire détection d'obstacle recherche zone 'T'	pas d'événement	5+394ms
Zone de recherche	suivi de trajectoire détection d'obstacle recherche de zone 'T' détection de la cible	cible détectée	5+394+10ms
Tracking (phase 1)	tracking + extra-stabilisation (V3) détection d'obstacle recherche de zone 'T'	obstacle détecté	10+5+394ms
	tracking + extra-stabilisation évitement d'obstacle recherche de zone 'T'	obstacle détecté	10+1+394ms
Tracking (phase 2)	tracking + extra-stabilisation (Désactiver) évitement d'obstacle recherche zone atterrissage d'urgence	obstacle détecté	1+255ms
	tracking + extra-stabilisation détection d'obstacle recherche zone atterrissage d'urgence	ressources insuffisante pour exécuter V3	10+5+255ms
Tracking (phase 3)	tracking initial détection d'obstacle recherche zone atterrissage d'urgence	pas d'événement	10+1ms

Tableau 6.3: *Politique obtenue avec le scénario 1 (vibrations) sous la stratégie 'safety'.*

gardant un minimum de safety pour le drone. Par conséquent, lorsque les ressources de calculs sont insuffisantes, au lieu de dégrader voir désactiver la version actuelle du 'tracking', nous désactivons l'application de 'recherche de zone' pour l'atterrissage et garder la 'détection d'obstacle' en exécution pour la safety.

La Table 6.5 et la Table 6.6 représentent les résultats obtenus avec le scénario 6 (vitesse cible). Dans ce scénario, nous observons que la vitesse de la cible est beaucoup plus élevée que la vitesse du drone. Par conséquent, la version appropriée du 'tracking' a exécuté est prendre de l'altitude avec le drone tout en exécutant le 'tracking' (V2). Durant la mission, nous constatons que le taux de ressources non occupés est insuffisant pour exécuter correctement les applications nécessaires au déroulement de la mission. Dans le cas où la mission est réalisée avec la stratégie 'safety', la version courante du 'tracking' (V2) est dégradée vers la version V1 du 'tracking' (traking + pré-filtrage) ; et nous maintenons la sécurité de la mission en laissant activer les actions de 'détection d'obstacles' et de 'recherche de zones d'atterrissage'. Avec la stratégie 'mission first', comme la cible se déplace à grande vitesse par rapport au drone, le modèle BFM choisit de garder V2 en l'exécutant

États	**Ensemble des actions**	**Évènements**	**Ressources**
Base	tacke-off	pas d'événement	-
WayPoint 1	suivi de trajectoire détection d'obstacle	pas d'événement	5ms
WayPoints intermédiaire	suivi de trajectoire détection d'obstacle recherche de zone 'T'	pas d'événement	394+5ms
Zone de recherche	suivi de trajectoire détection d'obstacle recherche de zone 'T' détection de la cible	pas d'événement	5+394+10ms
Tracking (phase 1)	tracking + extra-stabilisation (V3) détection d'obstacle recherche de zone 'T'	cible détectée	10+5+394ms
	tracking + extra-stabilisation évitement d'obstacle recherche zone atterrissage d'urgence	obstacle détecté	10+1+255ms
Tracking (phase 2)	tracking + extra-stabilisation (Dégrader) évitement d'obstacle recherche zone atterrissage d'urgence	obstacle détecté	1+7ms
	tracking + extra-stabilisation détection d'obstacle recherche zone atterrissage d'urgence	ressources insuffisante pour exécuter le tracking	10+5+7ms
Tracking (phase 3)	tracking + extra-stabilisation détection d'obstacle	ressources insuffisantes pour exécuter V3	10+5ms

Tableau 6.4: *Politique obtenue avec le scénario 1 (vibrations) sous la stratégie 'mission first'.*

États	**Ensemble des actions**	**Évènements**	**Ressources**
Base	tacke-off	pas d'événement	-
WayPoint 1	suivi de trajectoire détection d'obstacle	pas d'événement	5ms
WayPoint intermédiaire	suivi de trajectoire détection d'obstacle recherche zone 'T'	pas d'événement	5+7ms
Zone de recherche	suivi de trajectoire détection d'obstacle recherche zone 'T' détection de la cible	détection cible	5+7+10ms
Tracking (phase 1)	tracking initiale (V0) détection d'obstacle recherche zone 'T'	vitesse cible élevée	10+5+7ms
	tracking + prendre altitude (V2) détection d'obstacle and 'T' zone search	obstacle détecté	730+5+7ms
Tracking (phase 2)	V2 (Désactiver) évitement d'obstacle recherche zone atterrissage d'urgence	obstacle détecté	1+7ms
	V2, détection d'obstacle recherche zone atterrissage d'urgence	ressources insuffisantes pour exécuter V2	730+5+7ms
Tracking (phase 3)	tracking + pré-filtrage (V1) détection d'obstacle recherche zone atterrissage d'urgence	pas d'événement	410+10+5+7ms

Tableau 6.5: *Politique obtenue avec le scénario 2 (vitesse cible) sous la stratégie 'safety'.*

dans sa version HW (la plus rapide) afin de garantir l'activité de tracking. Nous constatons bien que la décision prise par le modèle BFM est adaptée à chaque fois en fonction de la stratégie de la mission.

La Table 6.7 indique les temps de tracking obtenus en exécutant le scénario 5 et 6 avec la stratégie 'safety' et avec la stratégie 'mission first'. Nous constatons que la cible est traquée plus longtemps avec la stratégie 'mission first' qu'avec la stratégie 'safety'. Comparativement aux résultats présentés dans la Table 6.1, le fait de considérer l'utilisation des ressources dans le processus de décision augmente le temps de réalisation de la mission, notamment le temps de tracking dans notre cas.

Bilan

Les résultats des différentes expérimentations montrent que le modèle BFM apporte les avantages suivants :

— Considérer plusieurs versions d'une application embarquée (ex : application de tracking) permet d'augmenter la durée de la mission (ex : mission de tracking).

États	**Ensemble des actions**	**Évènements**	**Ressources**
Base	tacke-off	pas d'événement	-
WayPoint 1	suivi de trajectoire détection d'obstacle	pas d'événement	5ms
WayPoint intermédiaire	suivi de trajectoire détection d'obstacle recherche zone 'T'	pas d'événement	5+394ms
Zone de recherche	suivi de trajectoire détection d'obstacle recherche zone 'T' détection de la cible	cible détectée	5+394+10ms
Tracking (phase 1)	tracking initiale (V0) détection d'obstacle recherche zone 'T'	vitesse cible élevée	10+5+394ms
	tracking + prendre altitude (V2) détection d'obstacle recherche zone 'T'	obstacle détecté	10+5+394ms
Tracking (phase 2)	V2 (Dégrader), évitement d'obstacle recherche zone atterrissage d'urgence	obstacle détecté	1+255ms
	V2 (dégrager), détection d'obstacle recherche zone atterrissage d'urgence	vitesse cible élevée	7+5+255ms
Tracking (phase 3)	V2 (version HW) détection d'obstacle	pas d'événement	7+5ms

Tableau 6.6: *Politique obtenue avec le scénario 2 (vitesse cible) sous la stratégie 'mission first'.*

Scénario	Temps de tracking (nbr steps)	
	Stratégie 'safety'	Stratégie 'mission first'
Scénario 5 (vibrations)	64	68
Scénario 6 (vitesse cible)	71	76

Tableau 6.7: *Durée de l'activité de suivi de la cible (tracking).*

— L'évaluation de la QoS d'une application permet au système de choisir la version appropriée de l'application à exécuter en fonction du contexte observé.

