Proposition d'une architecture de contrôle adaptative pour la tolérance aux fautes

Bastien Durand

ACADÉMIE DE MONTPELLIER

UNIVERSITÉ MONTPELLIER II

- SCIENCES ET TECHNIQUES DU LANGUEDOC -

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ MONTPELLIER II

Discipline : Génie Informatique, Automatique et Traitement du Signal
Formation Doctorale : Systèmes Automatiques et Microélectronique
École Doctorale : Information, Structures et Systèmes

présentée et soutenue publiquement

par

Bastien DURAND

le 15 Juin 2011

Titre :

Proposition d'une architecture de contrôle adaptative pour la tolérance aux fautes

JURY :

M. LACROIX Simon	Directeur de Recherche	LAAS-CNRS	Rapporteur
M. THIRIET Jean-Marc	Professeur	GIPSA-Lab	Rapporteur
M. LAPIERRE Lionel	Maître de Conférences	LIRMM	Examinateur
M. MALENFANT Jacques	Professeur	LIP6	Examinateur
M. CRESTANI Didier	Professeur	LIRMM	Directeur de thèse
Mme. GODARY-DEJEAN Karen	Maître de Conférences	LIRMM	Co-directeur de thèse

A Marie , Michel, Maguy et Thibault,
mais aussi aux autres à qui j'ai été fier de présenter mon travail,
et à tous ceux à qui j'aurais aimé le présenter.

Il a du bobo Léon
Il va peut-être canner Léon
Il a du bobo Léon
Oh, pauvre Léon

On l'a mené à l'hôpital
Pour le soigner où il avait mal
Il s'était fait mal dans la rue
Mais on l'a soigné autre part

Et il est mort ...

Extrait de la chanson « *Bobo Léon* » de **Boby Lapointe**.

Remerciements

Les travaux présentés dans ce mémoire ont été réalisés au *Laboratoire d'Informatique Robotique Microélectronique de Montpellier* dans le département Robotique et plus particulièrement dans l'équipe NERO. Ils ont été dirigés par Monsieur Didier CRESTANI et Madame Karen GODARY-DEJEAN que je remercie infiniment tant pour les contacts humains que nous avons liés que pour leur aide tout au long de cette expérience. mais aussi en collaboration avec de nombreux collègues du laboratoire que je souhaite tout autant remercier :

- Les roboticiens Lionel LAPIERRE, René ZAPATA et les thésards Lei ZHANG et Alfredo TORIZ PALACIOS qui m'ont permis d'utiliser le Pioneer 3-DX mais surtout qui m'ont offert leurs connaissances de la robotique mobile tant par les discussions et conseils, que par les algorithmes de contrôle utilisés pour mes expérimentations.
- Les experts en architecture de contrôle David ANDREU, Robin PASSAMA et Abdellah ELJALAOUI qui m'ont permis de comprendre, utiliser et maintenir l'architecture COTAMA.

Je souhaite aussi fortement remercier les membres extérieurs de mon jury présidé par Monsieur Jacques MALENFANT professeur de l'université Pierre et Marie Curie (Paris), Messieurs Jean-Marc THIRIET, professeur à l'Université Joseph Fourier (Grenoble) et Simon LACROIX, directeur de recherche au LAAS-CNRS (Toulouse) pour le temps passé à lire mon manuscrit et pour leurs commentaires précieux.

Je souhaite aussi saluer l'ensemble de la communauté CAR (conférence française Control Architecture of Robots) avec qui j'ai pu partager, apprendre et évoluer sur les concepts et développement des architectures de contrôle et des différents domaines connexes, mais aussi sur les bons moments passés ensemble lors de ces rencontres annuelles. Mon salut s'adresse tout particulièrement à Messieurs J. MALENFANT, D. ANDREU, N. BOURAQADI, J-L. FARGES, C. NOVALES, et G. MOURIOUX.

Je ne peux passer sous silence l'ensemble des personnes qui m'ont accompagné pendant mes 4 années de thèse, les doctorants aujourd'hui docteurs :

– les vieux Yousra, Abdellah, Olivier, Kevin, Aurélien,
– les plus jeunes Sébastien, Vincent, Jérémy, Guillaume, Rogerio, Carla, Lei, Bruno,
– et ceux avec qui j'ai entammé cette longue aventure d'un peu plus de 3 ans, Sébastien, Xianbo, Nicolas, Antonio, et Michael (aussi dans la liste suivante)

mais aussi ceux qui j'espère le seront bientôt Qin, Nicolas, Arnaud.

Pour finir, je me tourne vers ma famille, mes amis et l'ensemble de mes proches pour leur dire à quel point leur présence et leur soutiens m'ont été nécessaire pour la réussite de mon aventure.

Durand Bastien

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

Introduction Générale

Nous sommes à la fin des années 90 au Fort Léonard Woods, un camp d'expérimentation de l'armée américaine où l'on évalue les performances des dernières avancées technologiques du moment. Aujourd'hui le test porte sur un tank de près de 30 tonnes auquel a été ajouté un dispositif de déminage, le PANTHER II. On y a embarqué l'infrastructure logicielle nécessaire pour qu'il puisse être téléopéré sur des terrains de structure et de composition variées, dans les conditions climatiques les plus difficiles. Malheureusement le système de contrôle se révèle fréquemment instable et cela se traduit par un comportement erratique pouvant aller jusqu'à la perte de contrôle totale du blindé.

Nous sommes maintenant à Manhattan quelques heures après le terrible attentat du 11 septembre 2001. Au milieu des ruines de l'effondrement des tours du World Trade Center, un petit robot de type SOLEM© tente de se frayer un passage au milieu de décombres en tout genre et de la poussière omniprésente à la recherche d'hypothétiques survivants. Il est accompagné en permanence par deux opérateurs. L'un assure un contrôle distant sans fil, l'autre dispose d'un câble permettant si c'est est possible, d'assister le robot lorsqu'il se retrouve dans des positions délicates. Malgré cette assistance, en raison des distorsions induites par la structure métallique du gratte-ciel, la communication, qui devait être assurée jusqu'à plus d'un kilomètre, a été coupée au bout de seulement quelques mètres. Le robot, qui n'a pu revenir à sa base, a été définitivement perdu.

Ces exemples réels interpellent et démontrent que contrairement aux idées reçues, les robots restent encore des dispositifs complexes et fragiles dont le comportement devient rapidement instable et non déterministe en présence de dysfonctionnements internes ou de perturbations externes. S'ils peuvent remplir leur mission avec une performance acceptable dans des conditions nominales, ils se révèlent très peu tolérants aux fautes et la plupart du temps incapables d'adapter dynamiquement leur comportement en fonction des évènements perturbateurs rencontrés et des ressources opérationnelles dont ils disposent. Force est de constater que cette capacité à la tolérance aux fautes, si naturelle et essentielle dans le monde du vivant, est le plus souvent insuffisante dans les applications comme la robotique mobile autonome. Elle s'inscrit indiscutablement comme un des défis à relever dans les prochaines décades pour concevoir des robots plus fiables adaptées à des applications toujours plus critiques comme par exemple les robots compagnons qui nous accompagnerons dans nos tâches quotidiennes, ou des robots efficaces dans des missions de type « save and rescue » ou d'intervention sur des sites nucléaires. De

toute évidence l'architecture de contrôle, en raison de sa dimension éminemment décisionnelle, de sa capacité d'adaptation et de sa position stratégique au sein de la boucle de commande, sera l'organe central au sein duquel les mécanismes de tolérance aux fautes devront être déployés.

Le développement d'une architecture de contrôle tolérante aux fautes pour la robotique mobile constitue donc la problématique centrale de ce document qui s'organise selon cinq parties.

La première partie permet d'aborder les principes de sûreté de fonctionnement en se focalisant particulièrement sur les techniques relevant de la tolérance aux fautes. Après un positionnement général, le discours se concentre sur les approches relevant d'une démarche automaticienne avant de se polariser plus particulièrement sur les mécanismes, relevant de cette problématique, mis en place au sein des architectures de contrôle robotiques. Pour finir, une analyse globale permet d'établir les limitations et insuffisances des approches actuelles avant d'identifier un ensemble de principes devant être satisfaits pour concevoir et développer une architecture de contrôle tolérante aux fautes.

La seconde partie constitue la clef de voûte du travail présenté dans ce document. Il présente la démarche de conception proposée qui se décline en plusieurs phases successives. La première consiste à identifier au préalable, pour un système robotisé donné et dans le cadre d'une mission particulière, les fautes pertinentes devant être prises en compte. Les suivantes permettent de doter l'architecture de mécanismes de détection, de diagnostic et de recouvrement adaptés à la criticité des défaillances rencontrées et au contexte du robot et de la mission.

La troisième partie présente le contexte applicatif et expérimental de l'étude et dans lequel sera évalué la démarche proposée au partie II . D'une part il décrit les principes architecturaux du middleware COTAMA utilisé pour déployer l'architecture temps réel tolérante aux fautes. D'autre part il détaille comment la mission de livraison de courrier retenue, peut être déclinée au sein de cette architecture, au travers des niveaux fonctionnel et exécutif.

La quatrième partie s'inscrit comme la pierre angulaire de ce document puisqu'il explicite avec précision comment la méthodologie de conception d'une architecture de contrôle tolérante aux fautes proposée partie II peut être projetée dans le cadre expérimental présenté. Les mécanismes mis en place pour assurer l'identification des fautes ainsi que la détection, le diagnostic et le recouvrement en présence de défaillance y sont détaillés et commentés.

La cinquième partie analyse avec précision, lors d'une mission de livraison, le comportement de l'architecture tolérante aux fautes ainsi construite, en présence d'une succession de dysfonctionnements réels ou volontairement provoqués.

Enfin la conclusion, après avoir souligné l'importance de cette problématique et rappelé les apports de ce travail, identifie les limitations de l'approche proposée et présente un ensemble de pistes de recherche qui pourraient contribuer à améliorer les capacités de tolérance aux fautes des architectures de contrôle robotiques.

Première partie

État de l’art

Chapitre 1

Contexte de travail

Le contexte de cette thèse est la robotique, c'est à dire la gestion de robots autonomes dans un univers variable et incertain. Ce travail cherche a proposer un robot autonome s'adaptant à sa mission, a son environnement et à leurs aléas. La réalisation d'un robot parfaitement autonome sûr et permettant de répondre à une grande variété de mission est encore une utopie et nécessitera de nombreux travaux. Nombres d'études relevant de nombreux thèmes de recherche, se sont penchés sur cette problématique. Dans cet objectif, les architectures de contrôle occupent une place centrale. Elles permettent de structurer les différentes tâches du robot et de décomposer les prises de décisions. Elles facilitent ainsi la définition et l'exécution de mission complexe.

Cependant toute mission aussi simple soit elle peut être entachée d'aléas d'origines diverses et variées. Les travaux sur la détection de fautes permettent alors d'identifier les comportements erronés du robot en fonction de son environnement, de sa mission et de son propre état. Il est alors possible de décider si le robot est apte à poursuivre sa mission. Les travaux sur la sûreté de fonctionnement cherchent à garantir une exécution sure de la mission du robot dans un contexte défini. Enfin une mission peut nécessiter lorsque cela est possible l'intervention d'un opérateur humain distant, dans le cadre de son exécution normale ou pour palier un dysfonctionnement. Les interactions Homme-Robot sont donc amenées à jouer un rôle important dans le succès d'une mission. Face au multiples facettes relevant de la conception de robot autonomes tolérants aux fautes cette première partie présentera les différents travaux permettant de contrôler, monitorer et réagir aux fautes subies par le robot.

1.1 Retour sur la fiabilité des robots

1.1.1 Constat d'un échec persistant.

Nous avons vu qu'il existait des robots de tous types, réalisant des missions diverses et variées. Par contre tous ces robots ont un même point commun, ils ne sont pas fiables, et à un moment ou a un autre ils s'arrêtent de fonctionner, tombent en panne, ou simplement sont

dans l'incapacité de finir leur mission .Si l'on prend l'exemple d'un système éprouvé telle q'une voiture, il est de notoriété que les nouveaux modèles sont susceptibles d'avoir des défauts de conception, de fiabilité, qui ne sont résorbés que par l'expérience de l'utilisation et le temps. De même en informatique, les nouveaux logiciels arrivent toujours avec leurs défauts de jeunesse ainsi que des failles de sécurité. Il est donc clairement utopique de penser qu'un robot parfait existe en sachant que dans la majorité des cas les robots ne sont que des prototypes.

Les travaux de Carlson et Murphy viennent corroborer cette conjecture, dans [Carlson and Murphy, 2005, Carlson, 2004] ils regroupent plusieurs études de groupes de recherche indépendants dans le but d'étudier l'origines des fautes qui ont perturbés leurs différents robots mobiles au cours de missions de terrain. L'analyse se base sur des données collectées sur 24 robots, de 15 modèles différents produit par 7 fabricants. La variété des robots s'étend d'un petit robot géométriquement reconfigurable de quelques centimètres a un tank modifié de 60 tonnes.

Cette étude évalue les causes des fautes des robots, leurs impacts sur les missions en cours, leurs fréquence et les temps de maintenance nécessaires à leur remise en fonctionnement. Carlson et Murphy proposent une classification, que l'on retrouve sur la figure 1.1 où ils regroupent les différentes probabilités d'occurrence des fautes.

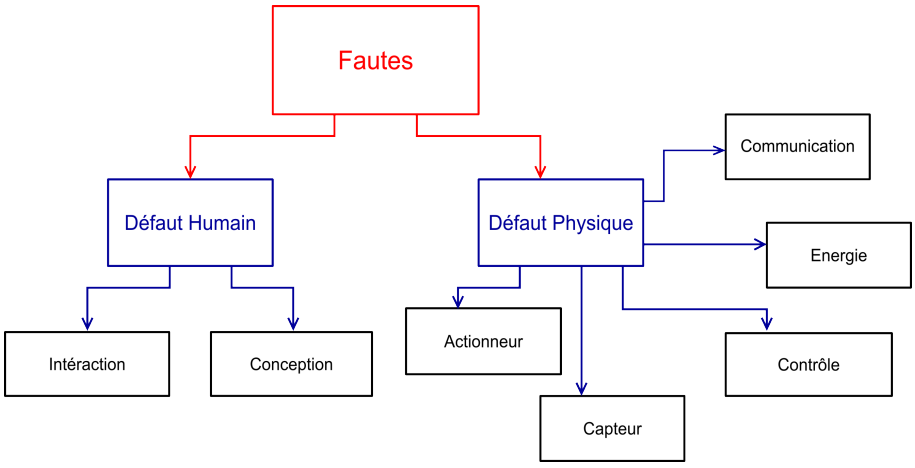


Figure 1.1 – Taxonomie des fautes présentée dans [Carlson and Murphy, 2005]

Il en résulte les statistiques suivantes :

Actionneur : entre 11% 50% des échecs ont été engendrés par des fautes d'actionneurs.

Capteurs : entre 9% et 26% des échecs d'une mission sont liés à des fautes de capteurs.

Contrôle : entre 33% et 54% des missions ont échoué à cause de défauts de contrôle.

Énergie : 9% des missions ont échoué suite à ce défaut

Communication : Aucune erreur de communication n'a été relevée lors de ces missions. La communication reste toutefois critique car elle peut conduire à la perte du robot en téléopération.

De plus il est a noté qu'entre 65% et 95% des cas la faute a mis fin à la mission en cours. Par contre, 80% des fautes commises ont pu être réparées sur site. Les défauts d'origine humaine sont sont identifiés dans 2 catégories présentées ci-dessous sans qu'aucune probabilité d'occurence n'ai pu être établie :

Conception : un défaut est introduit dans le robot lors de sa construction. Ce type de défaut est difficile à corriger.

interaction : une faute qui survient lorsque l'opérateur humain fait une action illicite ou erronée lors d'une interaction Homme-Robot.

Il est donc indispensable d'étudier ce qui empêche le robot de fonctionner correctement pour pouvoir par la suite trouver une réponse à ce dysfonctionnement mais surtout pour essayer de permettre au robot soit de terminer sa mission de manière nominale ou dégradée, soit de clore sa mission de façon sécuritaire pour lui et son environnement.

Après avoir présenté des travaux montrant que les fautes sont fortement perturbantes lors d'expérimentations ou lors de l'activité même du robot, nous allons maintenant présenter les différents travaux de la sûreté de fonctionnement qui ont pour but d'empêcher l'apparition de fautes, mais aussi de prévoir les mécanismes qui permettent au robot de continuer surement sa mission malgré leurs apparitions.

1.2 Principes de la sûreté de fonctionnement

[Avižienis et al., 2004, Laprie et al., 1996] définissent la sûreté de fonctionnement (*dependability* en anglais) comme *la propriété qui permet aux utilisateurs d'un système de placer une confiance justifiée dans le service qu'il leur délivre*. La sûreté de fonctionnement se définit selon les auteurs selon 3 axes différents : ses attributs, ses entraves et ses moyens. Les **attributs** de la sûreté de fonctionnement sont l'ensemble des propriétés que le système doit respecter pour rester sûr en fonctionnement. Ses **entraves** sont les causes qui empêchent le système de fonctionner correctement. Ses **moyens** sont les différentes possibilités qui permettent au système de rester sûr, même sous la contrainte d'un certain nombre d'entraves.