— Les résultats de simulations montrent que la décision prise par le modèle BFM permet au système de s'adapter aux différents événements (environnemental, disponibilité des ressources, système embarqué, etc.).

— La politique (décision) produite par le modèle BFM correspond au plan attendu à exécuter par le drone pour chaque scénario.

6.4 Intégration du mission manager dans HPeC

Une fois le planificateur de mission défini et implémenté, il peut être intégré à l'ensemble du projet HPeC. Pour cela, le planificateur de mission (ainsi que les autres modules du projet) sera ajouté à l'architecture ROS décrite dans le cadre du projet HPeC. Dans ce qui suit, nous présentons la démarche suivie pour intégrer notre planificateur de mission dans le projet en utilisant ROS ainsi que l'interaction avec le contrôleur de configuration.

6.4.1 Méthodologie d'intégration avec ROS

Afin d'effectuer des simulations de mission de drone à l'aide de Hardware In The Loop, dans un premier temps, l'implémentation et l'exécution d'une boucle de simulation est faite sur PC. La Figure 6.3 montre les quatre principaux acteurs pour effectuer une simulation de mission de drone à l'aide de ROS. ROS est un framework qui permet de développer des programme dédiés aux UAV, il est très utilisé dans le domaine de la robotique (voir ci-après la définition). Gazebo est un simulateur dans lequel nous pouvons décrire des mondes virtuels pour simuler des missions de véhicules autonomes (plus de détails ci-dessous). SITL, quant à lui, reproduit le comportement du l'auto-pilote PX4 embarqué sur une carte PixHawk et offre un simulateur de vol pour faire déplacer le drone dans Gazebo. Enfin, MAVROS fait le lien entre les programmes ROS et l'auto-pilote (plus de détails ci-dessous).

Présentation ROS

Robot Operating System (ROS) est un framework très utilisé pour la mise en œuvre de logiciels embarqués pour le contrôle de drones et robots. Il est largement utilisé pour les communications et les interactions entre les différentes applications (tâches) et entre les applications et les capteurs embarqués sur le robot. ROS est constitué d'une collection d'outils, librairies et conventions qui visent à simplifier la création de tâches complexes et robustes sur une grande variété de plateformes robotiques. Chaque programme dans ROS est appelé 'nœud'. Lors de l'utilisation de ROS, un nœud appelé 'ROS master' est créé, il fournit des services de nommage et

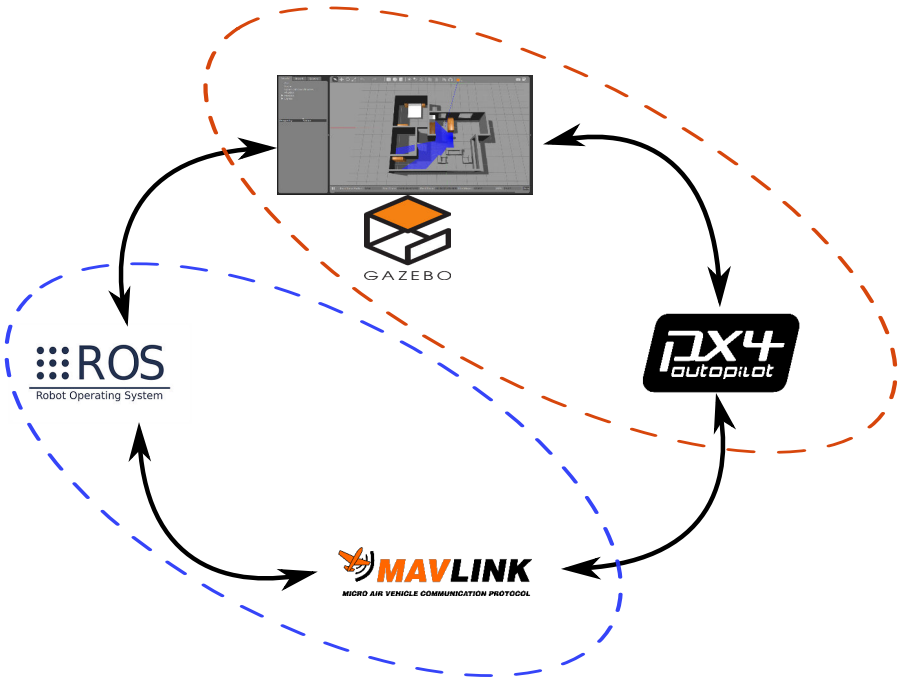


Figure 6.3: *Boucle de simulation ROS*

des abonnements aux autres nœuds du système ROS. Chaque tâche indépendante peut être séparée en nœuds ROS qui communiquent entre eux à l'aide de canaux. Ces canaux sont appelés 'topics'. Chaque topic peut avoir une multitude de nœuds diffuseurs et de nœuds récepteurs. Cette structure aide à séparer les différentes parties du code en plusieurs modules et les gérer séparément. Au niveau logiciel, ROS est une solution bien adaptée pour le développement rapide de logiciels embarqués pour la robotique, il doit être pris en compte pour les simulations. Cela peut être fait à l'aide d'une approche Hardware In The Loop (HIL).

Présentation du simulateur Gazebo

Gazebo [Koenig and Howard, 2004] est un logiciel très utilisé par la communauté robotique et est totalement open-source. Les utilisateurs peuvent facilement créer et définir des environnements virtuels 3D, des modèles de capteurs et des protocoles de communications. Grâce à l'ODE (Open Dynamique Engine), Gazebo peut présenter un modèle de système robot avec une grande précision dans des conditions en temps réel. De plus, il peut appliquer des forces et des moments dans le modèle de drone à six-degrés de liberté. Afin de collecter les données de capteurs et le flux vidéo à partir du monde virtuel, des modèles de capteurs et de caméras sont associés avec des plugins développés en C. Le même principe est utilisé pour contrôler les moteurs du drone avec l'auto-pilote.

Présentation de SITL

Software In The Loop (SITL) est un simulateur qui permet d'exécuter différents types de robots (planeur, copter ou terrestre) sans aucune plateforme matérielle. SITL représente une version d'auto-pilote appelée ArduPilot sur PC. Il fournit aux concepteurs un exécutable natif qui permet de tester le comportement du code sans le matériel. Lors de l'exécution avec SITL, les données de capteurs proviennent du modèle dynamique de vol décrit dans le simulateur de vol. Le protocole de communication utilisé par ArduPilot est MAVLink (Micro Air Vehicle Link). C'est un protocole pour communiquer avec des véhicules sans pilotes. Par conséquent, un pont est nécessaire entre l'auto-pilote et ROS.