1.2.1 Les attributs de la sûreté de fonctionnement

Les attributs de la sûreté de fonctionnement sont des sous-ensembles de celle-ci, définis selon différents points de vue. Nous donnons ci-dessous les définitions de ces attributs, extraites ou fortement inspirées de [Laprie et al., 1996] :

Fiabilité (*reliability*) : capacité d'un système à accomplir une tâche dans des conditions défavorables, pendant une durée déterminée et dans une plage spécifiée. Cela concerne donc la continuité du service assuré par ce système. L'indicateur de la fiabilité le plus communément rencontré est le MTBF (*Mean Time Between Failures*), temps moyen entre les pannes, défini par la formule (1.1) suivante :

$$MTBF = \frac{1}{\lambda} = \frac{\text{durée d'utilisation du système}}{\text{Nombres de fautes commises pendant cette durée}} \tag{1.1}$$

avec λ : taux d'erreur

Disponibilité (*availability*) : représente le temps de fonctionnement efficace et sans problème. Un indicateur est là aussi communément admis comme représentatif et se calcule suivant la formule (1.2) :

$$A = \frac{MTBF}{MTBF + MTTR} \tag{1.2}$$

avec MTTR *(Mean Time To Repair)* le temps moyen de réparation, calculé à l'aide de la formule (1.3) :

$$MTTR = \frac{1}{\mu} = \frac{\text{Nombre d'heures pour faire les réparations}}{\text{Nombres de réparations faites}} \tag{1.3}$$

avec μ : taux de réparation

Sécurité ou sécurité-innocuité (*safety*) : absence de conséquence catastrophique sur les personnes, l'environnement ou les autres équipements en interaction avec le système.

Intégrité (*integrity*) : absence d'altérations inappropriées de l'information.

Maintenabilité (*maintainability*) : aptitude du système aux réparations et aux évolutions.

Cette décomposition de la sûreté de fonctionnement permet une meilleure approche de la sûreté de fonctionnement suivant les contextes du travail. Par exemple, un attribut tel que la **confidentialité** (*confidentiality*), réservé aux systèmes dont les applications sont à caractère confidentiel, ne sera pas mis en avant dans le domaine de la robotique. Par contre, nous retrouverons dans ce contexte de la robotique des attributs tels la **fiabilité**, la **sécurité** et la **disponibilité**, ainsi que dans de moindre mesure la notion de **maintenabilité**.

1.2.2 Les entraves de la sûreté de fonctionnement

Les entraves à la sûreté de fonctionnement se décomposent en 3 classes : les **fautes** (*fault*), les **erreurs** (*error*), et les **défaillances** (*failure*). Une faute est une déviation d'au moins un élément du système ou d'une de ses propriétés caractéristiques. Une faute est la cause adjugée ou supposée d'une erreur. Une erreur est la partie de l'état du système qui est susceptible d'entrainer une défaillance. Enfin, une défaillance est une déviation du service rendu par le système par rapport à son service nominal.

Une représentation proposée par [Lussier, 2007] de la propagation des fautes vers les défaillances dans les systèmes de systèmes est présenté figure 1.2. La présence d'une faute dans l'un des systèmes va lors de son activation provoquer la propagation d'une erreur jusqu'à provoquer une défaillance dans ce système. Cette défaillance peut alors causer une faute pour les systèmes avec lesquels le système défaillant interagit.

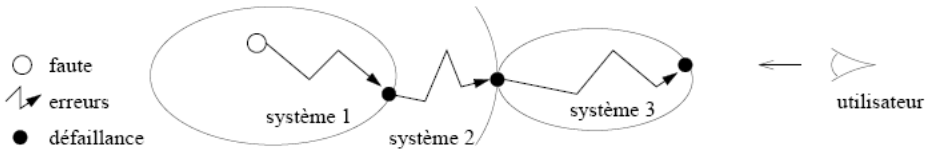


Figure 1.2 – Propagation d'erreurs dans un système de systèmes, [Lussier, 2007]

Classification des fautes

La classification des fautes élémentaires peut se faire selon de nombreux axes que l'on retrouve ci dessous :

- *La cause phénoménologique* : proposée par [Avižienis, 1978], les fautes sont soit physiques soit dues à l'Homme.
- *La nature* : qui classifie si la faute est accidentelle ou intentionnelle avec ou sans volonté de nuire.
- *La phase d'occurrence ou de création* : la faute est due au fonctionnement en cours ou au développement du robot.
- *frontières du système* : si la faute est interne ou externe.
- *persistance* : classifie les fautes selon la persistance de l'effet temporaire ou permanente sur le service. Cette classification est présentée plus en détail dans la section suivante.

Classification temporelle des fautes

Les fautes qui peuvent frapper un système au cours son existence relèvent fondamentalement de trois classes [Avižienis et al., 2004] : Les fautes physiques qui découlent de dysfonctionnements matériels, les fautes de développement qui sont induites lors de l'élaboration du système, enfin les fautes d'interaction qui interviennent durant la phase d'utilisation du dispositif. Elles peuvent être le résultat de phénomènes naturels (défaillance physique ou rayon cosmique) ou résulter d'actions humaines. Leur origine peut être située à l'intérieur (*internal fault*) ou à l'extérieur (*external fault*) des frontières du système.

L'occurrence d'une faute est liée à une évolution anormale d'un paramètre P du système dont la valeur, à un instant donnée, sort des limites acceptables qui lui sont associées (Figure 1.3). La faute F est alors détectée (F = 0).

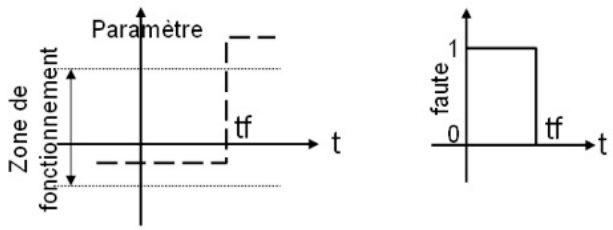
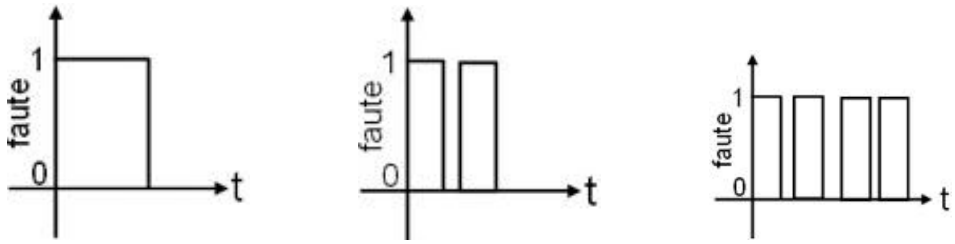


Figure 1.3 – Evolution d'un paramètre et détection de faute

De nombreuses autres caractéristiques peuvent être ajoutées afin de décrire plus précisément la nature d'une faute. Dans un premier temps on distingue les fautes abruptes (*abrupt*) qui entrainent un modification soudaine d'un paramètre ou d'un composant du système et les fautes de dérive (*incipient / drift like fault*) [Isermann, 2006] qui sont le résultat d'une évolution graduelle souvent lente.

La durée d'une faute est une dimension importante qui permet de distinguer classiquement trois types principaux de fautes soudaines (Figure 1.4) : les fautes abruptes permanentes (*abrupt fault*) qui ont un effet permanent sur le système, les fautes transitoires (*transient fault*) de durée limitée dans le temps, les fautes intermittentes (*intermittent fault*) qui correspondent à des fautes transitoires répétitives.



(a) faute abrupte (b) faute transitoire (c) faute intermittente

Figure 1.4 – Caractérisation d'une faute selon sa durée

Dans le cas le plus favorable, l'occurrence d'une faute peut avoir un simple effet additif facilitant sa détection. En revanche une faute peut aussi avoir un effet multiplicatif complexifiant sa mise en évidence [Isermann, 2006]. Évidemment, il est clair qu'un système réel doit être capable de faire face à la présence simultanée de plusieurs fautes.

1.2.3 Les moyens de la sûreté de fonctionnement

L'ensemble des moyens permettant de garantir la sûreté de fonctionnement est décomposé en deux grandes catégories. La première tend à éviter la présence d'entraves dans le fonctionnement du système. Elle se décompose en techniques de **prévention des fautes** et d'**élimination des fautes**. La seconde suppose que l'on accepte la présence des fautes et propose des moyens permettant soit de faire la **prévision des fautes**, soit de rendre le système **tolérant aux fautes**.

La prévention des fautes *(fault prevention)* représente l'ensemble des techniques mises en avant dans le domaine de l'ingénierie des systèmes pour leur bonne conception. L'apparition et la création des fautes sont fortement réduites par l'usage d'outils de développement sophistiqués, mais aussi par des techniques telle que la modularité. La modularité a pour but de découper le système en modules pour décomposer le développement d'algorithmes spécifiques. L'inter-connexion de ceux-ci permet d'obtenir le système dans son intégralité. Cela permet de faciliter le développement, et donc de limiter les fautes de conception,

mais aussi de faciliter les tests et les corrections à apporter au système.

L'élimination des fautes *(fault removal)* permet en cours de conception, en phase de test ou en phase opérationnelle, de réduire le nombre et la sévérité des fautes qui touchent le système. En phase de conception, l'élimination des fautes consiste en la vérification du respect des spécifications. Pour cela, les techniques de validation formelle permettent de prouver la validité d'un système. En phase opérationnelle, l'élimination des fautes est basée sur des techniques de maintenance corrective (éliminations des fautes qui ont produit des défaillances) ou préventive (élimination des fautes dormantes).

La prévision des fautes *(fault forecasting)* a pour but d'estimer la présence et les conséquences de fautes dans le système. Cette évaluation peut être conduite suivant deux axes différents. D'une part, une évaluation quantitative ou probabiliste qualifie la probabilité de respect des certains des attributs de la sûreté de fonctionnement. D'autre part, une évaluation qualitative ou ordinale aura pour but d'identifier et classifier les défaillances pouvant perturber le système.

La tolérance aux fautes *(fault tolerance)* a pour but de permettre au système de délivrer son service en dépit de fautes affectant ses différentes ressources. Les techniques de tolérance aux fautes sont généralement décomposées en trois phases : la détection des erreurs ou des défaillances, le diagnostic permettant d'identifier la faute à l'origine du problème, puis le recouvrement qui permet au système de continuer à fonctionner malgré l'occurrence du problème. La détection et/ou le diagnostic peuvent être suivis, lorsque cela est possible, d'une phase d'isolation des fautes détectées afin d'éviter qu'elles se reproduisent.

Après avoir présenté les notions et le vocabulaire relatifs à la sureté de fonctionnement et évoqué les différents moyens permettant de proposer des systèmes sûrs, nous allons nous présenter dans la suite sur les différentes techniques spécifiques à la tolérance aux fautes non sans évoquer les autres aspects de la sûreté de fonctionnement.

Chapitre 2

Approches pour la sûreté de fonctionnement

La sûreté de fonctionnement s'est largement développée, depuis une quarantaine d'année, dans de nombreux domaines industriels pour lesquels il est indispensable, pour des raisons sécuritaires (espace, avionique, centrale nucléaire, etc.) et/ou économiques (industrie chimique, pétrolière, etc.), d'assurer le fonctionnement le plus sûr et le plus optimal possible du système contrôlé même face à des évènements perturbateurs. Le schéma de contrôle classique en boucle fermée mis en œuvre pour piloter un processus physique (que se soit un dispositif mécatronique ou un procédé chimique) est présenté très schématiquement en figure 2.1.

Figure 2.1 – Schéma de contrôle en boucle fermée

Dans ce schéma, un contrôleur détermine la commande U à imposer au dispositif à contrôler, en fonction des informations capteurs Y reflétant l'état du système et de la consigne C à appliquer. Les principes de sûreté de fonctionnement qui ont été présentés dans le chapitre précédent doivent alors permettre d'atteindre les objectifs de fiabilité, disponibilité et sécurité visés même en présence de dysfonctionnements.

Dans cette partie nous allons donc balayer les principales démarches et méthodes développées pour déployer la sûreté de fonctionnement au niveau d'un système physique. Nous commencerons par présenter une typologie des fautes pouvant être observées. Cela nous permettra d'aborder les techniques de prévention de fautes avant d'évoquer celles d'élimination de fautes. Enfin nous nous attarderons plus longuement sur les approches de tolérances aux fautes qui relèvent de la problématique centrale de notre discours.

2.1 Prévention des fautes

La prévention des fautes représente l'ensemble des techniques mises en avant dans le domaine de l'ingénierie des systèmes pour en assurer la bonne conception. L'apparition et la création des fautes peuvent être fortement réduites par l'usage d'outils de développement adaptés, mais aussi par des techniques structurantes telle que la modularité. Celle-ci a pour but de décomposer le système en modules/composants pour faciliter son développement, limiter les fautes de conception mais aussi faciliter les tests et les corrections à apporter.

Les méthodes de gestion de projet issue de *l'ingénierie des systèmes* telle le cycle de développement en V (figure 2.2) permettent de prévenir la création des fautes et d'en encadrer la détection et l'élimination tout au long du processus de développement [Sommerville, 1988].

– La partie descendante de ce cycle spécifie à partir du cahier des charges le système jusqu'à la réalisation du système. La spécification fonctionnelle suivie de l'opérationnelle permet de décomposer le système et son développement tout en mettant en œuvre le principe de modularité.
– Les phases de tests de la partie montante sont réalisées composant et testant le système étape par étape en se conformant aux spécifications définies dans les phases en vis-à-vis. Lorsque des défauts sont détectés la boucle du circuit en V est relancée afin de corriger ces défauts du système.

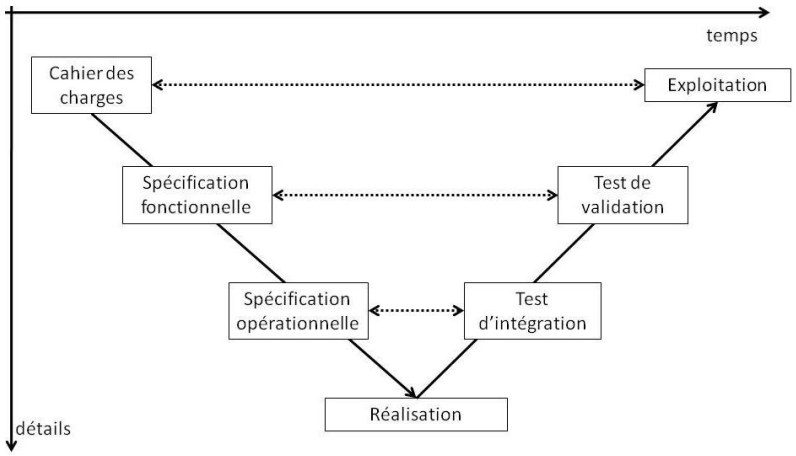


Figure 2.2 – Cycle de développement en V

Cette méthode apparentée à la gestion de projet relève de la prévention des fautes car elle permet d'encadrer la réalisation d'un système en faisant appel au concept de modularité et en définissant des points de passage structurant le processus de développement. Elle chevauche aussi le domaine de l'élimination des fautes que nous présentons dans la section suivante.

2.2 Elimination des fautes

L'élimination des fautes permet en cours de conception, en phase de test ou en phase opérationnelle, de réduire le nombre et la sévérité des fautes qui touchent le système. Il existe différentes techniques d'élimination des fautes. Chacune de ces techniques est souvent très spécifique à un type de fautes, l'ensemble de ces types est donc complémentaire pour traiter l'ensemble des fautes pouvant survenir dans un système.

2.2.1 Elimination des fautes au cours du développement d'un système

L'élimination des fautes au cours de la phase de développement du cycle de vie d'un système est principalement composée d'une étape de vérification. Cette phase s'applique à vérifier que le système est conforme à un ensemble de propriétés préalablement définies, appelées les conditions de vérification. Si ce n'est pas le cas, une phase de correction est exécutée pour éliminer la faute, avant de recommencer la vérification.

La figure 2.3 présente une classification des approches permettant de faire de la vérification, extraite de [Avižienis et al., 2004]. Elles sont principalement décomposées en fonction de leur mode de fonctionnement, suivant si elles exécutent ou non le système pour l'analyser.

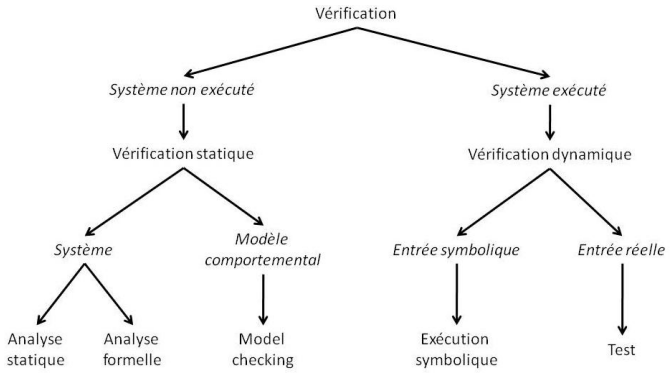


Figure 2.3 – Les approches de la vérification [Avižienis et al., 2004]

La vérification d'un système sans exécution est nommée la vérification statique. Elle peut être effectuée :

- sur un modèle comportemental du système, généralement un modèle de transitions (réseaux de Petri, automates à états finis ou infinis). Cela conduit au *model checking* [Merz, 2001], qui permet une vérification de propriétés exprimées en logique temporelle sur l'ensemble de l'espace d'états du modèle.
- sur le système lui-même, sous la forme d'analyses statiques (analyse de flot de données ou analyse syntaxique du code par exemple) ou par des techniques de preuve, qui consistent à prouver des propriétés par des démonstrations mathématiques [Lussier, 2004].

Les méthodes de ce type sont majoritairement mises en œuvre dans la phase descendante du

processus de développement en V présenté précédemment. Elles peuvent également être localisées dans la partie basse du cycle, lors de l'analyse élément par élément du système.

La vérification d'un système exécuté constitue la vérification dynamique. Les entrées fournies au système peuvent être soit symboliques dans le cas d'exécution symbolique, ou réelles dans le cas du test. Ces méthodes d'élimination des fautes sont le plus souvent exécutées sur l'implémentation du système lui-même ou sur un prototype, elles seront donc intégrées à la phase montante du processus de développement en V.