La liaison entre l'auto-pilote et ROS est assurée par MAVROS. MAVROS est un package ROS qui offre des pilotes de communications pour différents auto-pilotes en utilisant le protocole de communication MAVLink. Grâce à ce package, le programme développé sous ROS est capable d'envoyer des commandes et recevoir des données à partir de l'auto-pilote.

Approche Hardware In The Loop

Dans cette partie, nous allons voir comment réaliser des simulations de scénarios de mission de drone à l'aide de l'approche HIL (Hardware In The Loop). La carte sur laquelle s'exécuteront les applications embarquées est un Cyclone V Altera. Donc les connexions externes du Cyclone V doivent être adaptées à HIL. En effet, les capteurs physiques et caméras embarqués sur un drone réel seront remplacés par les modèles de capteurs et caméras fournis par Gazebo ; l'auto-pilote PixHawk sera remplacé par SITL. La Figure 6.4 représente l'architecture établie afin de pouvoir effectuer des simulations à l'aide de HIL. Nous distinguons deux principaux blocs, le PC hôte et le Cyclone V.

Grâce à l'utilisation de ROS, nous pouvons exploiter les canaux 'topics' pour établir des communications entre les différents composants du Cyclone V. Parmi ces composants, nous retrouvons des nœuds ROS. Le nœud ROS 'mission manager' contient notre modèle BFM qui permet de superviser la mission du drone dans le but de détecter les défaillances liées au contexte interne ou externe (environnement ou système) et prendre des décisions sur le plan de mission à exécuter en fonction du diagnostic évalué. Une fois la décision produite, le nœud 'mission manager' envoie une requête avec les applications à exécuter au nœud ROS contrôleur de configuration' à travers un 'topic'. Le nœud 'contrôleur de configuration' traite la requête du 'mission manager' en lançant l'exécution, sur le FPGA ou le CPU, des applications demandées dans leurs version adaptées (version SW ou version HW). Dans le cas ou le 'contrôleur de configuration' n'arrive pas à lancer l'exécution d'une application avec les performances souhaitées (et donc dans une version dégradée), par manque de ressources, il informe le 'mission manager' par feedback (via un topic) qui produira une nouvelle décision en fonction du feedback et du contexte observés.

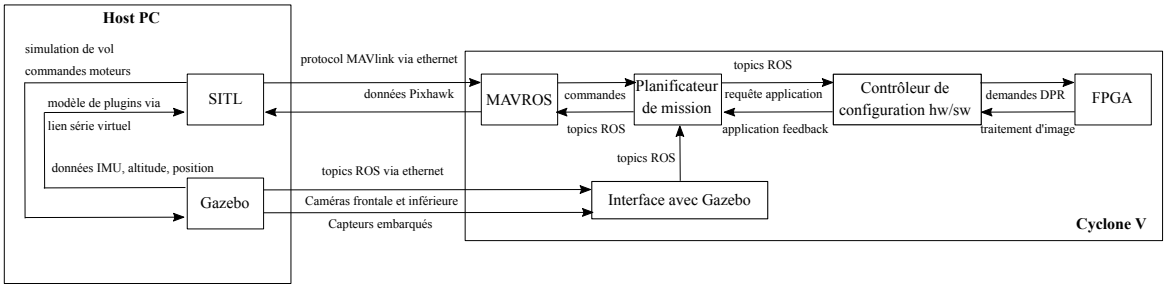


Figure 6.4: *Architecture pour les simulations avec HIL.*

Pour les simulations, le PC hôte est l'interlocuteur du Cyclone V, la communications entre les deux se fait à travers un lien ethernet. Afin d'autoriser les communications avec le monde virtuel crée dans Gazebo, un nœud 'interface Gazebo' est crée. De cette manière, la carte sera capable de récupérer les données des capteurs virtuels depuis Gazebo. Nous savons que Gazebo offre une multitudes de modèles de capteurs prêt à être intégrés au monde virtuel. Par conséquent, les capteurs de détections d'obstacles (infra-rouge, ultrason, LIDAR, etc.), les caméras, IMU, GPS, etc, nécessaire à la mission de tracking, seront utilisés et fourniront des données via les canaux de communications ROS 'topics'. Ces capteurs peuvent être configurés pour s'approcher au plus des caractéristiques des capteurs réels. Les données des capteurs associées à l'auto-pilote PixHawk (IMU, etc.) sont envoyées à SITL et les données des capteurs embarqués (caméras, ultrason, etc.) sont envoyées au Cyclone V via les 'topics' [Kancir et al., 2019]. L'envoi des données via des 'topics' ROS entre deux machines se fait en configurant le même nœud 'master ROS' pour les deux machines. De cette façon, le PC hôte et la carte embarquées sur l'UAV pourront échanger les données à l'aide de ROS.

SITL récupère les données des capteurs associés à la carte PixHawk en utilisant les plugins de Gazebo. Il est important de rappeler que SITL reproduit le comportement de l'auto-pilote ArduCopter embarqué sur la PixHawk. La connexion au Cyclone V se fait avec le lien série MAVLink, cela grâce à MAVROS et à la capacité de SITL à reproduire le protocole de communication MAVLink. L'intérêt d'utiliser SITL est la possibilité de simuler des défaillances au niveau des capteurs et au niveau du drone lui-même dans le but d'évaluer la capacité d'adaptation des programmes embarqués (planificateur de mission (décision), contrôleur de configuration, etc.) sur le drone. Par la suite le même programme peut être exécuté sur la carte HPeC embarquée sur le drone.

6.4.2 Interaction avec le contrôleur de configuration

Le projet HPeC est décomposé en trois niveaux qui s'articulent entre eux afin d'exécuter et gérer au mieux la mission du drone. Ces niveaux sont le planificateur de mission (mission manager), le contrôleur de configuration et l'architecture du

support d'exécution choisi. Dans cette partie, nous allons nous concentrer sur l'interaction entre le niveau mission, planificateur de mission avec le modèle BFM, et le niveau configuration avec le contrôleur de configuration. La Figure 6.5 montre l'architecture ROS mise en place qui permet au planificateur de mission et au contrôleur de configuration de communiquer.