Les limitations de ces classes d'approches sont multiples et varient selon l'approche [Godary, 2004]. La complexité de la mise en œuvre et ou de l'exécution des méthodes formelles limite leur utilisation (expertise mathématique désirée, risques d'explosion combinatoire) alors que les approches de tests ou de simulation ne peuvent être exhaustives. En effet, la considération explicite du comportement d'un système à l'égard de toutes ses entrées possibles est généralement irréalisable. De plus, les séquences de test sont réalisées tardivement dans le cycle de développement ce qui retarde d'autant plus le processus et engendre des coûts importants de correction.

2.2.2 Elimination de fautes au cours de l'utilisation d'un système

Les approches d'élimination de fautes lors de l'utilisation d'un système [Avižienis et al., 2004] peuvent être décomposées en deux classes : la maintenance préventive qui vise à identifier et à éliminer les défauts avant qu'ils ne puissent provoquer des erreurs pendant le fonctionnement normal, et la maintenance corrective qui vise à éliminer les failles qui ont produit des erreurs signalées.

Ces méthodes d'élimination au cours de l'utilisation d'un système supposent que, malgré l'ensemble des moyens de prévention et d'élimination des fautes, il est nécessaire d'accepter la présence de fautes au sein du système développé. La mise en œuvre des phases de maintenances nécessite de disposer de moyens de prévisions de fautes, présentés dans la section suivante, pour pouvoir les éradiquer. De même, il sera nécessaire de déployer des mécanismes spécifiques, présentés sections 2.4 et 2.5, permettant au système d'être tolérant aux fautes pouvant survenir durant son fonctionnement.

2.3 Prévision des fautes

La prévision des fautes « *vise à estimer la présence, la création et les conséquences des fautes ou erreurs sur la sûreté de fonctionnement et peut permettre, par contrecoup, de chercher à les éliminer ou à les tolérer* » [Laprie et al., 1996]. Pour [Avižienis et al., 2004], elle consiste à réaliser une évaluation du comportement du système lors de l'occurrence d'une faute. Pour y parvenir, on peut avoir recours à des approches quantitatives et qualitatives.

Les méthodes quantitatives s'appuient sur des approches formelles. Les chaînes de Markov [Siewiorek and Swarz, 1992] par exemple représentent les différents états dans lesquels peut se trouver le système ainsi que les probabilités de transition entre ces états lorsque les composants sont opérationnels ou fautifs. Il est ainsi possible de calculer la probabilité qu'a le système de ce trouver dans les différents états stationnaires. Les difficultés de mise en œuvre de telles approches relèvent de l'obtention de données fiables sur le taux de défaillance associés aux fautes envisagées et de la maitrise de l'explosion combinatoire inhérente à ce type de modèle. Les méthodes qualitatives sur lesquelles nous allons nous attarder plus largement permettent d'identifier et de classer les modes de défaillance qui peuvent conduire à un dysfonctionnement du système lors de l'occurrence d'une faute.

Parmi les nombreuses techniques pouvant être employées [Gould et al., 2000] nous n'évoquerons que l'AMDEC ainsi que les arbres de défaillance et d'évènement [Mortureux, 2010, Chevance, 2010].

2.3.1 L'AMDEC

L'AMDEC (Analyse de Modes de Défaillances, de leurs Effets et de leurs Criticités) est une démarche inductive permettant d'analyser les conséquences d'une faute sur le système étudié. On la retrouve dans la terminologie anglo-saxonne sous l'acronyme FMCEA (*Failure Modes, Effects and Criticality Analysis*) [BSIGroup, 1991]. C'est une technique pluridisciplinaire qui associe l'ensemble des acteurs partie prenante dans le développement du système à analyser. Elle se décompose en général selon les étapes suivantes :

- Définir les limites du système étudié.
- Identifier les fonctions réalisées par le système. La Technique SADT (*Structured Analysis and Design Technique*) [Ross, 1977] qui permet une décomposition fonctionnelle descendante peut par exemple être employée. Elle permet un raffinement hiérarchique des activités entre lesquelles transitent les données informationnelles et de contrôle.

Puis pour chacune des fonctions définies :

- Identifier les modes de défaillance (complète, partielle, intermittente, etc.) de la fonction analysée.
- Identifier les effets (conséquences) d'un mode de défaillance sur le système.
- Établir la sévérité de chaque effet identifié. Une classification qualitative comportant plusieurs niveaux de sévérité (de défaillance mineure à catastrophique) est généralement adoptée.
- Identifier les causes des modes de défaillance et donc les fautes qui peuvent conduire à un dysfonctionnement de la fonction analysée.
- Évaluer les moyens de détection possibles des défaillances identifiées.
- Estimer la probabilité d'occurrence P de la défaillance. Cette dernière si elle ne peut être quantifiée peut être définie qualitativement (de probabilité très faible à élevée par exemple).

– Définir un coefficient de gravité G de la défaillance allant par exemple de minime à très grave.
– La criticité C de la défaillance est alors définie comme étant le produit $C = P \times G$.
– Proposer des actions de réduction des risques de défaillance.

Cette analyse est en général synthétisée sous la forme d'un tableau. L'une des limitations de l'AMDEC est de ne pas pouvoir prendre en compte simultanément plusieurs défaillances. C'est pourquoi on lui associe souvent les arbres de défaillance.

2.3.2 Arbres de défaillance

Un arbre de défaillance (*Fault tree*) autorise une analyse déductive, à travers un arbre logique (Figure 2.4), permettant de remonter d'une défaillance observée, source du raisonnement, aux origines potentielles de celle-ci (fautes). L'arbre est décomposé, usuellement de haut en bas, en couches comportant des nœuds. Chacun d'eux détaille la ou les combinaisons d'évènements, traduite à l'aide d'opérateurs logiques (ET, OU), susceptibles de produire l'évènement précédent.

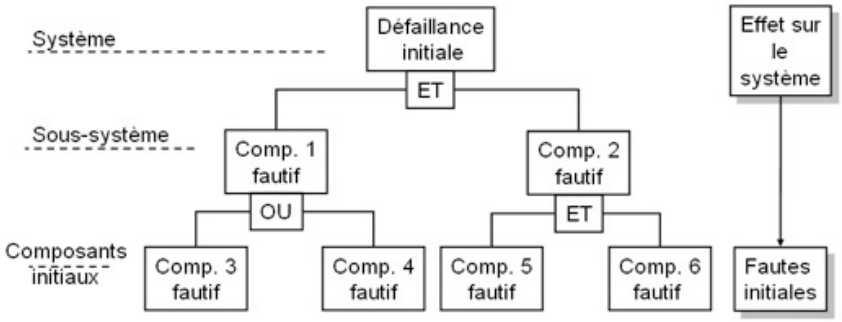


Figure 2.4 – Arbre de défaillances (adapté de [Isermann, 2006])

Par exemple, dans [Brissaud, 2011] l'auteur utilise les arbres de défaillances pour modéliser à la fois les aspects matériels et fonctionnels d'un réseau de Capteurs-Transmetteurs à fonctionnalités numériques. La méthodologie permet d'évaluer des critères de sureté de fonctionnement du systèmes en fonction de ses politiques de maintenance et de certains facteurs d'influence.

Une logique de construction duale peut être mise en œuvre pour construire un arbre d'évènement.

2.3.3 Arbres d'évènement

L'arbre d'évènement (*Event tree*) suit pour sa part une logique inductive en cherchant à représenter les différents conséquences induites par un évènement initiateur (défaillance originelle). L'arbre construit est orienté de la gauche vers la droite (Figure 2.5).

Il est là aussi décomposé en couches de plusieurs nœuds associées aux alternatives de fonctionnement d'un des composants pouvant être influencé par le dysfonctionnement initial. Pour

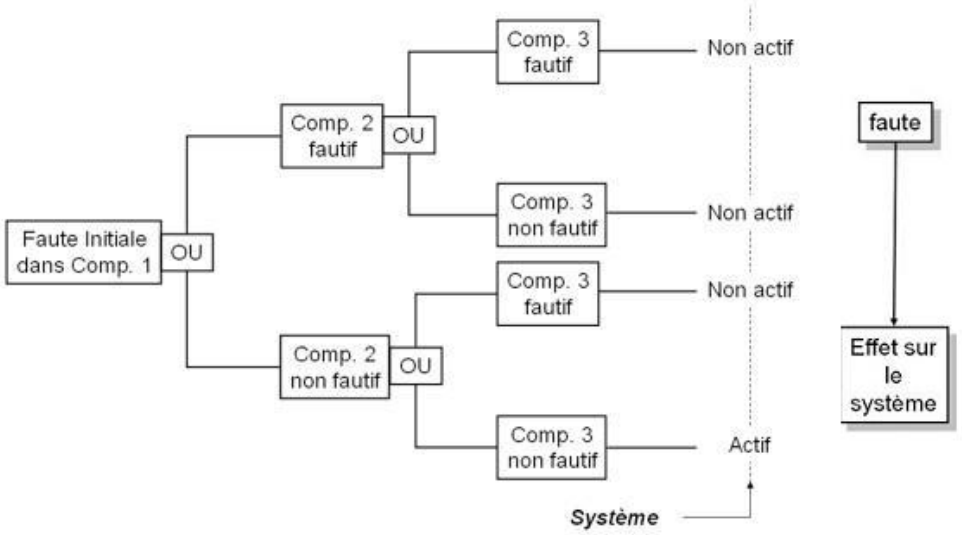


Figure 2.5 – Arbre d'évènements (adapté de [Isermann, 2006])

un composant donné chaque branche supérieure d'un nœud exprime le fonctionnement souhaité, et celle inférieure le fonctionnement fautif. L'arbre d'évènement ne traduit donc qu'un enchainement de OU logiques.

Les méthodes et outils que nous venons de présenter permettent une connaissance a priori des fautes auxquelles peut être exposé un système et de leur effet sur ce dernier. Elles procurent des informations essentielles pour le déploiement des approches de tolérance aux fautes et de prévention de fautes indispensables à la sûreté de fonctionnement.

2.4 Tolérance aux fautes : méthodes de détection et diagnostic de fautes

Comme nous avons vu précédemment la tolérance aux fautes est la qualité qui permet d'assurer la délivrance d'un service correct en dépit de fautes affectant les différentes ressources du système. Pour assurer le fonctionnement il faut donc être capable de détecter les fautes et de diagnostiquer précisément leurs origines pour permettre ensuite de délivrer, malgré les fautes, au mieux le service.

Il existe de nombreuses méthodes permettant de détecter la panne d'un système. Dans un premier temps nous présenterons dans cette section les étapes qui permettent de mettre en œuvre la tolérance aux fautes. Les méthodes de détection et diagnostic seront présentées ensuite avant d'aborder dans la section suivante l'ensemble des moyens permettant de répondre au diagnostic des fautes.

2.4.1 Les étapes de la tolérance aux fautes

Le développement de Systèmes de Contrôle Tolérants aux Fautes (SCTF) (*Fault Tolerant Control System*) est, depuis de nombreuses années, un champ d'investigation scientifique très dynamique où se côtoient de très nombreuses approches [Patton, 1997, Isermann, 2006], relevant de problématiques pluridisciplinaires [Zhang and Jiang, 2008] couvrant des champs d'investigations variés allant de l'avionique à la robotique.

L'objectif de cette partie du document est de présenter un tour d'horizon des principaux travaux proposés pour la tolérance aux fautes. Nous commencerons par identifier les étapes nécessaires au développement de SCTF. Puis, nous présenterons les méthodes de détection et de diagnostic de fautes, avant d'aborder les principes de recouvrement envisageables. Nous terminerons par une analyse critique de l'ensemble de ces travaux.

Les Systèmes de Contrôle Tolérants aux Fautes doivent avoir la capacité de maintenir la stabilité du système contrôlé ainsi qu'une performance acceptable (*gracefull performance*) même en présence de fautes [Zhang and Jiang, 2008].

Classiquement le développement de SCTF s'appuie sur la mise en œuvre successive de quatre tâches distinctes.

- La détection de la faute (*Fault Detection*) qui permet de savoir que le système contrôlé est confronté à un dysfonctionnement à un instant donné.
- L'isolation de la faute (*Fault Isolation*) qui a pour objectif de localiser l'emplacement précis de la faute.
- L'identification de la faute (*Fault Identification*) qui détermine la sévérité (magnitude) de la faute.
- Enfin le recouvrement (*Recovery*) qui, en fonction de la faute identifiée permet de modifier, si nécessaire, la structure du contrôleur de façon à maintenir la stabilité du système tout en préservant une performance acceptable.

Dans la littérature on trouve couramment les acronymes suivants :

- FDI : *Fault Detection and Isolation* ;
- FDD : *Fault Detection and Diagnosis* où le diagnostic correspond aux tâches d'isolation et d'identification de la faute. Nous parlerons pour notre part de Détection et Diagnostic de Faute (DDF).

Compte tenu de des différentes tâches évoquées, il est possible de présenter la structure générale que doit alors posséder un Système de Contrôle Tolérant aux Fautes (Figure 2.6).

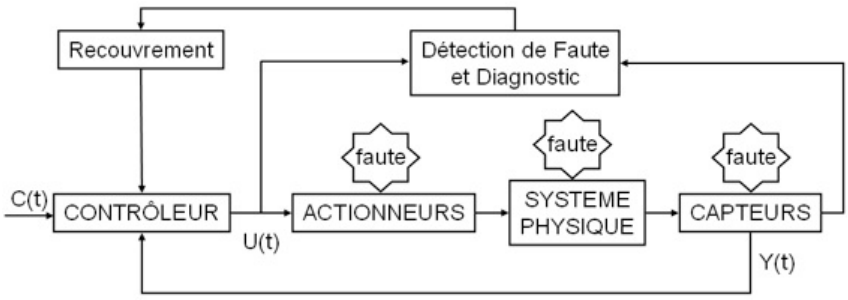


Figure 2.6 – Structure de principe d'un SCTF

Le schéma de principe proposé, que l'on retrouve dans de nombreux articles, montre que l'hypothèse de travail usuellement retenue est que les fautes ne peuvent affecter que les capteurs, les actionneurs où le système physique contrôlé. La détection, l'isolation et l'identification de la faute est prise en charge par un module de détection et diagnostic de faute. Enfin un mécanisme de recouvrement adapté est mis en œuvre à partir des informations issues du diagnostic pour agir sur le contrôleur du système de façon à compenser les effets de la faute en maintenant la stabilité du système et une performance acceptable.

Cependant, un tel schéma de contrôle, pour être opérationnel et efficace, se doit de satisfaire quelques critères incontournables. Tout d'abord le mécanisme de détection et de diagnostic de fautes doit fournir des informations précises et aussi sûres que possible sur la faute (date, type, sévérité) de façon à choisir les actions de recouvrement adaptées. Par ailleurs, il doit absolument, quelque soit la situation rencontrée, maintenir le système contrôlé opérationnel (stabilité, performance), plutôt que de le voir s'effondrer totalement. Enfin, et c'est un critère majeur, la combinaison des tâches de détection et diagnostic de faute et de recouvrement doit impérativement se faire en ligne et en temps réel. En effet des délais de traitement trop importants auraient une conséquence fâcheuse sur la stabilité du système [Mariton, 1989].

2.4.2 Le diagnostic

Le diagnostic de faute nécessite de disposer au préalable d'une connaissance de l'ensemble des fautes ainsi que des relations existantes entre des symptômes (observations) et des pannes. La stratégie de diagnostic est fortement dépendante de la nature des schémas de représentation de la connaissance. Les méthodes utilisées peuvent alors faire appel à des démarches quantitatives ou qualitatives.

Les auteurs des différents états de l'art suivant [Zhang and Jiang, 2008], [Isermann, 2006] et [Venkatasubramanian et al., 2003c, Venkatasubramanian et al., 2003a, Venkatasubramanian et al., 2003b] s'accordent pour décomposer les approches de détection et de diagnostic de faute en deux classes principales : les méthodes basées modèle (*Model based*) et celles basées données (*Data-based*) (Figure 2.7).

Nous commencerons par nous intéresser aux méthodes basées modèle qui s'appuient sur une connaissance approfondie (*deep knowledge*), causale et a priori du système à contrôler.