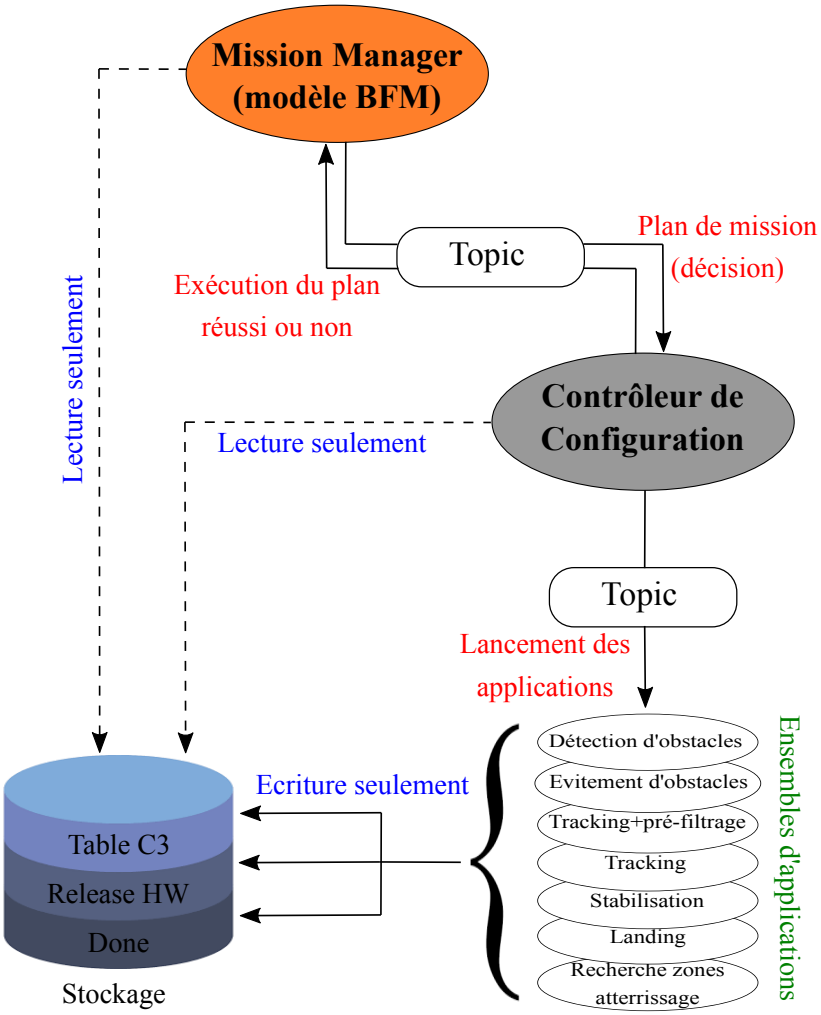


Figure 6.5: *Architecture ROS de l'interaction entre planification de mission et contrôleur de configuration.*

En premier, deux nœuds ROS sont créés, un nœud ROS pour le planificateur de mission 'mission manager' et un second nœud ROS pour le 'contrôleur de configuration'. Les deux nœuds ROS s'échangent des informations à travers les canaux de communications ROS 'topics'. Le nœud 'mission manager' évalue l'état de santé du système et de ses composants, particulièrement la QoS des applications, et produit une décision (ensemble d'applications à exécuter) en fonction du contexte environnemental observé ainsi que des contraintes de ressources, timing et QoS associées à

chaque application embarquée. La décision produite par le modèle BFM du 'mission manager' est envoyée au 'contrôleur de configuration' via un 'topic'. Le 'contrôleur de configuration' envoie l'ordre d'exécuter des applications demandées sur le support d'exécution (CPU ou FPGA). Chaque application est définie à l'aide d'un nœud ROS unique qui pourra pour certaines d'entre elles être exécuté matériellement. Dans ce cas, l'interface reste le nœud ROS qui prend en charge les communications HW/SW.

Les résultats de l'exécution des applications sont stockés dans des tables en mémoire partagée. Nous retrouvons dans cette mémoire partagée une table nommée 'table C3', la métrique 'release HW' et enfin la métrique 'done'. Dans la 'table C3', il seront stockées les informations sur le temps d'exécution et le niveau de QoS nécessaire pour chaque application. La métrique 'release HW' indique si une ou plusieurs tuiles FPGA sont libérées. La métrique 'done' renseigne si le 'contrôleur de configuration' a réussi à exécuter l'application demandée par le 'mission manager'. Ces informations sont accessibles par la 'mission manager' afin de les prendre en compte dans la génération des décisions à suivre. De cette manière le niveau mission et le niveau configuration collaborent pour gérer et mener à bien la mission du drone.

6.4.3 Validation pour la mise en place d'un prototype

Dans les sections précédentes, nous avons abordé la méthodologie et l'architecture mis en place afin de valider en simulation le module de planification de mission. Ces simulations sont faite à l'aide d'outils open-source comme ROS, HIL et SITL. L'objectif final du projet HPeC est de démontrer l'auto-adaptabilité des système adaptatif en mettant en avant l'aspect de configuration dynamique du plan de mission décidé par le planificateur de mission. Pour mettre en place un démonstrateur. Une carte embarquée a été conçue dans le cadre du projet HPeC, sur laquelle Linux et ROS seront installés pour permettre de simuler et d'exécuter les différents scénarios de la mission tracking définie dans HPeC. De cette manière, le prototype HPeC pour le contrôle d'une mission de véhicule autonome sera validé.

Validation avec Hardware In The Loop

La Figure 6.6 présente le résultat type d'une simulation de scénario effectué avec HIL. La figure représente la simulation d'un scénario de suivi d'une cible dans un contexte où le planificateur de mission (modèle BFM) demande l'exécution de l'application 'recherche zone atterrissage d'urgence' en plus du 'traking'. La partie C de la figure montre l'environnement virtuel 3D dans Gazebo avec le drone qui survole une zone avec différentes textures. Des rayons bleu émanent du drone, cela correspond à la portée des capteurs de détection d'obstacles (ultrason, infra-rouge, laser, etc). La partie B (de la même figure), représente l'image acquise par la caméra orientée vers le bas du drone. La partie A représente la même image acquise (partie B) après traitement par l'application. Enfin, la partie D correspond au terminal avec les résultats de l'exécution de l'application. Le terminal affiche les temps d'exécution

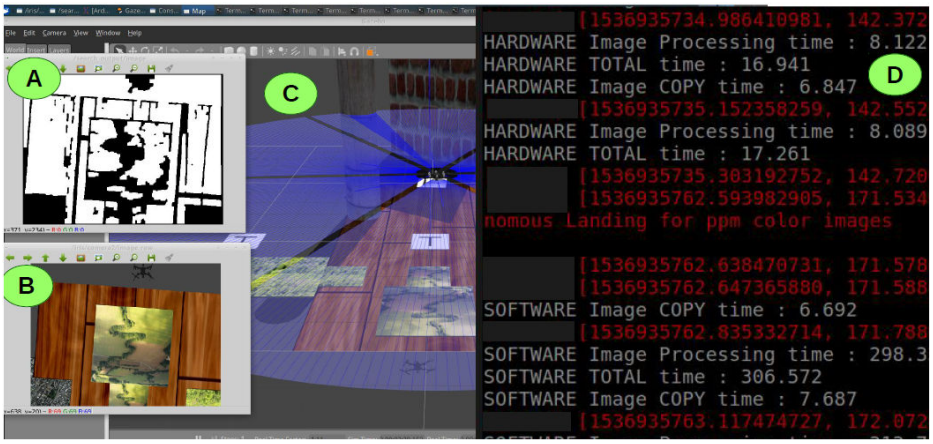


Figure 6.6: *Résultat de simulation avec ROS et HIL*

de l'application en software et en hardware. Pour l'application 'recherche zone atterrissage d'urgence', le temps d'exécution en software est d'environ 300 ms contre 8 ms dans sa version hardware. Ce qui génère une accélération de 26, intéressant pour décharger le CPU lorsque ce dernier est surchargé. De même, pour l'application 'tracking' (TLD [Kalal et al., 2012]) avec un temps d'exécution de 285 ms en software contre 7 ms pour une exécution en hardware. L'utilisation de HIL permet, ainsi, la validation de plusieurs aspects du projet HPeC comme la validation du modèle BFM pour la mission HPeC et l'exécution des plans de mission produits en fonction des scénarios déroulés ; aussi la validation de la partie re-configuration dynamique du plan de mission par le contrôleur de configuration. L'ensemble des outils utilisés, en particulier ROS tournera sur la carte HPeC Cyclone V sur laquelle est installé un Linux (ubuntu 16.04). Les Figures 6.7 et 6.8 montrent la carte conçue dans le cadre du projet HPeC.