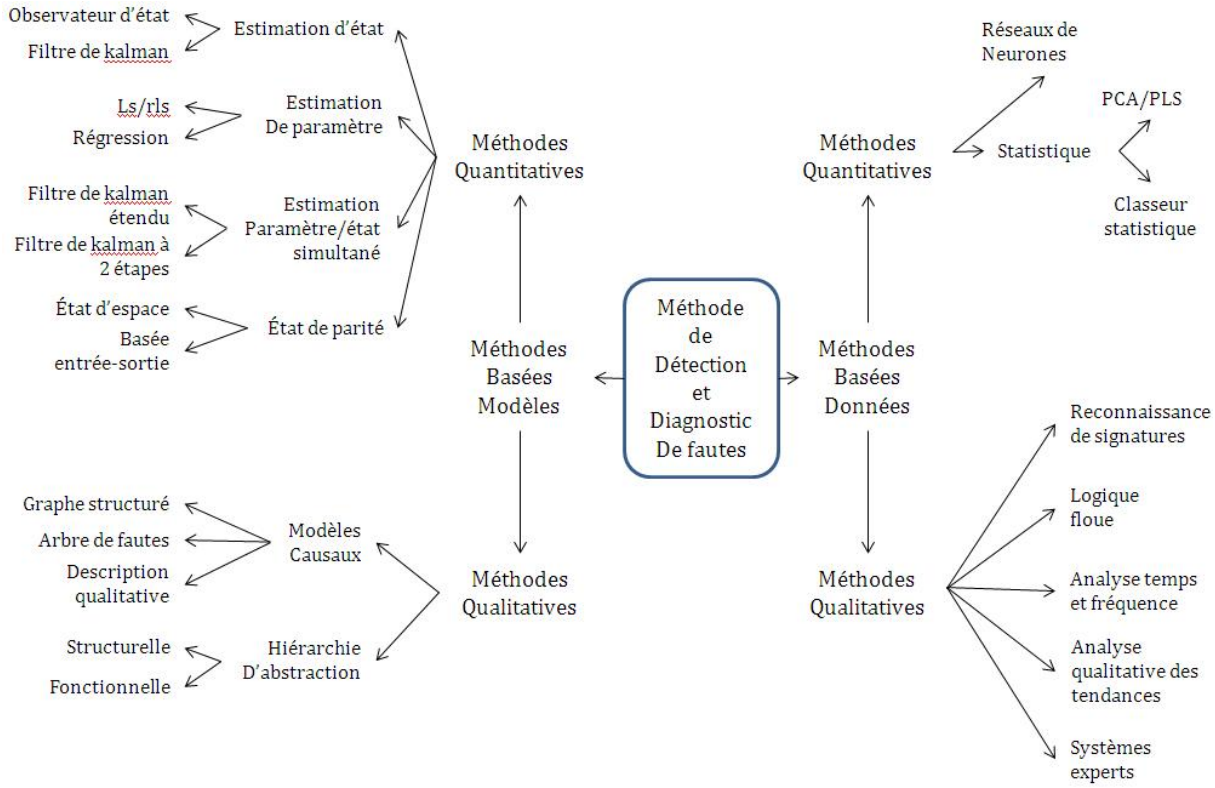


Figure 2.7 – Une classification des méthodes de détection et diagnostic de faute

2.4.2.1 Méthodes basées modèle quantitatives

Dans les approches quantitatives la représentation du système contrôlé s'effectue à travers un ensemble d'expressions mathématiques modélisant les relations existant entre les entrées et les sorties du système. Elles sont exploitées pour la DDF à travers le mécanisme de redondance analytique (*Analytical (software) redundancy*).

Le principe de redondance a toujours joué un rôle central dans le développement de systèmes tolérants aux fautes. Historiquement la redondance matérielle a été utilisée pour détecter la présence de dysfonctionnements dans les applications critiques (avionique). Cette approche simple permet une détection et une localisation rapide de la faute. Cependant compte tenu du coût engendré (multiplication des capteurs et augmentation de la probabilité de panne) cette démarche est actuellement complétée [Bartley, 2001] ou totalement remplacée par la mise en place d'une redondance analytique / logicielle.

Dans ce cadre le processus de DDF est décomposé en trois étapes (Figure 2.7). Dans un premier temps, on s'assure du bon fonctionnement du système surveillé en comparant son comportement avec celui obtenu à l'aide d'un modèle mathématique précis et explicite [Frank, 1990] du système. En effet, l'occurrence de fautes induisant des changements dans les variables d'état et/ou les paramètres du modèle, il est possible d'estimer ces modifications, à partir de l'observation des informations capteurs et de la commande du système, en faisant appel à des

méthodes d'estimation d'états ou de paramètres [Frank, 1998]. On distingue usuellement les méthodes suivantes qui présentent, sous certaines hypothèses, des analogies de formulation [Gertler, 1995, Frank and Wünnenberg, 1989] :

- Les observateurs d'état : le principe est de reconstruire les états d'un système uniquement à partir de la connaissance de ses entrées et sorties et d'un modèle du système. Les observateurs d'états permettent d'engendrer une erreur d'estimation pour les fautes auxquelles ils sont sensibles. De nombreuses approches ont été développées comme [Luenberger, 1971, Frank, 1990] pour le cas déterministe ou [Massoumnia, 1986] pour les filtres détecteurs de défaut. Il est important de souligner que les filtres de Kalman [Willsky and Jones, 1976], relèvent de cette classe d'approche.
- Les équations de parité : dans ce cas les équations du modèle sont projetées dans l'espace de parité où les redondances permettent d'éliminer les inconnues [Willsky, 1976, Ben-Haim, 1980, Gertler and Singer, 1990]. Les équations obtenues ne font alors intervenir que des variables mesurables correspondant aux entrées et sorties du système. Il est alors possible de vérifier la consistance (la parité) de leurs valeurs vis-à-vis des valeurs mesurées pour détecter un dysfonctionnement.
- L'identification paramétrique [Isermann, 1984, Young, 1981] : cette approche considère que l'impact des défauts se reflète sur les paramètres du système et non pas, comme c'est le cas sur les observateurs sur les variables de celui-ci. On analyse donc en permanence l'évolution de ces paramètres par le biais de mesures entrées/sorties de façon à évaluer la distance qui les sépare de valeurs de référence en fonctionnement nominal.

Dans un second temps, toute inconsistance (écart de comportement significatif) entre le comportement réel et simulé va alors générer un résidu qui va être évalué par quantification (seuillage simple ou adaptatif, tests statistiques [Willsky, 1976], réseau de neurones [Koppen-Seliger et al., 1995]) de façon à générer un symptôme signifiant qu'un dysfonctionnement a été détecté.

Enfin la phase de diagnostic doit permettre de localiser la faute en s'appuyant sur l'agencement et la connectivité du système étudié. Pour cela on fait souvent appel à la notion de signature de la panne qui est engendrée par la structure du vecteur de résidus du système [Simani et al., 2003]. Deux types d'approches sont envisageables [Gertler, 1991]. D'une part il est possible de construire des résidus structurés qui ne sont affectés que par un sous-ensemble de fautes et robustes (non affecté) par les autres. D'autre part des résidus directionnels [Gertler and Monajemy, 1995] peuvent être définis. Ils sont élaborés de façon à ce qu'en réponse à une panne donnée, le vecteur des résidus s'oriente selon une direction précise dans l'espace des résidus.

Les approches quantitatives basées modèle ont donc largement été étudiées et sont très largement utilisées [Zhang and Jiang, 2008]. Il est possible d'identifier un certain nombre de limitations [Venkatasubramanian et al., 2003c]. D'une part, ces approches sont le plus souvent limitées aux modèles linéaires du système analysé. D'autre part la plupart des travaux ne consi-

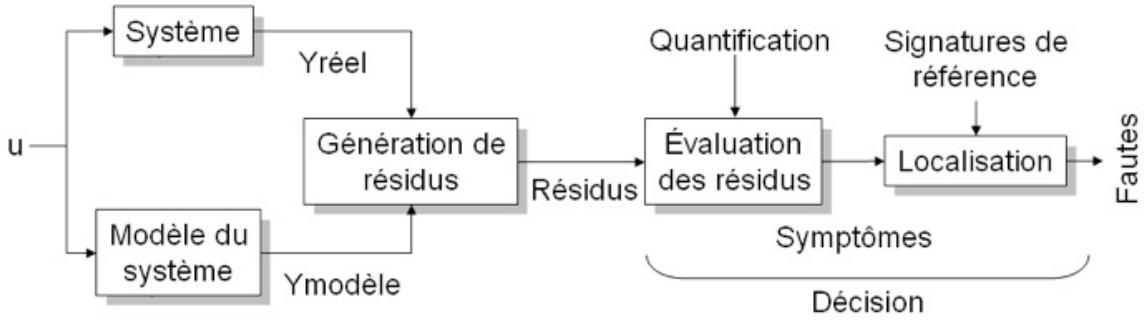


Figure 2.8 – Schéma de principe de la redondance analytique

dèrent que des perturbations intervenant simplement de façon additive. Bien évidemment cette classe d'approche suppose que l'on connaisse au préalable les défauts pouvant affecter un système sans donc garantir qu'une faute non envisagée puisse être détectée. Enfin, pour des systèmes de grandes dimensions, le temps de calcul nécessaire pour certaines techniques (banque de filtres de Kalman) peut devenir rédhibitoire.

Nous allons maintenant nous intéresser aux méthodes basées modèle qui, plutôt que de faire appel à une modélisation mathématique des relations entrées/sorties du système s'appuient sur une évaluation qualitative de celles-ci.

2.4.2.2 Méthodes basées modèle qualitatives

Dans un premier temps, le diagnostic étant par essence un processus causal, cette classe d'approche fait souvent appel à des modèles causaux pour établir une base de connaissance du système étudié sans pour autant nécessairement disposer d'une compréhension détaillée des phénomènes physiques qui lui sont associés. Puis, dans un second temps, à partir de cette représentation des connaissances des stratégies de recherche sont utilisées pour réaliser le diagnostic de faute [Venkatasubramanian et al., 2003a].

Les modèles relevant de la physique qualitative ont été souvent utilisés par la communauté de l'intelligence artificielle. Certains travaux cherchent à établir un ordre causal entre les différentes variables d'un système à partir des relations fonctionnelles connues du système [Iwasaki and Simon, 1986] ou à l'aide des équations différentielles de ce dernier [Soylemez and Seider, 1973]. D'autres extrapolent le comportement qualitatif de ce système en présence de fautes en s'appuyant sur des équations différentielles ordinaires [Sacks, 1988] ou qualitatives [Kuipers, 1986]. L'avantage de ces approches est de pouvoir, par simulation, tirer des conclusions en ne disposant le plus souvent que d'une connaissance incomplète et incertaine du système.

Les digraphes (graphes orientés) peuvent aussi être employés pour traduire graphiquement les relations causales reliant les différentes variables d'un système. Les nœuds représentent ces dernières, les arcs les relations entre ces nœuds. Introduits par [Iri et al., 1979] dans le domaine du diagnostic de faute, ils sont aussi employés pour construire d'autres mo-

dèles comme les arbres de simulation [Kramer and Palowitch, 1987] ou les graphes cause-effet [Wilcox and Himmelblau, 1994]. Des approches floues ont été introduites pour en améliorer la résolution [Tarifa and Scenna, 1997]. Enfin d'autres modèles [Finch and Kramer, 1988] font appel à un processus d'abstraction hiérarchique qui avance que le comportement du système global peut être inféré simplement à partir des lois gouvernant le comportement de ses sous-systèmes. Évidemment ici le principe de décomposition structurel et fonctionnel est central. De telles approches, pour des systèmes complexes, permettent de rapidement de se focaliser sur la zone à l'origine du dysfonctionnement pour employer alors des techniques plus performantes localement [Prasad et al., 1998].

En ce qui concerne les stratégies de diagnostic, en fonction des modèles qualitatifs utilisés on peut envisager des approches de recherche symptomatiques et topographiques [Rasmussen, 1986]. La stratégie symptomatique fait appel aux symptômes pour guider la recherche et localiser la faute. Le principe de diagnostic s'appuie le plus souvent sur l'emploi itératif d'un mécanisme de génération et de vérification d'hypothèse. Par exemple, une fois cette dernière formulée, on détermine par simulation l'effet de la faute envisagée sur le système, puis on compare le résultat obtenu avec les données observées. En cas de correspondance totale ou partielle l'hypothèse est retenue puis la procédure est répétée jusqu'à ce qu'aucune meilleure hypothèse ne soit trouvée. Évidemment l'efficacité de ce type de stratégie dépend principalement de l'efficacité du générateur d'hypothèse et de sa capacité à réduire l'ensemble des fautes envisageables grâce par exemple à des méthodes probabilistes [Peng and Reggia, 1987a, Peng and Reggia, 1987b]. La stratégie topographique développe quand à elle une analyse des dysfonctionnements s'appuyant sur une représentation du comportement normal. On compare le système fautif par rapport à celui de référence en fonctionnement normal. On peut faire appel à une stratégie structurelle qui s'appuie sur identification itérative des chemins de données incluant un composant fautif pour déterminer celui à l'origine de la faute [Milne, 1987]. Une méthodologie similaire est utilisée pour développer une stratégie fonctionnelle de diagnostic en analysant les fonctionnalités saines et fautives. A l'inverse de l'approche symptomatique, l'approche topographique présente l'avantage de ne pas nécessiter une connaissance a priori de la nature des dysfonctionnements mais reste limitée à la simple localisation du sous-système fautif.

La principale limitation des approches qualitatives reste la possibilité de génération de solution erronée lors du diagnostic.

Après avoir présenté les approches basées modèle nous allons maintenant nous intéresser à celles qui s'appuient sur un historique du fonctionnement du système.

2.4.2.3 Méthodes basées données

L'étape initiale de toute démarche de diagnostic s'appuyant sur un historique mémorisé est le processus qualitatif ou quantitatif d'extraction de caractéristiques au sein duquel les données initiales sont transformées en une représentation des connaissances pouvant être utilisée par le

système de diagnostic [Venkatasubramanian et al., 2003b].

Les principales approches qualitatives sont les systèmes experts et les méthodes de modélisation de tendance.

Un système expert est composé d'une part d'un dispositif d'acquisition, de représentation et de codage des connaissances (base de connaissances), et d'autre part d'un ensemble de procédures d'inférence pour le diagnostic. Un état de l'art des systèmes experts utilisés dans la communauté de l'intelligence artificielle de 1995 à 2004 est présenté dans [Liao, 2005].

Depuis [Henley, 1984] ils ont été largement utilisés pour le diagnostic de faute sur différents systèmes physiques [Ramesh et al., 1989, Venkatasubramanian, 1989]. Certains travaux ont cherché à améliorer les performances de la base de connaissance en la structurant [Ramesh et al., 1988] ou en introduisant des heuristiques [Basila et al., 1990]. D'autres travaux associent le système expert à un réseau de neurones pour réaliser un pré-diagnostic [Becraft and Lee, 1993, Zhao et al., 1997]. La dimension probabiliste a aussi été traitée dans [Leung and Romagnoli, 2000]. Les réseaux bayesiens sont aussi utiliser pour déterminer les fautes et perturbations de systèmes dynamiques [Mechraoui, 2010].

Les systèmes experts sont assez faciles à développer et peuvent aisément fournir des explications sur la solution proposée. Cependant la base de connaissances utilisée reste toujours réduite, très spécifique et difficile à maintenir [Rich and Venkatasubramanian, 1987].

L'extraction d'informations pour l'analyse de tendance qualitative est aussi importante pour la conduite d'un système. En effet la modélisation de tendance peut être utilisée pour en expliquer le comportement, réaliser un diagnostic de fautes ou prédire son évolution [Venkatasubramanian et al., 2003b]. De nombreuses méthodes de représentation des tendances ont été proposées : triangulation [Cheung and Stephanopoulos, 1990], différences finies [Janusz and Venkatasubramanian, 1991], ondelettes [Vedam and Venkatasubramanian, 1997].

Les méthodes quantitatives quant à elles formulent la problématique du diagnostic sous la forme d'un problème de reconnaissance de formes où les données sont affectées à des classes prédéterminées grâce à des outils statistiques où à des réseaux de neurones.

En fonctionnement fautif les probabilités de distributions des mesures réalisées et les caractéristiques associées (moyenne, déviation standard) diffèrent de celles observées pour un comportement sain du système. Pour traiter les systèmes de grandes dimensions l'analyse en composantes principales est largement utilisée. Elle permet de représenter l'ensemble initial des variables corrélées du système par un ensemble plus réduit de variables non corrélées [Pearson, 1901, Hotelling, 1947]. Cette technique a été largement utilisée pour la détection de faute et le diagnostic en ligne au niveau de systèmes de grandes dimensions [MacGregor and Kourti, 1995] mais aussi de capteurs [Dunia et al., 1996, Qin and Li, 1999]. L'un des problèmes des méthodes statistiques est de s'affranchir des bruits de mesure de façon à éviter de mauvaises interprétations et déclencher de fausses alarmes.

Les réseaux de neurones ont eux aussi largement été employés pour le diagnostic de fautes [Isermann and Ballé, 1997, Venkatasubramanian, 1985, Whiteley and Davis, 1994] à des fins

de classification et donc de reconnaissance de formes. Pour une structure donnée du réseau (nombre de neurones, de couches), après une phase d'apprentissage permettant de configurer les poids synaptiques et les fonctions d'activation à partir des données disponibles, le réseau de neurones est capable d'analyser une nouvelle forme d'entrée de façon à la classifier et donc à déterminer la nature et la présence d'une faute dans le système. Ce type d'outil rapide et robuste au bruit peut cependant difficilement travailler en dehors de son domaine d'apprentissage et peine à prendre en considération des fautes multiples.

Les approches de diagnostic de fautes s'appuyant sur des données historisées sont très largement utilisées dans l'industrie, particulièrement pour détecter des dysfonctionnements des capteurs, car elles sont relativement simples à mettre en œuvre et nécessitent peu d'effort de modélisation et de connaissance a priori sur le système [Venkatasubramanian et al., 2003b].

Nous venons d'achever le tour d'horizon des très nombreux modèles et approches de diagnostic pouvant être utilisés pour localiser l'origine du dysfonctionnement observé sur le système que l'on cherche à contrôler. Nous allons maintenant aborder la présentation des principales techniques proposées pour maintenir la stabilité et si possible la performance d'un système en présence de fautes.

2.5 Tolérance aux fautes : Techniques de recouvrement

Le recouvrement est une étape essentielle de la tolérance aux fautes qui est traditionnellement décomposée en deux classes. D'une part elle peut être passive (*Passive Fault Tolerant Control System*) lorsque le contrôleur est conçu pour être robuste vis-à-vis d'un ensemble de fautes présumées [Eterno et al., 1985]. D'autre part elle peut être active (*Active Fault Tolerant Control System*), lorsque le contrôleur est reconfiguré dynamiquement pour faire face à l'occurrence de fautes. A ces deux classes nous adjoindrons pour notre part la possibilité, lorsque cela est possible, d'adapter le niveau d'autonomie du système en introduisant ponctuellement ou définitivement l'entité Humaine dans la boucle de contrôle.

Nous allons maintenant présenter les principales techniques relevant de ces trois approches de tolérance aux fautes.