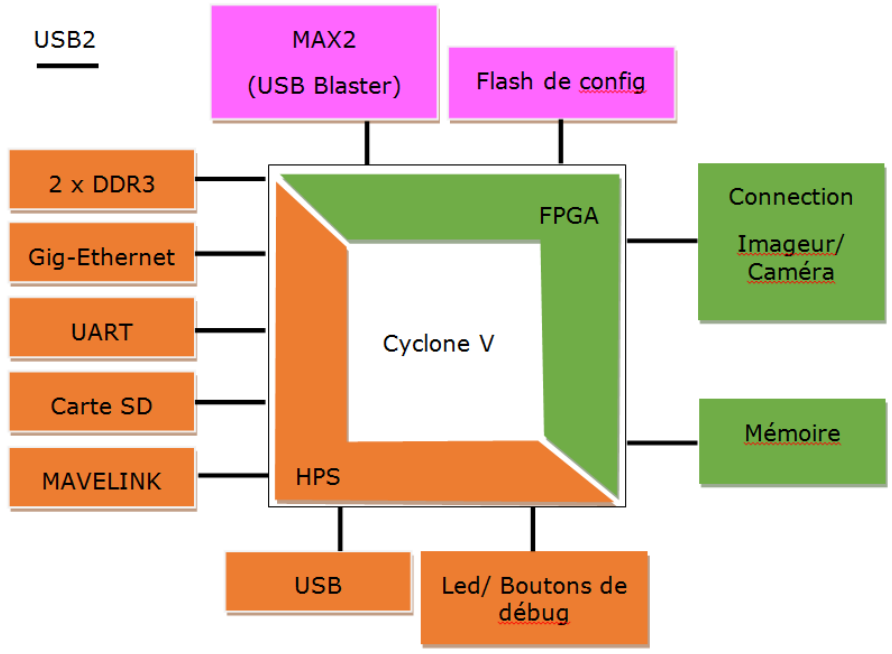


Figure 6.7: *Schéma de la carte HPeC conçue.*

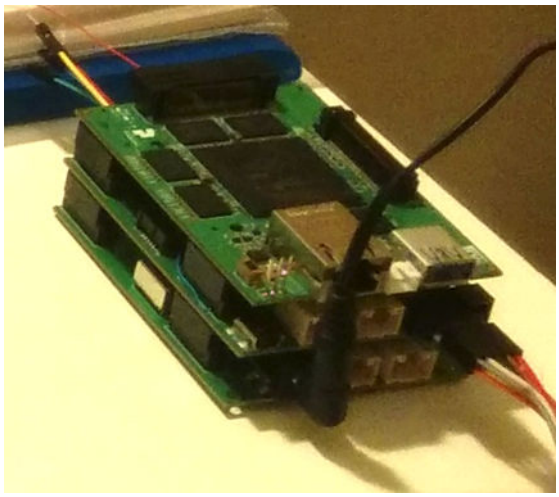


Figure 6.8: *Carte HPeC finale.*

6.5 Conclusion

Nous avons abordé, dans ce chapitre, l'intégration du modèle BFM dans le projet HPeC. Nous avons, dans un premier temps, présenté le projet HPeC, les différents niveaux le constituant et ses objectifs. La validation du modèle BFM, qui constitue le planificateur de mission du projet, a été faite sur une mission de tracking définie dans le cadre de HPeC. Différents scénarios ont été décrits pour montrer le rôle du planificateur de mission BFM au sein du projet HPeC.

Afin d'effectuer des simulations réalistes des scénarios de la mission de tracking HPeC, une architecture a été proposée en utilisant plusieurs outils. ROS pour les programmes et applications nécessaire au projet (planificateur de mission contrôleur de configuration, application de tracking, etc) : Gazebo pour créer le monde virtuel en 3D de la mission ainsi que le modèle du drone utilisé pour la mission, SITL pour reproduire et simuler le comportement de l'auto-pilote et enfin HIL pour faire des simulations et exécutions des applications sur la carte embarquée.

Une partie de ce chapitre (validation du modèle BFM avec les scénarios HPeC) a été publié dans un journal [Hireche et al., 2018b]. D'autres publications sur l'intégration des différents modules du projet HPeC sont prévues.

- Chapitre 7 -

Conclusion générale

Sommaire

7.1 Conclusion

Avec les travaux réalisés et présentés dans cette thèse, nous avons atteint principalement trois objectifs. Le premier est de proposer un planificateur de mission pour systèmes autonomes qui soit compact et générique dans le but de décider du plan de la mission en fonction des événements aléatoires qui peuvent apparaître durant la mission. Le deuxième objectif est de proposer un modèle de diagnostic pour l'évaluation de l'état de santé du système autonome, des capteurs ainsi que des applications embarqués à l'aide de réseaux Bayésiens. Ensuite vient la prise de décision en fonction du diagnostic établi et des événements aléatoires observés. La description de la mission et la prise de décision sont faites à l'aide des Markov Decision Process. Enfin, le troisième objectif est de proposer une implémentation logicielle et matérielle embarqué du MDP pour la décision pour faciliter son embarquabilité sur systèmes autonomes.

Notre modèle de planification de mission (BFM modèle) permet d'évaluer l'état de la mission et du système autonome en faisant appel aux blocs diagnostic du planificateur. Il permet de produire la décision sur le plan de mission à exécuter à l'aide cette fois du bloc décision de notre planificateur. Le bloc diagnostic repose sur l'utilisation de BNs, ces BNs sont construits à partir d'une table d'analyse FMEA. Le diagnostic et l'apprentissage des paramètres du BN sont calculés par inférence en utilisant la compilation AC plus simple pour être embarqué. Le bloc de décision s'appuie sur les MDPs qui nous permettent de décrire la mission et ensuite de produire la décision. La particularité de notre modèle BFM est sa facilité d'utilisation par un non expert car les modèles BN et MDP peuvent être générés facilement à partir de la table FMEA (pour les BNs) et la table de spécification de la mission (pour les MDPs). Aussi, le BFM modèle est *scalable* (passage à l'echelle) car nous proposons deux versions possibles, monolithique et concurrente, pour décrire la mission.

Pour l'implémentation du planificateur de mission, nous nous sommes concentrés sur la partie décision en commençant par étudier les différents algorithmes classiquement utilisés pour la résolution de MDP afin de choisir l'algorithme qui s'adapte le mieux à notre contexte. Ensuite, nous proposons une implémentation SW/HW pour l'embarqué du *solver* MDP choisi. Cet algorithme est intégré aux outils de synthèse haut niveau (HLS) pour adapter la solution à nos besoins en définissant les directives de synthèse haut niveau (HLS) nécessaires. La HLS permet de générer le HDL (VHDL) qui sera synthétisé par la suite sur le FPGA. Le déploiement de l'architecture se fait avec Vivado Design Suite, enfin l'exécution sur la carte ZedBoard Xilinx est réalisé avec l'outil SDK de vivado.