2.5.1 Recouvrement passif

De fait, le recouvrement passif ne nécessite pas de disposer d'un diagnostic préalable d'occurrence et de nature de la faute. Le contrôleur est, dès sa conception, synthétisé de façon à être insensible à certaines fautes tout en préservant la stabilité et la performance du système lors de leur apparition.

Les approches robustes telles que la méthode QFT (*Quantitative Feedback Theory*) proposé par [Horowitz et al., 1985], la commande CRONE [Oustaloup and Melchior, 1993] ou la synthèse H_∞ [Murad et al., 1996] sont très performantes mais limitées aux incertitudes paramétriques.

L'approche par stabilisation simultanée peut être une réponse à la sélectivité de la démarche robuste. Son principe est de chercher un contrôleur unique « unifiant » l'ensemble des contrôleurs permettant de préserver la stabilité du système en fonctionnement nominal et en présence d'un défaut connu. A la suite de travaux de Youla [Youla et al., 1974] relatifs à la conception de régulateur asymptotiquement stables, de nombreux auteurs se sont intéressés au problème de stabilisation simultanée [Saeks and Murray, 1982, Noura et al., 1993, Howitt and Luus, 1991]. Cependant le nombre de fautes considérées reste en général faible.

Enfin la commande adaptative [Landau et al., 1998] permet d'ajuster en temps réel les paramètres du contrôleur en comparant les performances réelles du système contrôlé avec celles désirées [Huang and Stengel, 1990, Ochi, 1993].

Les techniques de recouvrement passif restent limitées à un nombre restreint de fautes. Pour obtenir une meilleure couverture des fautes il faut s'orienter vers la tolérance aux fautes active.

2.5.2 Recouvrement actif

A l'opposé de l'approche passive, le recouvrement actif nécessite une connaissance préalable des fautes pouvant affecter le système. Ainsi, il est possible de prévoir les actions de reconfiguration à mettre en œuvre au niveau du contrôleur tout en connaissant les nouvelles limitations de fonctionnement induites par la faute. On cherche alors à préserver la stabilité du système et la meilleure performance possible [Zhang and Jiang, 2008].

Il existe de très nombreuses approches de recouvrement actif [Zhang and Jiang, 2008] s'appuyant sur un très large éventail de schémas de contrôle (méthode de la pseudo-inverse, commande par placement de pôles, commande à mode glissant, etc.). Cependant il est possible de dégager deux principales stratégies de reconfiguration du contrôleur. D'une part il peut être possible de sélectionner des lois de contrôle prédéterminées pour compenser l'impact de fautes connues. Des approches faisant appel à des modèles multiples et des commandes à gain variable peuvent par alors être employées [Boskovic and Menta, 2002, Thelliol et al., 2003, Moerder et al., 1989, Yen, 2003]. D'autre part, un nouveau contrôleur peut aussi être synthétisé en ligne pour répondre à l'occurrence d'une faute [Looze et al., 1985, Zhang and Jiang, 2002].

Pour obtenir un mécanisme de recouvrement actif performant il faut souligner l'importance de connaitre le plus rapidement possible la nature exacte du défaut et sa date d'occurrence. Faute de quoi, et bien que des performances dégradées soient admissibles [Blanke et al., 2003, Zhang and Jiang, 2003], la stabilité du système ne saurait être assurée. Cela souligne donc la position stratégique que tient le processus de détection et de diagnostic de faute au sein d'une démarche de tolérance active. Par ailleurs il faut aussi remarquer que la quasi totalité des travaux développés reconfigurent le contrôleur sans prendre en considération ses interactions avec le module de DDF [Dumont, 2006]. Ce n'est qu'assez récemment que certaines études ont envisagé cette problématique [Chen and Jiang, 2005, Jiang and Chowdhury, 2005].

La tolérance aux fautes permet donc d'obtenir une bien meilleure couverture de fautes que la tolérance passive. Cependant, on doit aussi s'interroger sur la capacité du contrôleur à fonc-

tionner en totale autonomie, en toute circonstance. Face à une situation imprévue, il peut être judicieux d'intégrer l'Homme dans la boucle de contrôle pour garder la maitrise du système.

2.5.3 Recouvrement par interaction Homme-Machine

Dans le but de proposer une stratégie de recouvrement permettant d'utiliser les capacités cognitives humaines, nous devons nous intéresser à l'intégration d'opérateur humain dans le fonctionnement de système autonome. Cette notion d'interaction entre l'entité humaine et le système artificiel (chaine de production, avion, robot, etc.) relève d'un domaine en plein effervescence. Nous tenterons ici, en nous appuyant sur quelques acteurs du domaine, de présenter les notions d'autonomie, d'interaction Homme-Machine et les différents types d'interaction homme-robot permettant de répondre à notre objectif l'utilisation du recouvrement par ajustement du niveau d'autonomie pour permettre de tolérer les fautes.

2.5.3.1 L'autonomie

L'autonomie est un concept ambigu, difficile à cerner et à préciser [Dalgalarrondo, 2003, Legras, 2007, Goodrich and Schultz, 2007, Moreno et al., 2008]. Une première définition qui peut être retenue est celle proposée par [Braynov and Hexmoor, 2001] : « Le concept d'autonomie est apparenté à une capacité individuelle ou collective de décider et d'agir de manière cohérente sans contrôle ou intervention extérieure ». Dans ce travail les auteurs la définissent comme une relation impliquant un sujet (devant être autonome), des influençants (sur l'autonomie du sujet), une portée (les moyens d'influence), un objet (vis-à-vis duquel le sujet peut être considéré comme autonome) et un degré (mesure de la possibilité d'intervention de l'influençant). Cette définition fait apparaitre l'importance de l'environnement du sujet, la notion de niveau d'autonomie mais aussi le caractère relatif de ce concept.

Steels dans [Steels, 1995] propose de définir un système autonome comme étant un dispositif capable de développer des lois et stratégies lui permettant de contrôler son comportement. Cette définition fait ici implicitement émerger la notion d'objectif et de maitrise comportementale des choix du système.

Dans [Clough, 2002] l'auteur proclame que l'intelligence est différente de l'autonomie, ce qui contredit les résultats de travaux du NIST [Meystel et al., 2000], et retiens deux citations pour la définir :

- « la capacité de générer ses propres fins sans aucune instruction de l'extérieur », de L. Fogel [1]
- « avoir le libre arbitre », de lui même (B. Clough).

Enfin, on trouve dans [Dalgalarrondo, 2003] une caractérisation à connotation plus automaticienne qui définit l'autonomie comme étant la capacité à résister à des perturbations externes en utilisant ses ressources internes. Elle met en avant ici l'importance de la relation avec l'envi-

1. http ://www.asc-cybernetics.org/foundations/Fogel.htm

ronnement et donc la capacité d'adaptation que doit avoir tout système autonome de maintenir ses performances même en présence d'adversité.

On ne peut que constater que l'autonomie est un concept multiforme et difficile à cerner avec précision. En tout cas l'autonomie ne doit pas être associée à la notion d'indépendance ou d'autosuffisance [Chopinaud, 2007]. C'est pourquoi on est naturellement conduit à définir la notion de niveau d'autonomie en fonction des relations mises en place entre l'entité humaine et le système artificiel.

2.5.3.2 Les niveaux d'autonomie

Le travail présenté par Sheridan et Verplank [Sheridan and Verplank, 1978] et complété dans [Parasuraman et al., 2000] propose une échelle comprenant 10 niveaux caractérisant la capacité d'autonomie d'un système (table 2.1). Cette dernière évolue continument de l'autonomie totale (niveau 10) sans aucune référence à l'Homme à celui où le système est complètement contrôlé par l'entité humaine (niveau 1) sans assistance de la machine. Les niveaux intermédiaires s'organisent en fonction de l'agent en charge de la prise de décision (niveaux 2 à 4), et de la façon dont elle est appliquée par le système (niveaux 5 à 9).

BASSE	1	n'offre aucune assistance.
	2	présente un ensemble complet de possibilités,
	3	présente un ensemble restreint de possibilités,
	4	suggère une possibilité,
	5	exécute sa suggestion si l'humain approuve,
	6	accorde un délai pour un éventuel véto avant l'exécution,
	7	agit puis informe l'humain,
	8	informe l'humain s'il le lui demande,
	9	informe l'humain s'il (l'ordinateur) le décide,
HAUTE	10	L'ordinateur décide de tout, agit seul, ignore l'humain,

Table 2.1 – Les niveaux d'autonomie proposés dans [Parasuraman et al., 2000]

L'importance de la position de l'Homme vis-à-vis du système artificiel dans le mécanisme de prise de décision est une question centrale dans la définition du niveau d'autonomie. Cela conduit à la définir en fonction des relations mises en place entre ces deux entités. [Wickens et al., 1998] et [Parasuraman et al., 2000] proposent donc de détailler le niveau d'autonomie en l'évaluant pour chacune des étapes de traitement de l'information par un être humain, à savoir : le traitement sensoriel (acquisition de l'information), la perception (analyse de l'information), la décision (sélection de l'action) et la réponse (mise en œuvre). Nous retrouvons d'autre études de l'interaction Homme-Machine basées sur des axes relativement équivalents [Clough, 2002, Endsley and Kaber, 1999]

D'autres travaux sur l'évaluation des niveaux d'autonomie dans le champ spécifique de la

robotique autonome sont aussi menés par le groupe de travail ALFUS [Huang et al., 2004] pour définir une grille multi-axes pour qualifier l'autonomie des systèmes autonomes. De plus, les travaux de l'Air Force Research Laboratory permettent de déterminer les niveaux de contrôle autonome (Autonomous control level) [Clough, 2002] et ceux de [Hasslacher and Tilden, 1995] donnent une mesure de la *Mobilité*, l'*Acquisition* et la *Protection* des robots biomorphes à l'aide de l'échelle tri-dimensionnelle MAP.

L'utilisation de différents niveaux d'autonomie permet d'automatiser le plus possible les tâches des opérateurs mais il reste nécessaire de garantir le fonctionnement d'un système autonome ou partiellement automatisé. L'étude de l'efficacité de l'interaction Homme-Machine est utile à l'évaluation de ce fonctionnement et permet de trouver des solutions d'amélioration.

2.5.3.3 Efficacité de l'interaction Homme-Machine

De nombreux articles reportent [Goodrich et al., 2001, Kaber and Endsley, 2004] que l'efficacité de la réalisation d'une mission en utilisant l'interaction homme-machine est fortement dépendante de l'implication de l'opérateur dans la réalisation de la mission et par conséquent de la qualité de sa perception de la situation (*situation awareness* [Chalandon, 2003]). En effet, un opérateur sera d'autant plus efficace pour les prises de décisions qu'il aura une compréhension exacte du contexte du robot, de son environnement et des ressources disponibles.

La surveillance aérienne ainsi que la protection des territoires étant très critiques, l'utilisation de systèmes autonomes reste extrêmement complexe [Taylor, 2006, Galster et al., 2007]. Par exemple, dans [Legras, 2008], l'auteur mesure l'oisiveté d'opérateurs humains lors de la supervision de système de surveillance de zone sécurisé et permettant l'interception des intrus.

Dans [Scholtz, 2003], l'auteur présente les différents rôles qu'un humain pourrait jouer en interagissant avec un robot. Elle discute ensuite des éléments nécessaires à l'homme pour lui garantir une bonne perception de la situation du robot.

2.5.3.4 Types d'interaction Homme-Robot

Dans la littérature on rencontre de nombreux types d'interaction Homme-Robot, et dans le domaine de la robotique les classes d'interactions suivantes (non nécessairement disjointes) sont présentées [Dalgalarrondo, 2003] :

- La **commande déportée** (*teleoperation*) où l'opérateur contrôle à distance et en permanence le système en assurant l'analyse des données capteurs et la prise de décision.
- Le **contrôle supervisé** (*supervisory control*) où l'utilisateur décompose le comportement du système en un ensemble de tâches qui seront ensuite exécutées de façon autonome, sauf en cas de problème [Mercier et al., 2009, Miller et al., 2005b].
- L'**échange de contrôle** (*traded control*) [Kortenkamp et al., 1997] lorsque l'entité humaine peut intervenir momentanément pour aider le système artificiel. L'Homme peut alors être vu comme une ressource cognitive supplémentaire pouvant assurer une fonctionnalité déficiente.

- Le **contrôle partagé** (*shared control*) [Röfer and Lankenau, 1999] où l'entité humaine a en charge de façon permanente certaines fonctionnalités du système.
- Le **contrôle coopératif** (*cooperative control*) [Fong et al., 1999] où le système artificiel et l'Homme dialoguent en cas de difficultés. Ce dernier est considéré comme une ressource particulière dont le système peut négliger les informations s'il le décide.
- L'**initiative mixte** (*mixed initiative*) [Allen, 1999, Adams et al., 2004] où l'entité humaine et le système joignent leurs capacités de façon à atteindre un résultat qu'ils n'auraient pu atteindre individuellement.

L'utilisation de l'**autonomie ajustable** (*ajustable autonomy*) permet de modifier le niveau d'autonomie du robot en fonction de la situation rencontrée. Deux types d'autonomie ajustables [Opperman, 1994, Miller et al., 2005a] différents sont dissociés suivant l'entité qui prend la décision de l'ajustement :

- L'**autonomie adaptable** (*adaptable autonomy*) permet de modifier les niveaux d'autonomie du système par l'opérateur lorsqu'il le décide.
- L'**autonomie adaptative** (*adaptive autonomy*) est utilisée lorsque le système à la capacité de modifier son niveau d'autonomie en fonction d'un modèle de la charge de travail et des capacités de l'opérateur. Dans [Parasuraman et al., 2007] les auteurs mettent en avant que ce type d'adaptation permet d'augmenter à la fois les capacités de l'opérateur pour se représenter l'environnement du système, sa situation réelle et permettre de diminuer sa charge de travail.

Pour finir l'interaction entre l'entité humaine et un système artificiel nécessite la mise en place d'un ensemble de mécanismes contextuels permettant à l'Homme d'une part de disposer des informations pertinentes nécessaires (état du système, mode de fonctionnement, dysfonctionnement supposé ou attesté, etc.) sous une forme qu'il puisse rapidement analyser et d'autre part d'interagir efficacement avec le système artificiel (langage naturel, gestes, interface haptique, etc.) [Calefato et al., 2008].

2.6 Analyse et conclusion

Ce tour d'horizon des méthodes et techniques utilisées pour la détection, l'isolation, l'identification puis le recouvrement de fautes démontre le foisonnement de travaux liés au développement de systèmes de contrôle tolérants aux fautes. Il est cependant possible d'en dégager certaines limitations et lignes directrices.

Tout d'abord l'analyse de l'ensemble de ces travaux permet d'identifier certains concepts scientifiques centraux qui nous semblent incontournables pour le développement de SCFT en robotique mobile autonome. En particulier la notion de résidu reste pour nous l'une des clefs du déploiement de la tolérance aux fautes en facilitant la détection de l'occurrence des fautes. Il faut aussi souligner l'importance du concept de redondance fonctionnelle, qu'elle soit matérielle ou logicielle, qui reste l'un des piliers de la tolérance aux fautes. D'une part la redondance est

utile pour la construction de résidus et donc indispensable au processus de diagnostic, et d'autre part elle peut faciliter la mise en place de solutions de recouvrement. Par ailleurs l'ensemble de ces travaux souligne le caractère critique du processus de détection, diagnostic et recouvrement de faute qui doit être effectué en temps réel pour assurer la stabilité du système. Enfin, il nous semble essentiel de retenir qu'il existe deux principales classes d'approches de recouvrement. Celles passives qui ne modifient pas fondamentalement le contrôleur développé, et celles actives qui, au contraire, changent la structure du contrôleur pour faire face aux dysfonctionnements rencontrés. Sans oublier l'adaptation du niveau d'autonomie qui peut être une réponse pour maintenir le système en fonctionnement même non optimal.

Par ailleurs il nous paraît important de souligner qu'il n'existe pas de méthodes de conception de SCFT communément acceptées et utilisées faute d'une prise en compte globale de la problématique. En effet, pour des raisons historiques, les travaux de recherche relevant de la détection et du diagnostic de fautes d'une part, et du recouvrement d'autre part, ont le plus souvent été élaborés séparément. Par ailleurs les équipes de recherche se focalisent en général sur la construction d'une approche spécifique, certes performante, mais qui ne couvre que certains aspects de la problématique et qui donc manque très fortement de généricité, de flexibilité et d'adaptabilité. Ces travaux gagneraient en efficacité si une démarche plus holistique était proposée.

Enfin est essentiel de relever que tous ces travaux, sans exception, se focalisent exclusivement sur les fautes pouvant affecter les capteurs, les actionneurs et le système physique contrôlé. A aucun moment ils n'envisagent que le contrôleur lui-même, où l'environnement qui supporte son déploiement, soit le siège de dysfonctionnements rendant pourtant caduque, par la même, toute la logique de tolérance aux fautes déployée.

C'est pourquoi nous allons maintenant nous intéresser, dans le cadre de la robotique, aux architectures logicielles qui constituent le centre névralgique de contrôle de tout robot. Elles permettent de déployer le contrôleur souhaité ainsi que les principes de sûreté de fonctionnement envisagés lors de la conception.

Chapitre 3

Architectures de contrôle robotiques et sûreté de fonctionnement

Cette partie se focalise sur le domaine de la robotique. Après avoir précisé la notion d'architecture de contrôle et présenté les grandes classes architecturales rencontrées nous allons analyser comment les principes de sûreté de fonctionnement sont effectivement mis en œuvre au sein des architectures robotiques et d'identifier quelles en sont, de notre point de vue, les principales limitations actuelles.