Cette thèse fait partie du projet HPeC qui propose de démontrer la capacité auto-adaptative des systèmes embarqués au sein des systèmes autonomes face aux contraintes du système et aux perturbations environnementales. La validation des différents aspects de notre modèle BFM est faite sur la mission de tracking d'un drone du projet HPeC. Différents scénarios de mission on été définis dans le cadre de HPeC

que nous avons utilisé pour valider les décisions produites par le modèle BFM. Le second point important de la validation est l'intégration du planificateur de mission à l'architecture du projet HPeC et l'interaction avec le contrôleur de configuration pour l'exécution du plan de mission généré par le planificateur. L'architecture HPeC est réalisée à l'aide de ROS. La validation de la mission HPeC est réalisé à l'aide du simulateur Hardware In The Loop et GAZEBO dans lequel l'environnement 3D de la mission est construit.

7.2 Perspectives

Le travail effectué connaît encore un certain nombre de limitations. Nous donnons quelques idées pour améliorer et étendre l'approche proposée.

1. Au niveau de la planification de mission, des améliorations sont possibles sur les points suivants :
   - Amélioration au niveau du modèle BFM : proposer une méthode pour mieux définir les valeurs de *rewards* des différentes actions exécutées durant la mission. Cette amélioration permettra d'obtenir des décisions plus proches de la réalité dans n'importe quel contexte.
   - Améliorer le framework proposé pour la description de missions en intégrant la spécification de BFMs concurrents dans le but de faciliter cette tâche à un utilisateur non expert.
   - Formalisation de la gestion de conflits entre MDPs, proposée dans cette thèse, pour passer à l'échelle et intégrer à l'environnement de programmation.
   - Étendre l'application du modèle BFM proposé à un contexte de mission réalisé par un essaim de drones. Cela permettra un travail collaboratif et distribué entre les drones de l'essaim pour mieux gérer la mission. Concrètement, chaque drone pourra prendre une décision à l'aide de son propre modèle BFM en fonction du contexte, de son état interne et aussi de la situation de ses drones voisins.
   - Nous avons traité l'aspect 'safety' de la mission à l'aide de BN diagnostic. Il est possible de compléter ce travail en incluant l'aspect sécurité de la mission en considérant par exemple l'évaluation de la qualité des communications entre le drone et la base ou entre différents drones dans le cas d'un essaim. Une thèse vient d'être démarrée dans l'équipe pour traiter ce point.
   - Développer un outil de CAO afin que l'expert puisse produire le code du planificateur de mission sans avoir à connaître les détails de l'implémentation. La méthode choisie le permet mais l'environnement de programmation haut niveau reste à concevoir.

2. Au niveau implémentation embarquée, des améliorations sont aussi envisageables :
   - Intégrer l'apprentissage des paramètres de BNs dans la version embarquée du modèle BFM dans le but d'affiner les probabilités initiales du BN. Dans un premier temps, il sera nécessaire de valider l'apprentissage de paramètres de BN en utilisant des bases de données réalistes issues par exemple de différents historiques de vol. Dans un second temps, embarquer la méthode d'apprentissage sur le système embarqué du drone dans le but d'améliorer les valeurs de paramètres des BNs et obtenir un diagnostic robuste au fur et à mesure de l'évolution de la mission.
   - Nous avons proposé une implémentation SW/HW de la prise de décision à l'aide de MDPs. Il est possible d'améliorer cette version en essayant d'autres optimisations telles que l'utilisation d'une représentation en virgule fixe pour le type de données.
   - Proposer d'autres implémentations embarquées pour le modèle BFM pour un déploiement sur d'autres types de systèmes embarqués tels que les GPU.
3. Perspectives à long termes :
   - Coopération avec le cloud : proposer des méthodes de coopération entre le cloud et notre modèle BFM. L'idée est de migrer le calcul de la décision dans le cloud. Par conséquent, chaque fois que le modèle BFM détecte un événement, une commande est envoyée au cloud pour calculer une nouvelle décision à l'aide du *solver* MDP. Cependant cette coopération est envisageable seulement dans le cas où la contrainte de temps de réponse le permet et que la communication avec le cloud est possible. Une solution intermédiaire consisterait à utiliser le cloud lorsque la contrainte de temps de réponse le permet. Dans le cas où cette contrainte est critique, le calcul se fera au sein du drone avec la version matérielle embarquable accélérée de la prise de décision. Dans ce cas, nous nous retrouvons dans le cadre d'un problème d'optimisation de type Edge Computing.
   - Le modèle BFM que nous avons proposé pour la planification de mission est générique avec des modèles de décision et de diagnostic répandus dans d'autres domaines. Notre modèle peut donc être utilisé dans d'autres domaines que la robotique comme l'industrie (avec les chaînes de production), la finance, etc.

Liste des figures

Liste des tableaux

Glossaire

AC	Circuit arithmétique (Arithmetic Circuit)
BFM	modèle Bayésiens à partir de l'analyse FMEA pour alimenter le MDP
BN	réseaux Bayésiens (Bayesian Networks)
EM	Algorithme Expectation et Maximisation
FMEA	Analyse de Mode de Défaillances et leurs Effets (Failure Mode and Effects Analysis)
HIL	Hardware In the Loop
JT	Arbre de jonction (Jonction Tree)
MDP	Processus de Décision Markovien (Markov Decision Process)
PI	Algorithme Policy Iteration
POMDP	Processus de Décision Markovien Partiellement Observable
ROS	Robotic Operating System
SITL	Software In The Loop
VI	Algorithme Value Iteration

Bibliographie

Filippo Amato, Alberto López, Eladia María Peña-Méndez, Petr Vaňhara, Aleš Hampl, and Josef Havel. Artificial neural networks in medical diagnosis, 2013. 13

D.F. Bacon, S.L. Graham, and O.J. Sharp. Compiler transformations for high-performance computing. *ACM Comput. Surv.*, 26(4) :345–420, December 1994. ISSN 0360-0300. doi : 10.1145/197405.197406. URL http://doi.acm.org/10.1145/197405.197406. 42

Nicole Bäuerle and Ulrich Rieder. *Markov decision processes with applications to finance*. Springer Science & Business Media, 2011. 28

Ann Becker, Dan Geiger, and Alejandro A Schäffer. Automatic selection ofloop breakers for geneticlinkage analysis. *Human heredity*, 48(1) :49–60, 1998. 14

Brett Bethke, Luca Bertuccelli, and Jonathan How. Experimental demonstration of adaptive mdp-based planning with model uncertainty. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 6322, 2008. 39, 74

Andrea Bobbio, Luigi Portinale, Michele Minichino, and Ester Ciancamerla. Improving the analysis of dependable systems by mapping fault trees into bayesian networks. *Reliability Engineering & System Safety*, 71(3) :249–260, 2001. 13