3.1 L'architecture de contrôle robotique

3.1.1 La notion d'architecture : Définitions

De façon intuitive il est possible d'énoncer que l'architecture désigne l'agencement des différentes parties d'un système logicielle et matérielle et donc l'expression des relations entre celles-ci. L'architecture matérielle peut être définie comme étant une représentation abstraite d'un composant électronique et/ou électromécanique pouvant accueillir un programme fixé ou pouvant évoluer [Zimmermann et al., 2008]. Même s'il n'est pas possible d'occulter la dimension matérielle d'une architecture qui influence largement l'implémentation informatique, en robotique la notion d'architecture recouvre le plus souvent principalement la seule dimension logicielle. On peut alors retenir la définition énoncée dans [Mataric and Michaud, 2008] et qui propose comme synonyme **Contrôle du robot** (*Robot Control*) où **Prise de décision du robot** (*Robot Decision-Making*) et la définit ainsi : « L'architecture de calcul d'un robot désigne le processus d'extraction d'informations sur l'environnement à partir des capteurs du robot, et si nécessaire le traitement de ces informations pour décider comment agir, ainsi que l'exécution de ces actions dans l'environnement ».

D'autres auteurs [Kortenkamp and Simmons, 2008] considèrent enfin que la notion d'architecture robotique recouvre en fait deux aspects liés mais distincts. D'une part, la **Structure** de l'architecture qui permet de décrire comment le système est divisé en sous-systèmes et comment

ces derniers interagissent. D'autre part le **Style** de l'architecture qui se rapporte aux concepts sous-jacents et particulièrement aux mécanismes de communications employés.

Comme toutes les architectures logicielles, une architecture de contrôle se doit de posséder, si possible, un ensemble de propriétés.

Cependant elle comporte aussi un ensemble de spécificités propres qui sont liées au fait qu'un robot doit interagir en temps réel et de façon asynchrone avec un environnement souvent incertain et dynamique tout en proposant potentiellement des temps de réponse à des échelles temporelles très différentes (de la milliseconde à l'heure) en fonction de la complexité des tâches à exécuter [Kortenkamp and Simmons, 2008].

L'efficience d'une architecture dépend évidemment d'un ensemble de principes de développement relevant de génie logiciel (modularité, maintenabilité, testabilité, etc.) mais aussi de ses capacités d'adaptation à l'environnement matériel du robot (capteur, actionneur) [Orebäck and Christensen, 2003].

Après avoir précisé la signification de la notion d'architecture et souligner son expression et ses particularités dans le domaine de la robotique nous allons maintenant balayer les principales classes architecturales robotiques développées.

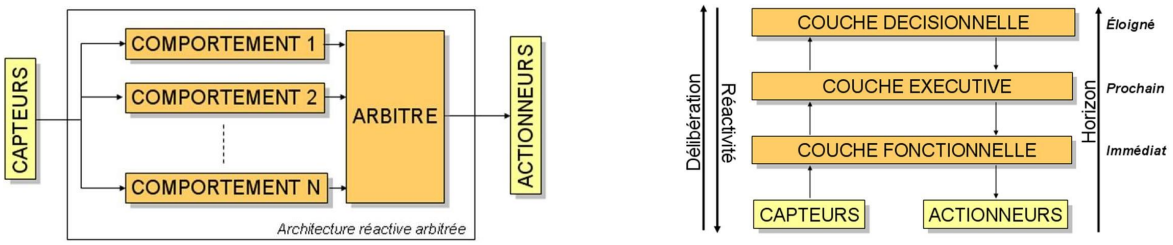
3.1.2 Les architectures de contrôle robotiques : La typologie classique

On peut distinguer deux grandes classes de robotique [Mataric and Michaud, 2008]. D'une part celle liée à des dispositifs placés dans des environnements complexes et fortement dynamiques (*Situated Robotic*). D'autre part, à l'opposé, celle s'appliquant à des robots évoluant dans des environnements pouvant être certes complexes mais fortement structurés (robotique d'assemblage par exemple). La complexité et le niveau de prédictibilité de l'environnement dans lequel évolue le robot à donc un impact majeur sur la complexité de son architecture. C'est pourquoi conceptuellement, compte tenu du rapport de l'architecture de contrôle avec la dynamique de l'environnement et de l'organisation adoptée, trois grandes classes architecturales ont été proposées en robotique, les architectures réactives, délibératives et hybrides.

Les architectures réactives (ou comportementales) n'utilisent pas de modèles de l'environnement du robot mais s'appuient sur un ensemble de comportements réflexes s'exécutant concurremment et de façon cyclique (voir figure 3.1a). Chaque comportement réagit aux données capteurs (vecteur d'état) pour générer la commande correspondante. Évidemment pour cette classe architecturale la difficulté majeure est de combiner l'ensemble des comportements de façon à obtenir une réaction compatible à la fois avec la dynamique de l'environnement et l'objectif assigné au robot. Pour tenter d'y parvenir un module d'arbitrage est le plus souvent ajouté en aval des modules comportementaux pour construire le vecteur de sorties à imposer aux actionneurs. Brooks dans ses travaux [Brooks, 1986] propose l'architecture de subsumption où les comportements sont hiérarchisés des plus aux moins prioritaires, les derniers pouvant inhiber les premiers. Des mécanismes d'arbitrage plus complexes ont été proposés dans l'architecture DAMN [Rosenblatt, 1997] où des poids, qui évoluent dynamiquement en fonction de

la situation rencontrée, sont affectés à chaque comportement pour construire la commande des actionneurs. L'architecture Aura [Arkin, 1987] fait appel à la notion de schémas moteurs (entité indépendante fonctionnant en parallèle) [Arbib, 1981] associés à des comportements élémentaires pour construire par sommation le vecteur de commande à appliquer.

Cette classe architecturale s'avère très performante pour faire émerger un comportement global permettant d'évoluer dans un environnement fortement dynamique. Cependant si le principe proposé de composition comportementale semble simple, sa flexibilité reste limitée car l'ajout de nouveaux comportements impacte fortement l'ensemble de l'architecture. De plus en l'absence de planification et de modélisation de l'environnement il est difficile de mener à bien des missions complexes.



(a) Réactive (b) Délibérative

Figure 3.1 – Schéma de principe des principales classes architecturales robotiques

A l'inverse des précédentes, les architectures délibératives (ou hiérarchisées) s'appuient sur une décomposition des mécanismes décisionnels [Albus et al., 1989, Albus et al., 2002, Gat, 1998]. Au dessus de la couche physique, elles s'organisent en général en trois couches associées à des niveaux de prise de décision et à des horizons temporels différents (voir figure 3.1b). Par ailleurs, structurellement, une couche ne peut communiquer qu'avec celle immédiatement voisine. On distingue :

- La couche fonctionnelle, située à l'interface avec la couche physique, qui contient les modules de perception, de commande et de génération de trajectoire du robot.
- La couche exécutive qui supervise l'exécution des tâches robotiques et dirige l'exécution des modules de la couche fonctionnelle impliqués.
- La couche décisionnelle qui planifie l'enchainement des tâches robotiques à effectuer pour atteindre les objectifs imposés à la mission.

Cette classe architecturale permet une bonne maitrise du comportement du robot en fonction des objectifs définis. Cependant sa principale limitation provient évidemment de son manque de réactivité puisque le flux d'informations capteurs doit traverser toutes les couches pour atteindre le niveau décisionnel avant, qu'en sens inverse, l'adaptation choisie ne soit à son tour propagée vers les actionneurs. Ce temps de latence est incompatible avec une utilisation temps réel dans un environnement dynamique. Pour palier à cette limitation des architectures hybrides ont été développées.

Les architectures hybrides (ou mixtes) concilient les avantages des architectures réactives et délibératives. Pour y parvenir elles complètent la hiérarchisation des architectures délibératives par des boucles de réaction entre les différentes couches améliorant par la même la réactivité de chacune d'elles (figure 3.2.

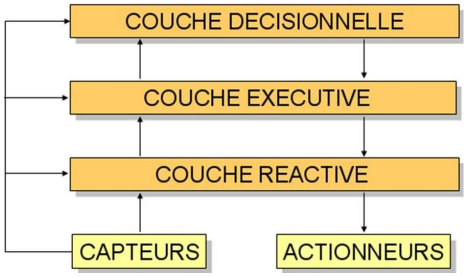


Figure 3.2 – Schéma de principe des architectures hybrides

La plupart des architectures robotiques relèvent de cette classe architecturale. Par exemple ORCCAD (Open Robot Computer Aided Design) [Borrelly et al., 1998, Simon et al., 1997] propose une architecture décomposée en deux couches dédiées aux applications robotiques. La couche application pour représenter les tâches complexes et la gestion de la mission fait appel à des Procédures Robotiques. Ces dernières sont décomposées au niveau de la couche commande en tâches robotiques plus élémentaires (*Robot-Task*). L'adaptation du contrôle peut s'effectuer au niveau requis en fonction de l'évolution de la mission. CLARATY (*Coupled Layer ARchitecture for Robotic Autonomy*) [Volpe et al., 2001, Volpe et al., 2000] (figure 3.5), dédiée elle aussi à la robotique, découpe l'architecture en deux niveaux. La couche fonctionnelle s'appuie sur une approche objet ce qui permet la définition de frameworks et de classes génériques facilitant la réutilisation du code. La couche décisionnelle assure la planification et l'exécution de la mission. L'architecture du LAAS [Alami et al., 1998, Ingrand, 2006] (figure 3.4) est une des architectures les plus abouties. On y retrouve les trois niveaux classiques :

- la couche fonctionnelle à l'interface de la couche physique qui comprend un ensemble de modules proposant un ensemble de services via des requêtes asynchrones
- un niveau contrôle d'exécution qui a en charge la gestion et l'exécution des tâches élaborées par le niveau décisionnel
- le niveau décisionnel définit et supervise l'exécution des plans nécessaires à la réalisation de la mission.

Les architectures robotiques sont supportées par des middlewares logiciels aux caractéristiques (OS, temps-réel, langage et style de programmation, mécanismes de communication, outillage, etc.) très hétérogènes [Passama, 2010] mais dont les performances sont pourtant essentielles pour faciliter et garantir le développement d'architectures de contrôle efficientes. Cependant, ces dernières mettent-elles effectivement en œuvre les principes nécessaires au déploiement de la sûreté de fonctionnement ?

3.2 Prévention des fautes

La prévention des fautes fait appel comme nous l'avons dit en introduction à l'ensemble des techniques utilisées dans le domaine de l'ingénierie des systèmes pour leur développement sans fautes.

Elle peut prendre en compte la dimension matérielle lors de la conception d'un robot. Par exemple, le confinement d'erreur par partitionnement matériel est mis en œuvre dans l'architecture du robot RoboX [Tomatis et al., 2002] et consiste à faire appel à deux processeurs différents :

- l'un dédié aux tâches critiques de localisation et déplacement et dont l'élaboration est maitrisée par les chercheurs,
- l'autre associé aux tâches interactives qui utilisent des composants logiciels sur étagère pour lesquels les développeurs n'accordent que peu de confiance.

Mais la prévention des fautes adresse essentiellement la dimension logicielle de l'architecture en préconisant un développement modulaire des systèmes et l'utilisation d'outils de développement spécifiques [Lussier et al., 2005].

3.2.1 Modularité des systèmes

L'utilisation d'un développement modulaire permet de décomposer le système global en différents sous systèmes indépendants appelés **modules**, ce qui permet de faciliter le développement au sein de l'architecture :

- la modularité permet de dissocier dans le développement d'un module, sa structure générique du cœur de métier qu'il implémente :
  - la structure générique du module est conçu comme une boite gérant notamment les communications entre modules et l'activation de celles-ci.
  - Le cœur de métier correspond à l'algorithme robotique implanté en son sein.

  Ceci permet de faciliter la maintenance et l'évolutivité de chaque partie du module.
- La modularité permet le développement et le test unitaire de chaque module. Un intervenant externe à l'architecture, par exemple un roboticien, a la possibilité d'implanter et tester son algorithme au sein du module avant de l'intégrer au sein de l'architecture.
- La réutilisabilité des modules permet de composer une architecture de contrôle comme l'assemblage de composants dits sur étagère. Ceci permet d'utiliser des algorithmes et des fonctionnalités éprouvés.

Le principe de modularité est très largement dépendant de l'utilisation d'une couche logicielle nommée « middleware ». C'est une infrastructure d'exécution [El Jalaoui, 2007] commune à tous les modules qui génère une abstraction du système d'exploitation, une communication inter-modules, et la gestion de l'activation de ceux-ci. L'abstraction du système d'exploitation permet une plus grande portabilité de l'architecture vers un autre système d'exploitation.

Le développement modulaire ou par agent autonome ne contraint ni le type organisationnelle (réactive, délibérative ou hybride) de l'architecture, ni le mécanisme d'activation de ses modules. Ainsi, le concept d'agent autonome exécutant une tâche est présent dans les architectures RAX [Feather and Smith, 1999] et IDEA [Muscettola, 1998, Muscettola et al., 2002] où chaque agent dispose de son propre contrôleur lui permettant de gérer sa planification, son activation, ses propres outils de diagnostic ainsi que ses ressources (voir figure 3.3). Par contre, dans COTAMA [El Jalaoui, 2007] un ordonnanceur dédié assure le séquencement de l'activation de l'ensemble des modules impliqués dans la boucle de contrôle courante.

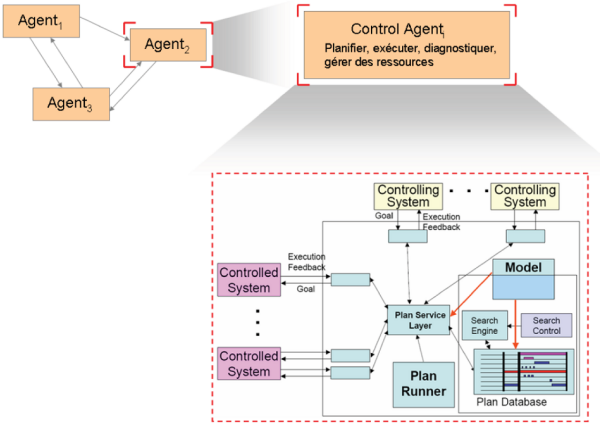


Figure 3.3 – Agent de contrôle de l'architecture IDEA [Diaz et al., 2003, Muscettola et al., 2002]

La modularité est ainsi présente dans la plupart des architectures robotiques comme celles du LAAS [Alami et al., 1998], CIRCA [Goldman et al., 1997] et ORCCAD [Borrelly et al., 1998] et CLARATY [Volpe et al., 2001].

Le développement modulaire fait le plus souvent appel à de nombreux outils de conception facilitant, structurant et rendant plus sûr le développement des architectures de contrôle.

3.2.2 Outils de conception des systèmes

Les outils de conception ont pour but de faciliter l'implémentation des contrôleurs. Ce sont des environnements spécifiques de travail où les informations et fonctionnalités nécessaires à la construction des contrôleurs sont présentes, à savoir :

- Une bibliothèque des modules développés et fournis qui permet de générer l'architecture souhaitée.
- Un détecteur d'erreurs de combinaison ou d'interaction de modules, qui facilite leur association tout en évitant des erreurs humaines dues à des compositions incompatibles ou incomplètes.
- La génération automatique du code exécutable à partir de l'organisation architecturale modulaire définie par le concepteur qui minimise les erreurs humaines de traitement.

Les outils de conception des architectures peuvent se décliner sous la forme de différents environnements de développement en fonction des objectifs visés. Par exemple, CONTRACT propose une méthodologie globale de conception et de développement d'architectures de contrôle de robots [Passama, 2010]. Celle-ci permet de générer l'ensemble du code de l'architecture de contrôle en s'appuyant sur un middleware robotique et sur un outil logiciel (IDE - *Integrated Development Environment*) permettant de décrire textuellement le comportement de l'architecture. Un outil de vérification permet de l'analyser et de reporter, le cas échéant, les erreurs de conception à l'utilisateur. Pour sa part, l'environnement de planification distant PILOT [Nana et al., 2005] est doté d'un mécanisme d'édition permettant de garantir la validité syntaxique d'un plan à chaque phase de sa construction (insertion, modification, suppression de primitives). Grâce à la nature interprétée du langage, les plans générés demeurent modifiables dynamiquement tout en assurant leurs terminaisons. Des logiciels dédiés sont aussi souvent employés pour décrire le niveau fonctionnel des systèmes robotisés et en limiter les erreurs. C'est par exemple le cas des environnements de programmation G^{en}oM de l'architecture LAAS [Fleury et al., 1997], CONTROLSHELL [Schneider et al., 1998], ORCCAD [Borrelly et al., 1998] et de l'outil utilisant le langage synchrone SIGNAL proposé par [Marchand et al., 1998].

L'apparition et la création des fautes sont donc fortement réduites par l'usage de ces outils de développement sophistiqués qui en combinaison avec la modularité permettent d'imposer un cadre spécifique de développement.

Le nombre de fautes introduites lors de cette phase est limité sans pour autant en garantir l'absence totale. De plus ces outils ne concernent qu'un sous-ensemble des fautes, les fautes de conception. La dernière phase du développement d'un système consiste donc à vérifier qu'il fonctionne correctement et sans faute. Les techniques utilisées en robotique pour éliminer celles encore présentes sont détaillées dans la section suivante.

3.3 Élimination des fautes

En robotique, l'élimination des fautes regroupe l'ensemble des moyens pouvant être mis en œuvre lors du cycle de développement du robot, pour détecter et corriger les fautes encore présentes.

3.3.1 Test et simulation

Ce cadre d'évaluation du fonctionnement d'un robot fait appel à deux approches complémentaires. D'une part la simulation est utilisée pour s'assurer du bon fonctionnement des composants matériels et logiciels. D'autre part des campagnes spécifiques de tests expérimentaux permettent de valider le fonctionnement du robot dans son ensemble. Cette dernière démarche reste difficile à mettre en œuvre (hors de la robotique industrielle) faute des moyens et de l'infrastructure nécessaires à son déploiement.

A notre connaissance, un des seuls exemples ayant donné lieu à publication est celui du projet RAX DS1 qui s'appuie sur l'architecture REMOTE AGENT [Bernard et al., 2000]. La simulation et le test intensif ont été déployés pour chercher à détecter l'ensemble des fautes d'un système spatial robotisé. Six bancs d'essai ont été mis en œuvre tout au long du processus de développement, et ont permis de conduire pas moins de 600 tests. Les auteurs soulignent la pertinence de telles campagnes intensives d'évaluation mais en reconnaissent la difficulté d'application. Entre autre ils pointent le problème lié à l'exhaustivité du test.

En effet, cette exhaustivité est purement utopique dans des systèmes robotiques d'une telle complexité. La simulation et le test restent majoritairement focalisés sur les composants et algorithmes les plus critiques. La validation formelle permet cependant de s'inscrire dans une perspective plus globale et générique pour garantir un ensemble de propriétés de bon fonctionnement à l'aide d'outils mathématiques.

3.3.2 Validation formelle

La validation formelle rassemble un ensemble de techniques utilisant des outils mathématiques, souvent sophistiqués, pour valider le comportement d'un système sur un ensemble de propriétés données. Cette dimension doit être prise en compte dès le début de la conception du système en choisissant des modèles mathématiques adaptés à la formalisation et à la preuve des propriétés visées pour le système étudié.

Certains outils de conception (présentées section 3.2.2) d'architectures de contrôle intègrent donc à la fois des moyens de modélisation pour construire le système et de vérification formelle pour éliminer les fautes.

Par exemple, l'architecture ORCCAD est une des architectures mettant l'accent sur l'élimination des fautes en intégrant dans ses outils de conception des moyens de validation formelle. Elle fait appel au langage ESTEREL pour la spécification de la partie contrôle des applications [Espiau et al., 1995] ce qui permet l'utilisation des outils de vérification formelle associés à ce langage synchrone. Un autre langage synchrone, le langage RMPL (*Reactive Model-Based Programming Language*) [Williams et al., 2003b] a été développé pour permette une programmation basée modèle. Il a été utilisé dans le noyau d'exécution TITAN mis en place sur la sonde DS1 de la NASA [Williams et al., 2003a] afin de permettre le diagnostic des fautes au sein des systèmes autonomes réactifs. L'architecture LAAS s'appuie quand à elle sur le langage à composants BIP (*Behavior-Interaction-Priority*) [Basu et al., 2006] pour assurer la validation formelle de propriétés de sûreté (absence de blocage par exemple) pour le contrôle d'exécution du niveau fonctionnel [Basu et al., 2008]. Dans [Rutten, 2001] l'auteur propose aussi une méthode pour la programmation sûre d'applications robotiques s'appuyant, pour synthèse des contrôleurs, sur l'utilisation d'une boite à outils faisant appel au langage synchrone SIGNAL [Le Guernic et al., 1991].

D'autres langages formels sont utilisés en robotique lors de la conception comme par exemple le langage *Symbolic model checking* SMV [Burch et al., 1992]). Dans [Simmons et al., 2000] un mécanisme de traduction du système robotique étudié permet de le décrire formellement en

langage SMV pour qu'il puisse être analysé grâce à des outils dédiés. Dans l'architecture CIRCA [Goldman et al., 2000], le planificateur SSP (STATE SPACE PLANER) utilise les automates temporisés [Alur, 1998] pour vérifier la vivacité d'un plan et le corriger en ligne si nécessaire.

Malheureusement, la validation formelle définit un cadre de développement et de modélisation spécifique et contraignant qui impose des limitations aux architectures. La plupart du temps les langages formels, de par leur spécificité, ne sont pas directement utilisés lors du processus de conception de l'architecture. Il est possible d'obtenir, dans un second temps, après une étape de modélisation, une représentation formelle du comportement du système robotisé afin de pouvoir le valider.

Par rapport à la simulation, la validation formelle présente l'immense avantage de pouvoir assurer qu'une propriété attendue, correctement formulée, est satisfaite quelques soient les configurations comportementales envisageables du système analysé. Cependant la complexité comportementale des robots et la dimension temporelle associée induit une explosion combinatoire des états qui limite fortement la portée de la validation formelle. Par ailleurs, les fautes de type capteurs, actionneurs, ou encore celles dues à une mauvaise représentation de l'environnement sont rarement prises en compte dans les travaux actuels.

Malgré toutes les techniques que nous venons de balayer pour la prévention et l'élimination des fautes il est impossible d'éviter leur occurrence lors du fonctionnement d'un système robotisé. L'exhaustivité des méthodes d'élimination est encore une utopie, et de plus certaines fautes ne peuvent être évitées. Il est donc indispensable de disposer de moyens permettant de les identifier avant de pouvoir les détecter, les diagnostiquer et de réagir à leur présence.

3.4 Prévision des fautes

À notre connaissance peu de travaux ont étudiés les fautes affectant des robots autonomes et leurs impacts sur le déroulement de leur mission. Différentes démarches sont pour autant discernables. L'analyse de relevés sur de longues périodes d'expérimentation et l'utilisation d'approches globales pour l'identification des fautes pouvant affecter un système robotisé. Ces travaux sont complétés avantageusement par l'étude des niveaux de sévérité des fautes identifiées.

3.4.1 Analyses statistiques

Les travaux menés par Murphy [Carlson and Murphy, 2005] et son équipe, déjà évoqués en section 1.1, relèvent de ce type de démarche. En effet ils étudient le comportement d'un grand nombre de robots, le plus souvent dans des conditions d'utilisation réelles, au travers des rapports d'études disponibles. L'analyse statistique ainsi produite permet de proposer une taxonomie des fautes, d'en identifier les plus fréquentes, et de connaitre leur impact sur le fonctionnement du système robotisé.

Tomatis dans [Tomatis et al., 2002] présente un robot-guide ROBOX développé pour conduire la visite guidée du musée de Neuchâtel (Suisse). Dans [Tomatis et al., 2003] les au-

teurs présentent l'analyse de fiabilité de leur robot guide après un trimestre d'exploitation et souligne la fragilité du contrôleur développé.

Bien que pertinente et très enrichissante ce type d'approche expérimentale a postériori reste marginale et ne s'intègre qu'en fin du cycle de conception d'un robot. L'utilisation d'une démarche a priori, comme l'AMDEC, déployée dès le début de la phase de conception du robot est nécessaire pour l'élaboration d'une architecture tolérante aux fautes.

3.4.2 Application de l'AMDEC en robotique

L'AMDEC qui pourtant fournit un cadre d'analyse des défaillances d'un système éprouvé dans de nombreux domaines scientifiques, a peu été appliquée, à notre connaissance, en robotique.

On trouve un exemple d'application de l'AMDEC dans le cadre d'un projet de diagnostic de fautes en ligne d'un robot BEARCAT [Nikam and Hall, 1997]. Il a été mené par plusieurs élèves de Master [Kanakaraju, 2000, Vishnuvardhanaraj, 2000] de l'université de Cincinnati sous la conduite du Professeur Hall. Son but est de permettre d'avoir un accès en ligne, depuis un navigateur internet, à un système de diagnostic performant permettant à un opérateur d'analyser l'état du robot. Une variante de l'AMDEC, le PFMEA (*Potential Failure Mode and Effects Analysis*) est utilisée pour structurer et diriger le diagnostic humain en se basant sur les informations d'erreurs retournées par le robot. Les solutions envisagées pour pallier l'occurrence de défaillances restent purement matérielles.

Dans [Guiochet and Baron, 2004], l'auteur combine la méthodologie AMDEC et le langage UML pour maitriser la sécurité de systèmes robotiques médicaux. L'analyse réalisée permet d'identifier les risques liés à la téléopération d'un robot esclave TER. Les modes de défaillance sont alors classifiés selon leurs origines et un ensemble de solutions est proposé pour concevoir un nouveau robot plus sécurisé.

Nous allons maintenant évoquer l'analyse de sévérité des fautes qui fait partie intégrante de la démarche AMDEC et qui s'inscrit comme l'un des paramètres central du mécanisme de tolérance aux fautes déployé au sein des architectures robotiques.

3.4.3 Analyse de la sévérité des fautes

L'évaluation par niveaux de la sévérité des fautes au sein de contrôleurs robotiques peut prendre différentes formes.

Dans [Brandstötter et al., 2007] les auteurs proposent une discrimination simpliste de seulement deux classes de fautes, celles dont l'origine est connue et où une solution de recours a été modélisée, et celles sans origine déterminée où le robot doit être arrêté.

Dans [Ji et al., 2003] un contrôleur adaptatif hybride met en œuvre 3 niveaux de sévérité en fonction des solutions de recouvrement adoptées. Si la faute est considérée comme peu importante, un contrôle robuste est utilisé pour maintenir la performance du système. En présence d'une faute de niveau intermédiaire, un nouveau contrôleur est choisi et inséré dans la boucle

de commande. Enfin en cas de faute fatale, le système est stoppé.

Dans l'architecture PROCOSA [Barbier et al., 2006], dans le contexte du vol d'un drone, les auteurs déterminent 4 niveaux d'erreurs en fonction de l'incidence sur la mission en cours. Ils distinguent les évènements :

- qui conduisent à adapter la mission sans pour autant la remettre en cause
- qui identifie la coupure de communication forçant le système à passer en autonomie totale
- liés à des problèmes de sécurité exigeant un changement de plan de vol
- catastrophiques imposant l'abandon de la mission en cours

Enfin récemment [Olive, 2010] décrit les mécanismes de tolérance aux fautes déployés par la société Thalès au sein de satellites pour lesquels quatre niveaux de sévérité sont utilisés. Le plus faible correspond à des fautes sans impact sur les performances des sous-systèmes du satellite (erreur de parité, etc.) où le recouvrement peut se faire localement et de façon autonome. Le niveau suivant correspond aux fautes, sans conséquence sur le déroulement de la mission, mais nécessitant la commutation autonome vers une unité redondante. L'antépénultième niveau correspond à la perte de performance d'un sous-système. Dès lors le processeur impliqué est remplacé. Le niveau le plus critique est rencontré en cas d'alarmes multiples provenant du niveau précédent ou en présence de dysfonctionnements matériels critiques. La mission doit alors être interrompue et le satellite est positionné en mode sécurité. Le recouvrement est ensuite réalisé depuis le centre de commande sur Terre.

Ces nombreux exemples transcrivent de fait un lien plus ou moins direct entre le niveau de sévérité d'une faute et les solutions de recouvrement envisageables.

L'application de l'AMDEC ainsi que de l'ensemble des moyens de prévisions des fautes restent peu fréquent dans le contexte de la robotique. Pourtant ce type d'analyse nous semble cruciale pour connaitre les fautes pouvant potentiellement affecter le système robotisé. Nous pensons que cette étape est indispensable à la mise en place d'un processus de tolérance aux fautes ciblé et cohérent.

3.5 Tolérance aux fautes

Après avoir présenté l'ensemble des techniques permettant d'éviter la présence des fautes au début de ce chapitre, nous avons vu dans la section précédente que de nombreuses fautes pouvaient encore perturber les robots pendant l'exécution de leur mission. Aux fautes internes liées aux dysfonctionnements du robot viennent s'ajouter celles d'origine externe induites lors de l'évolution du système robotisé dans un environnement dynamique et perturbé. Ainsi, le concept de robustesse (*robustness*) souvent rencontré dans la littérature des architectures de contrôle se réfère à la tolérances aux fautes externes. [Lussier, 2007] la définie « *comme la délivrance d'un service correct en dépit de situations adverses dues aux incertitudes vis-à-vis de l'environnement du système* ».

Dans cette partie nous allons présenter et analyser les mécanismes logiciels mis en place au

sein des architectures de contrôle robotiques pour la tolérance aux fautes internes et externes. Après avoir abordé les méthodes de détection de fautes nous détaillerons les différents processus de recouvrement pouvant être mis en œuvre.

3.5.1 Les méthodes logicielles de détection de fautes

La classification retenue dans [Lussier et al., 2005] dans le domaine de la robotique décompose les méthodes logicielles de détection en quatre classes :

Contrôle temporel (*timing checks*) Ce type de mécanisme de détection, très largement utilisé en robotique, notamment sous la forme de « chien de garde » (*watchdog*), permet de détecter les déviations temporelles et les ruptures de service des sous-systèmes. On s'assure que l'élément surveillé ne reste pas bloqué dans un état indéterminé en vérifiant, par exemple, qu'il produise ses résultats dans un laps de temps imparti ou en respectant une fréquence préalablement définie. Par exemple dans [Tomatis et al., 2002] l'activité des différents composants importants de RoboX (contrôleur de vitesse, évitement d'obstacle, capteurs par-chocs et lasers) est contrôlée de cette façon. Ce mécanisme est aussi employé dans [Steinbauer et al., 2005] sur un robot participant à la Robocup 2005 pour vérifier que les composants de l'architecture respectent certaines contraintes temporelles protocolaires. Des watch-dogs accompagnent chacun des composants BIP [Basu et al., 2008] de l'architecture du LAAS. Grosclaude dans [Grosclaude, 2004] développe un ensemble d'indicateurs, basés sur une démarche générique d'observation du comportement des composants logiciels d'une architecture, dont l'analyse statistique permet de détecter la présence de dysfonctionnements temporels d'exécution.

Contrôle de vraisemblance (*reasonableness checks*) Les méthodes de contrôle de vraisemblance vérifient que les données produites respectent des domaines de validité spécifiés, dans le but de supprimer les valeurs aberrantes. Le respect des domaines de fonctionnement des capteurs spécifiés par les constructeurs, mais aussi le respect des valeurs produites par certains algorithmes sont regroupés ici. Cette technique de détection est couramment employée par les développeurs au sein des architectures de contrôle. Malheureusement, la plupart du temps, elle ne s'intègre pas dans une démarche structurée, mais est plutôt intégrée au fil de l'eau, dans le code, en fonction des besoins des algorithmes de l'application.

Les seuls exemples à notre connaissance explicite dans la littérature sont le ROBOX proposé dans [Tomatis et al., 2002] et les robots médicaux non-autonomes HIPPOCRATE et SCALPP présenté dans [Duchemin et al., 2004].

Contrôle de commande (*safety bag checks*) Les méthodes de contrôle de commande vérifient la validité des données en sortie d'un système pour bloquer les valeurs erronées et stoppent ainsi leurs propagations dans le système. Tout comme le contrôle de vraisemblance ces mécanismes sont le plus souvent intégrés directement au cours du dé-

veloppement.Un exemple est donné au sein de l'architecture LAAS le contrôleur R2C [Py and Ingrand, 2004] permet de déployer le contrôle de commande par tests d'assertions ou de propriétés sur le niveau fonctionnel et de positionner le robot dans une configuration sure lorsqu'un dysfonctionnement est détecté.

Duplication et comparaison (*Monitoring for diagnosis*) L'utilisation de la redondance physique étant trop couteuse pour détecter l'occurrence de défauts matériels le recours à un modèle du système est souvent employé pour détecter la présence de fautes. Ce type de démarche est souvent déployé à travers des approches basées modèles. Par exemple dans [Brandstötter et al., 2007] les auteurs utilisent des modèles probabilistes hybrides pour détecter l'occurrence de fautes au niveau des roues d'un robot mobile participant à la RoboCup 2006. Les filtres de Kalman multi-modèles sont très souvent mis en œuvre pour la détection de dysfonctionnement matériels. Par exemple, dans [Roumeliotis et al., 1998] les auteurs proposent une méthode d'estimation multi-modèles de détection et de diagnostic des fautes. L'expérimentation proposée met en avant la détection de fautes sur les roues d'un robot mobile et le diagnostic de la roue incriminée. Un court état de l'art présentant à la fois l'historique, les limitations et les améliorations nécessaires aux méthodes de diagnostic multi-modèles est proposé dans [Aguiar, 2007].

L'utilisation de mécanismes de redondance dans les systèmes permet aussi de faire de la tolérance aux fautes et de proposer un niveau de sécurité plus élevé. Cependant ce principe reste cantonné à des domaines très critiques car principalement très couteux (aérospatiale, avionique) et n'a donc pas été évoqué en tant que tel dans la précédente classification.

L'analyse des mécanismes de détection de fautes mis en place au sein des systèmes robotisés démontre que leur mise en œuvre relève, la plupart du temps, d'une démarche développeur difficilement compatible avec la mise en place d'un processus structuré de prise en compte de la tolérance aux fautes. Plus généralement en robotique, la détection de fautes se révèle être un domaine d'étude particulièrement dynamique et foisonnant compte tenu du nombre de publications relevant de cette thématique [Duan et al., 2005].

En effet ces études visent des fautes dont la pertinence est loin d'être démontrée ou sont souvent décorrélées d'un processus de diagnostic. Ce dernier fortement mis en avant lors de la présentation générale de la sûreté de fonctionnement (section 2.4) est dans le contexte des architectures de contrôle quasiment inexistant. Nous allons donc maintenant aborder la dernière phase de la tolérance aux fautes en présentant les techniques de recouvrement déployées pour répondre à l'occurrence de dysfonctionnements au sein des architectures de contrôle.

3.5.2 Le recouvrement

La dernière phase de la tolérance aux fautes, le recouvrement, est l'action qui permet malgré l'occurrence de dysfonctionnements de poursuivre et réaliser les buts fixés. La réaction induite dépend de la nature de la faute et de l'impact qu'elle induit sur le système robotisé.

Pour présenter les principes de recouvrement autonomes nous allons tout d'abord évoquer les moyens logiciels pouvant être exploités dans le domaine de la sûreté de fonctionnement en les étayant de quelques exemples d'utilisation en robotique. Ensuite nous allons nous focaliser plus précisément sur les mécanismes employés au sein des architectures robotiques pour pallier à l'occurrence de dysfonctionnements. Pour finir nous présentons comment un opérateur distant peut être impliqué, lorsque cela est possible et nécessaire, dans le processus de recouvrement à mettre en œuvre.