Christos G Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer Science & Business Media, 2009. 5, 27

Caroline P Carvalho Chanel, Florent Teichteil-Königsbuch, and Charles Lesire. Multi-target detection and recognition by uavs using online pomdps. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013. 27

J. Cong and J. Xu. Simultaneous fu and register binding based on network flow method. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08, pages 1057–1062, New York, NY, USA, 2008. ACM. ISBN

978-3-9810801-3-1. doi : 10.1145/1403375.1403629. URL http://doi.acm.org/10.1145/1403375.1403629. 42

Elva Corona-Xelhuantzi, Eduardo F Morales, and Enrique Sucar. Executing concurrent actions with multiple markov decision processes. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 82–89. IEEE, 2009. 81

P. Coussy and A. Morawiec. *High-Level Synthesis : From Algorithm to Digital Circuit*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 1402085877, 9781402085871. 41

A. Darwiche. A differential approach to inference in Bayesian networks. *J. ACM*, 50(3) :280–305, 2003. 22, 23, 25

Gwenaül Delaval, Soguy Mak-Karé Gueye, and Éric Rutten. Distributed execution of modular discrete controllers for data center management. *IFAC-PapersOnLine*, 48(7) :139–146, 2015. 126

Kazuo J Ezawa and Til Schuermann. Fraud/uncollectible debt detection using a bayesian network based learning system : A rare binary outcome with mixed data structures. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 157–166. Morgan Kaufmann Publishers Inc., 1995. 14

N. Friedman. The bayesian structural em algorithm. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 129–138, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-555-X. URL http://dl.acm.org/citation.cfm?id=2074094.2074110. 25

N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian networks to analyze expression data. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, RECOMB '00, pages 127–135, New York, NY, USA, 2000. ACM. ISBN 1-58113-186-0. doi : 10.1145/332306.332355. URL http://doi.acm.org/10.1145/332306.332355. 14

Blake Fuller, Jonathan Kok, Neil A Kelson, and Luis F Gonzalez. Hardware design and implementation of a mavlink interface for an fpga-based autonomous uav flight control system. In *Proceedings of the 2014 Australasian Conference on Robotics and Automation*, pages 1–6. Australian Robotics & Automation Association ARAA, 2014. 39

Zoubin Ghahramani. Learning dynamic bayesian networks. In *International School on Neural Networks, Initiated by IIASS and EMFCSC*, pages 168–197. Springer, 1997. 81

Malay Ghosh. The bayesian choice : A decision-theoretic motivation. *Journal of the American Statistical Association*, 91(433) :431–433, 1996. 26

D. Grossman and P. Domingos. Learning bayesian network classifiers by maximizing conditional likelihood. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 46–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi : 10.1145/1015330.1015339. URL http://doi.acm.org/10.1145/1015330.1015339. 25

Soguy Mak-Karé Gueye, Éric Rutten, and Jean-Philippe Diguet. Autonomic management of missions and reconfigurations in fpga-based embedded system. In *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 48–55. IEEE, 2017. 124, 126

Peter E Hart and Jamey Graham. Query-free information retrieval. *IEEE Expert*, 12(5) :32–37, 1997. 14

M. Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In *UAI '86 : Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence, University of Pennsylvania, Philadelphia, PA, USA, August 8-10, 1986*, pages 149–164, 1986. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1776&proceeding_id=1002. 19

Chabha Hireche, Catherine Dezan, and Jean-Philippe Diguet. Online diagnosis updates for embedded health management. In *2017 6th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–5. IEEE, 2017. 9, 69, 103

Chabha Hireche, Catherine Dezan, Jean-Philippe Diguet, and Luis Mejias. Bfm : a scalable and resource-aware method for adaptive mission planning of uavs. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6702–6707. IEEE, 2018a. 9, 69

Chabha Hireche, Catherine Dezan, Stéphane Mocanu, Dominique Heller, and Jean-Philippe Diguet. Context/resource-aware mission planning based on bns and concurrent mdps for autonomous uavs. *Sensors*, 18(12) :4266, 2018b. 9, 96, 145

Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. Spudd : Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 279–288. Morgan Kaufmann Publishers Inc., 1999. 81

Ronald A Howard. Comments on the origin and application of markov decision processes. *Operations Research*, 50(1) :100–102, 2002. 28

A. Huang, C.and Darwiche. Inference in belief networks : A procedural guide. *International Journal of Approximate Reasoning*, 15 :225–263, 1996. 21

F.V. Jensen and T.D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, 2nd edition, 2007. ISBN 0387682813. 15, 19

Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7) :1409–1422, July 2012. ISSN 0162-8828. doi : 10.1109/TPAMI.2011.239. 57, 131, 143

Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7) :1409–1422, 2011. 129

Pierre Kancir, Jean-Philippe Diguet, and Marc Sevaux. Development of tools for multi vehicles simulation with robot operating system and ardupilot. 2019. 140

Y.G. Kim and M. Valtorta. On the detection of conflicts in diagnostic bayesian networks using abstraction. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 362–367, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-385-9. URL http://dl.acm.org/citation.cfm?id=2074158.2074199. 14

U. Kjærulff. Triangulation of graphs – algorithms giving small total state space. Technical report, 1990. 20

Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004. 138

Jonathan Kok, Luis Felipe Gonzalez, and Neil Kelson. Fpga implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning. *IEEE transactions on evolutionary computation*, 17(2) :272–281, 2012. 39

Andrey Kolobov. Planning with markov decision processes : An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1) :1–210, 2012. 28

MH Korayem and A Iravani. Improvement of 3p and 6r mechanical robots reliability and quality applying fmea and qfd approaches. *Robotics and Computer-Integrated Manufacturing*, 24(3) :472–487, 2008. 13, 46

Pierre Laroche, Yann Boniface, and René Schott. A new decomposition technique for solving markov decision processes. In *Proceedings of the 2001 ACM symposium on applied computing*, pages 12–16. ACM, 2001. 81

S. L. Lauritzen and D. J. Spiegelhalter. Readings in uncertain reasoning. chapter Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1-55860-125-2. URL http://dl.acm.org/citation.cfm?id=84628.85343. 19

S.T Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87 :1098–1108, 1992. 19

Steffen L Lauritzen. The em algorithm for graphical association models with missing data. *Computational Statistics & Data Analysis*, 19(2) :191–201, 1995. 26

Bo Li, M-Y Chow, Yodyium Tipsuwan, and James C Hung. Neural-network-based motor rolling bearing fault diagnosis. *IEEE transactions on industrial electronics*, 47(5) :1060–1069, 2000. 13

B. Liu and X. Wu. Mission reliability analysis of missile defense system based on dodaf and bayesian networks. In *2011 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering*, pages 777–780, June 2011. doi : 10.1109/ICQR2MSE.2011.5976725. 14

O.J. Mengshoel, D.C. Wilkins, and .D Roth. Initialization and restart in stochastic local search : Computing a most probable explanation in bayesian networks. *IEEE Transactions on Knowledge and Data Engineering*, 23(2) :235–247, 2011. URL http://cogcomp.org/papers/MengshoelWiRo11.pdf. 19

Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas L Dean, and Craig Boutilier. Solving very large weakly coupled markov decision processes. In *AAAI/IAAI*, pages 165–172, 1998. 81

P. Naïm, P.H. Wuillemin, P. Leray, O. Pourret, and A. Becker. *Réseaux bayésiens*. Algorithmes. Eyrolles, 2007. URL https://hal.archives-ouvertes.fr/hal-00412267. 13, 14, 15, 17, 21, 25, 26, 63

J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7. 13

J. Pearl and S. Russell. *Bayesian networks*. Computer Science Department, University of California, 1998. 5

Caroline Ponzoni Carvalho Chanel, Florent Teichteil-Königsbuch, and Charles Lesire. Multi-target detection and recognition by uavs using online pomdps. 2013. 33

Martin L Puterman. *Markov Decision Processes. : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014. 27

Rajesh PN Rao. Decision making under uncertainty : a neural model based on partially observable markov decision processes. *Frontiers in computational neuroscience*, 4 :146, 2010. 28

Joh Damon Reese and Nancy G Leveson. Software deviation analysis : A "safeware" technique. In *AIChe 31st Annual Loss Prevention Symposium*. Citeseer, 1997. 13

Juan Jesús Roldán, Jaime del Cerro, and Antonio Barrientos. A proposal of methodology for multi-uav mission modeling. In *2015 23rd Mediterranean Conference on Control and Automation (MED)*, pages 1–7. IEEE, 2015. 27

J. Rose, R. J. Francis, D. Lewis, and P. Chow. Architecture of field-programmable gate arrays : the effect of logic block functionality on area efficiency. *IEEE Journal of Solid-State Circuits*, 25(5) :1217–1225, Oct 1990. ISSN 0018-9200. doi : 10.1109/4.62145. 40

Stuart J Russell and Peter Norvig. *Artificial intelligence : a modern approach*. Malaysia ; Pearson Education Limited,, 2016. 31, 106

Scott Sanner. Relational dynamic influence diagram language (rddl) : Language description. *Unpublished ms. Australian National University*, 32, 2010. 27

Olivier Sigaud and Olivier Buffet. Processus décisionnels de markov en intelligence artificielle, 2008. 34

C. Skaanning. A knowledge acquisition tool for bayesian-network troubleshooters. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, pages 549–557, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-709-9. URL http://dl.acm.org/citation.cfm?id=2073946.2074010. 14

Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3) :165–188, 2005. 28

Serge Romaric Tembo, Sandrine Vaton, Jean-Luc Courant, and Stéphane Gosselin. A tutorial on the em algorithm for bayesian networks : application to self-diagnosis of gpon-ftth networks. In *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 369–376. IEEE, 2016. 26

Massimo Tipaldi and Luigi Glielmo. A survey on model-based mission planning and execution for autonomous spacecraft. *IEEE Systems Journal*, 12(4) :3893–3905, 2017. 27

Hoa Van Nguyen, Michael Chesser, Lian Pin Koh, S Hamid Rezatofighi, and Damith C Ranasinghe. Trackerbots : Autonomous uav for real-time localization and tracking of multiple radio-tagged animals. *arXiv preprint arXiv :1712.01491*, 2017. 33

Fernando Vanegas, Duncan Campbell, Nicholas Roy, Kevin J Gaston, and Felipe Gonzalez. Uav tracking and following a ground target under motion and localisation uncertainty. In *2017 IEEE Aerospace Conference*, pages 1–10. IEEE, 2017. 27, 33, 72, 96

S. Zermani, C. Dezan, C. Hireche, R. Euler, and J. P. Diguet. Embedded and probabilistic health management for the gps of autonomous vehicles. In *2016 5th Mediterranean Conference on Embedded Computing (MECO)*, pages 401–404, June 2016. doi : 10.1109/MECO.2016.7525792. 9

S. Zermani, C. Dezan, C. Hireche, R. Euler, and J.P. Diguet. Embedded context aware diagnosis for a UAV soc platform. *Microprocessors and Microsystems - Embedded Hardware Design*, 51 :185–197, 2017a. doi : 10.1016/j.micpro.2017.04.013. URL https://doi.org/10.1016/j.micpro.2017.04.013. 9

Sara Zermani. *Implémentation sur SoC des réseaux Bayésiens pour l'état de santé et la décision dans le cadre de missions de véhicules autonomes*. PhD thesis, Université de Bretagne occidentale-Brest, 2017. 62

Sara Zermani, Catherine Dezan, Chabha Hireche, Reinhardt Euler, and Jean-Philippe Diguet. Embedded context aware diagnosis for a uav soc platform. *Microprocessors and Microsystems*, 51 :185–197, 2017b. 5, 25, 59, 73, 100, 103

Titre : titre (en français) Etude et implémentation sur SoC-FPGA d'une méthode probabiliste pour le contrôle de mission de véhicule autonome.

Mots clés : Planification de mission, Markov Decision Process, réseaux Bayésiens, implémentation matérielle/logicielle, FPGA.

Résumé : Les systèmes autonomes embarquent différents types de capteurs, d'applications et de calculateurs puissants. Ils sont donc utilisés dans différents domaines d'application et réalisent diverses missions simples ou complexes. Ces missions se déroulent souvent dans des environnements non déterministes avec la présence d'évènements aléatoires pouvant perturber le déroulement de la mission. Il est donc nécessaire d'évaluer régulièrement l'état de santé du système et de ses composants matériels et logiciels dans le but de détecter les défaillances à l'aide de réseaux Bayésiens. Par la suite, une décision est prise par le planificateur de mission en générant un nouveau plan de mission assurant la continuité de la mission en réponse à l'événement détecté. Cette décision est prise à l'aide du modèle Markov Decision Process en fonction de contraintes telles que l'objectif de la mission, l'état de santé des capteurs et des applications embarqués, la stratégie de réalisation de la mission 'stratégie safety' ou 'stratégie mission first', etc. Comme les systèmes autonomes exécutent différentes tâches qui demandent différentes performances, il est nécessaire de penser à l'utilisation d'accélérateurs matériels sur SoC-FPGA dans le but de répondre aux contraintes de calculs hautes performances et décharger le CPU si besoin.

Title : titre (en anglais) Study and implementation on SoC-FPGA of a probabilistic method for mission planning in autonomous vehicle.

Keywords : Mission planning, Markov Decision Process, Bayesian Networks, Hardware/Software implementation, FPGA.

Abstract: Autonomous systems embed different types of sensors, applications and powerful calculators. Thus, they are used in different fields of application and perform various simple or complex tasks. Generally, these missions are executed in non-deterministic environments with the presence of random events that can affect the mission's progress. Therefore, it is necessary to regularly assess the health of the system and its hardware and software components in order to detect failures using Bayesian Networks. Subsequently, a decision is made by the mission planner by generating a new mission plan that ensures the mission in response to the detected event. This decision is made using the Markov Decision Process model based on constraints such as the mission objective, the health status of sensors and embedded applications, the mission policy "safety policy" or "mission first policy", etc. As autonomous systems perform different tasks that require different performance, it is necessary to consider the use of hardware accelerators on SoC-FPGA in order to meet high-performance computing constraints and unload the CPU if needed.