3.5.2.1 Mécanismes de recouvrement logiciel en sûreté de fonctionnement

[Avižienis et al., 2004] présente une classification des méthodes permettant de faire du recouvrement dans le cadre de la tolérance aux fautes. Les auteurs le définissent comme *l'action de transformer l'état d'un système qui contient une ou plusieurs erreurs et (probablement) fautes dans un état sans les erreurs détectées et sans faute qui puissent être à nouveau réactivées.*

Nous allons maintenant présenter la classification des méthodologies de recouvrement proposée dans [Avižienis et al., 2004] pour traiter les fautes ou erreurs rencontrées dont des exemples pratiques peuvent être trouvés dans [Essamé et al., 2000].

3.5.2.1.a Recouvrement par traitement des erreurs

Le traitement des erreurs permet uniquement de fournir à un instant donné un service correct en sortie du système.

La reprise d'erreur (*rollback*) renvoie au dernier état stable du système pour tenter une poursuite. En robotique autonome les difficultés liées à cette technique sont nombreuses. En effet cette approche nécessite l'utilisation de sauvegardes de taille importante ce qui induit (entre autre) par la même de long temps de rétablissement. De plus, pour un système physique tel un robot le retour à un état antérieur n'est pas toujours envisageable.

Le traitement par poursuite (*rollforward*) prend le risque de continuer l'exécution du système un certain temps dans l'espoir d'atteindre un état stable ce qui permettra ensuite de repartir de celui-ci.

La compensation (*compensation*) masque les erreurs contenues dans l'état d'un système en utilisant la redondance de celui-ci. Par exemple, l'utilisation du code de Hamming [Hamming, 1959] permet de détecter une faute lors d'une transmission de données et de la corriger à l'aide de l'information redondante. Cette solution est comme toute celles basées sur la redondance efficace, mais couteuse (temps, place, argent). Elle reste cependant parfois la seule solution.

Dans le cas où aucune des solutions de recouvrement permettant de continuer la mission dans des conditions acceptables n'est éligible ou dans l'attente d'une solution en cours de calcul, la **mise en état sûr** du système est nécessaire pour garantir l'intégrité du système autonome

et de son environnement. Ce type de stratégie s'apparente au traitement par poursuite. Elle est utilisée dans les architectures LAAS [Lemai-Chenevier, 2004], RAX [Muscettola, 1998]et IDEA [Muscettola et al., 2002] lors d'un échec de planification ou après une défaillance majeure du système. Le robot ROBOX [Tomatis et al., 2003] se met aussi dans un état sûr lors de la défaillance d'un de ses sous-systèmes critiques avant qu'un opérateur soit appelé pour en effectuer la maintenance. De même le robot guide CARE-O-BOT [Graf, 2005] se positionne dans un état sûr lorsque l'un de ses boutons d'arrêt d'urgence est enclenché ou s'il sort de la zone de travail allouée.

Dans le contexte de la robotique et notamment face aux pannes matérielles, ces méthodes ne permettent cependant pas de traiter leur origine, il faudra dans ce cas engager un processus permettant le recouvrement de la faute.

3.5.2.1.b Recouvrement par traitement des fautes

Pour agir directement sur la faute à l'origine du dysfonctionnement observé, il est possible soit d'apporter une solution « réparatrice », soit de se priver des composants atteints par la faute, pour qu'elle ne s'active plus. Les différentes étapes du traitement des fautes sont décrites ci-dessous :

Le diagnostic est la première étape du traitement de fautes car il permet d'identifier et de localiser les fautes activées.

L'isolement permet de rendre dormante la faute détectée. Elle est isolée physiquement ou logiciellement de l'exécution de la boucle de contrôle-commande du robot. Par exemple le robot ROBOX [Tomatis et al., 2003] dispose de deux processeurs pour isoler physiquement, par soucis de prévention, sur l'un les tâches critiques temps réel et sur l'autre les tâches d'interaction avec les visiteurs du musée souvent sujettes à des bugs des applications multimédias.

La reconfiguration fait appel soit à l'utilisation de composants matériels ou logiciels redondants, soit à la réassignation de tâches sur des composants non fautifs. Dans l'architecture RAX [Feather and Smith, 1999] la redondance fonctionnelle des composants matériels permet de reconfigurer le robot pour réaliser la mission. Dans [Tomatis et al., 2003], ROBOX met en œuvre le concept de modularité pour basculer entre différents modes de fonctionnement dans le but de reconfigurer logiciellement le robot. De même les robots d'exploration martiens PATHFINDER10 et SPIRIT11 ont été reconfigurés à distance par des opérateurs pour les rendre à nouveau opérationnels. Comme nous l'avons déjà vu dans [Brandstötter et al., 2007], les auteurs proposent de reconfigurer le contrôle des roues multidirectionnelles du robot pour qu'il puisse continuer à fonctionner malgré l'occurrence de certains défauts sur l'une de celles-ci. On notera ici une dégradation de la performance du robot.

La réinitialisation d'un composant fautif peut permettre après réactivation d'obtenir un nou-

vel état stable. [Weber and Wotawa, 2008] propose une stratégie de réinitialisation des modules fautifs de l'architecture présentée pour répondre à la propagation d'une faute au travers de plusieurs modules. Ils utilisent un arbre de dépendance des fautes pour réinitialiser le système dans les meilleures conditions selon l'ordre chronologique le plus adapté.

Si cette présentation démontre qu'il existe une large palette de mécanismes de recouvrement de fautes, il s'avère que pratiquement les architectures robotiques font plus particulièrement appel aux principes d'isolement et de reconfiguration qui sont évoqués (tant en présence de fautes internes qu'externes).

3.5.2.2 Mécanismes de recouvrement au sein des architectures de contrôle

Dans un système robotisé autonome, un dysfonctionnement ou situation adverse peut avoir plusieurs origines. D'une part elle peut provenir d'un composant interne fautif qui ne respecte pas ses contraintes temporelles ou qui est incapable d'assurer son service par exemple s'il n'a pas les ressources nécessaires. D'autre part la défaillance peut être induite par une incertitude de l'environnement ou par une faute d'origine inconnue. A partir de ces situations, et sans être pour autant totalement disjointes, deux classes de réaction peuvent être envisagées. Dans la première, le système fait face à des fautes internes ou externes bien identifiées, et peut engager des actions locales et parfaitement ciblées de recouvrement au niveau fonctionnel de l'architecture. Dans la seconde, les situations adverses rencontrées, à l'origine indéterminée ou sans alternative locale, conduisent à reconsidérer plus globalement la réalisation de la mission du robot.

Pour répondre à l'occurrence de situations adverses les mécanismes proposés au sein des architectures de contrôle utilisent principalement le contrôle d'exécution et la (re)planification.

3.5.2.2.a Recouvrement par contrôle d'exécution

Le contrôle d'exécution choisit et supervise la réalisation de chacune des actions nécessaires à la mise en œuvre du plan spécifié. Il assure la détection de défaillances, le diagnostic de la faute correspondante et, s'il le peut, engage le recouvrement au niveau fonctionnel. Les différentes actions possibles sont :

Le traitement particulier : il répond à une situation adverse connue et diagnostiquée en reconfigurant le système physique ou l'algorithme impliqué.

La reprise d'action : il relance l'action qui vient d'échouer en supposant que le contexte d'exécution lui soit maintenant plus favorable.

La reprise d'action avec redondance fonctionnelle : il cherche des solutions alternatives à l'action qui vient d'échouer. Cette approche qui semble préférable à la démarche précédente nécessite de disposer de ressources fonctionnelles redondantes.

La modalité : c'est une extension du principe de reprise d'action qui représente un ensemble de tâches pouvant toutes effectuer une même action. Pour un contexte donné le contrôleur d'exécution choisit la modalité la plus performante.

Ce type d'approche est employé dans [Ranganathan and Koenig., 2003] qui présente un basculement entre plusieurs modes de navigation pouvant être assimilé au principe de modalité. Lorsque le robot est bloqué dans un minimum local (c'est-à-dire, lorsqu'il n'est pas capable de progresser vers l'objectif pendant un intervalle de temps donné) plusieurs modes alternatifs peuvent être utilisés : navigation réactive, navigation réactive avec un point de passage ou navigation avec le suivi de courts chemins locaux replanifiés fréquemment. Dans [Morisset et al., 2004], le principe de modalités est appliqué en proposant des modes faisant appel à différentes techniques de localisation et de navigation (par exemple odométrie, stéréo-odométrie et GPS pour la localisation, planification de trajectoire ou navigation réactive pour la navigation). Un mécanisme d'apprentissage basé sur l'évaluation de l'état du robot et de son environnement permet de choisir la modalité la plus performante.

Les mécanismes de recouvrement par contrôle d'exécution sont toujours présents, sous une forme ou sous une autre, et de façon explicite ou non, au sein d'une architecture de contrôle. Outre le principe de modalité, ceux de traitement particulier et de reprise d'action avec redondance fonctionnelle sont les plus largement déployés au sein des mécanismes de contrôle d'exécution mis en place dans les architectures robotiques. Il est possible de mettre en évidence différentes logiques d'implémentation.

Utilisation d'un module spécifique de contrôle d'exécution : dans l'architecture LAAS le contrôle d'exécution a été centralisé au sein d'un composant dédié R2C (voir figure 3.4) qui permet de tenir à jour l'état du système à l'aide des requêtes et bilan transitant entre les niveaux décisionnel et fonctionnel [Py and Ingrand, 2004]. Il utilise l'état courant pour valider les nouvelles requêtes émises et déterminer les réactions lors de détection de dysfonctionnements.

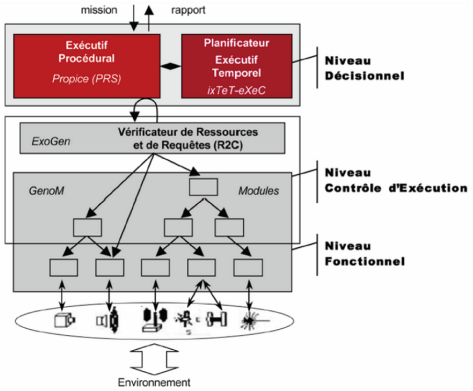
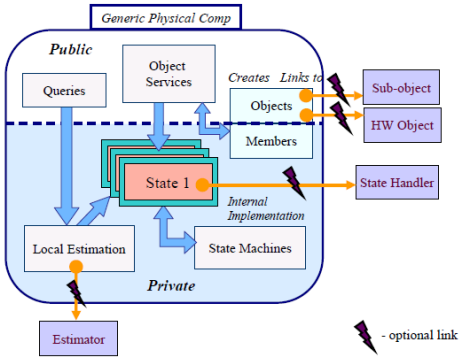


Figure 3.4 – Architecture LAAS [Alami et al., 1998]

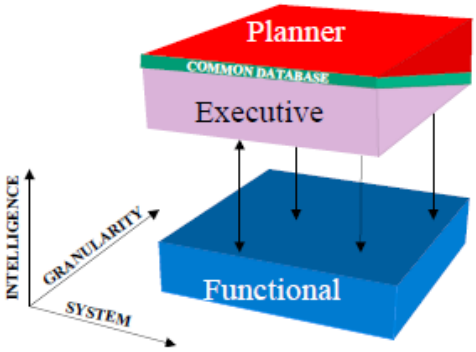
Le composant MIR (*Mode identification and reconfiguration*) du système Livingstone [Williams and Nayak., 1996], implémenté dans l'architecture RAX [Muscettola, 1998], propose à la fois des capacités de détection et diagnostic des fautes et des solutions de

reconfiguration et de reprise d'action lorsque cela est possible.

Déploiement interne aux modules fonctionnels : Dans CLARATY [Volpe et al., 2000], les différents modules fonctionnels (figure 3.5a) disposent en leur sein des mécanismes de recouvrement locaux, permettant de réagir à la détection d'un dysfonctionnement connu.



(a) Composant générique de l'architecture



(b) Structure de l'architecture

Figure 3.5 – Architecture CLARATY [Volpe et al., 2001]

Dans l'architecture IDEA [Muscettola et al., 2002], un planificateur réactif permet à chaque agent de l'architecture de gérer l'occurrence d'un dysfonctionnement en sélectionnant en fonction des contraintes d'exécution définies un script spécifique pré-établi au sein d'une base de données locale. En cas d'échec, l'agent est positionné en état d'attente pendant que le planificateur cherche une solution au problème rencontré.

Gestion du recouvrement au niveau décisionnel : L'architecture CLARATY utilise le langage TDL (*Task description langage*) au sein de la partie exécutive du niveau décisionnel (figure 3.5b) pour effectuer de la reprise d'action [Estlin et al., 2001].

Dans l'architecture ORCCAD (figure 3.6) plusieurs solutions de reprise d'action par redondance et de traitement particulier sont proposées en fonction de la nature du dysfonctionnement détecté. La réponse peut être locale en reparamétrant la tâche robotique courante. Elle peut être plus large en changeant de tâche robotique. En cas de problème fatal le positionnement dans un état sûr est cherché. La détection de défaillance est assurée au niveau fonctionnel des tâches robotiques par des observateurs spécifiques. La gestion du recouvrement est assuré par les ROBOTIC PROCEDURE [Borrelly et al., 1998] que l'on peut assimiler au niveau décisionnel de l'architecture.

Le contrôle d'exécution permet une réponse locale et présentant une bonne réactivité à la détection d'un dysfonctionnement. Cependant plusieurs limitations peuvent être soulignées.

– D'une part pour répondre à un dysfonctionnement il est nécessaire au préalable de l'avoir identifié. Bien que cette démarche soit naturelle elle semble relever, au sein de toutes les architectures, d'une approche empirique s'appuyant sur l'expérience du développeur et/ou de l'expérimentateur.

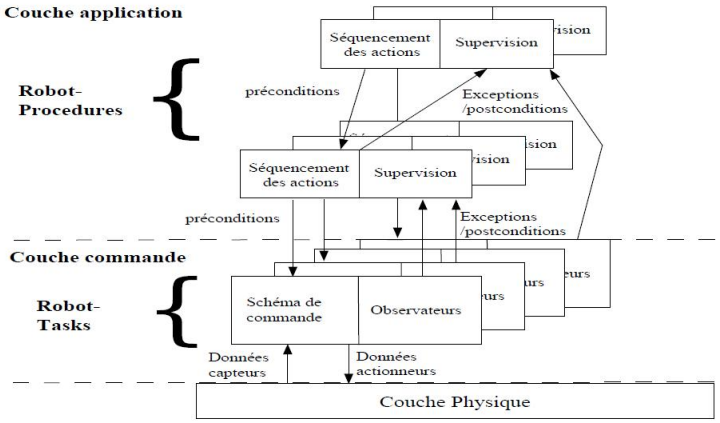


Figure 3.6 – Structure de l'architecture ORCCAD [Borrelly et al., 1998]

– D'autre part sans réelle étape d'identification la tolérance aux fautes embarquée peut souffrir à la fois « d'incomplétude », et d'un ensemble de défauts inutiles à détecter et à traiter. Une démarche structurée pourrait permettre de concentrer la détection des défauts sur les plus critiques.
– Enfin les solutions de recouvrement proposées supposent la plupart du temps d'être capable de mettre en œuvre un processus de diagnostic à partir des symptômes de défaillances. Ces mécanismes sont pratiquement toujours passés sous silence dans les publications alors qu'ils constituent une étape essentielle du processus de recouvrement. Cela conduit à penser que, la plupart du temps, un dysfonctionnement observé est assimilé à une seule faute. Une étape de diagnostic plus sophistiquée permettrait de détecter des situations plus complexes lorsque par exemple plusieurs défaillances sont générés par une seule faute.

Le contrôle d'exécution doit être le fruit d'une démarche réfléchie et rester ouvert à d'autres mécanismes de recouvrement en présence de dysfonctionnements ou de situations imprévues. En effet, si une réponse locale ne peut être apportée, le plan d'exécution courant conduit à une situation d'échec. Une solution plus globale doit alors être cherchée, lorsque cela est possible, au niveau du planificateur de mission pour identifier une alternative de recouvrement.

3.5.2.2.b Recouvrement par planification

La planification permet de définir la succession des tâches que doit réaliser le robot pour atteindre les buts fixés pour sa mission. Lorsque certains composants du niveau fonctionnel sont déclarés en échec, une modification du plan d'exécution peut permettre de réaliser la mission sans pour autant les utiliser.

Pour ce contexte de recouvrement, les moyens suivants peuvent être envisagés pour répondre aux situations adverses :

la prise en compte de situations adverses spécifiées : Au sein du planificateur elle permet

d'éviter dès la planification que le robot se dirige inévitablement vers une situation adverse répertoriée. Cette méthode située en amont du processus de recouvrement même, est mise en œuvre dans l'architecture CIRCA [Goldman et al., 1997] au niveau décisionnel dans le but d'éviter des situations dues aux incertitudes de l'environnement. Par exemple, la couverture radar ou la présence d'ennemis sont donc des propriétés prises en compte pour ne pas mettre le robot dans une situation difficile.

la réparation de plan : Elle a pour but, lors de la détection de l'échec d'un plan, de le modifier localement pour tenter de le poursuivre sans avoir à payer le cout temporel associé à une replanification complète. Cette approche est déployée sur le planificateur IXTET de l'architecture LAAS [Lemai and Ingrand, 2004].

Dans l'architecture CLARATY [Estlin et al., 2001] le niveau décisionnel est mis en œuvre par le système CLEAR (*Closed-Loop Execution and Recovery*) (figure 3.5b). Le système de planification itérative par réparation successive du plan est géré par CASPER (*Continuous Activity Scheduling, Planning, Execution and Re-planning*). Lorsque la partie exécutive (TDL) n'a pas réussi à répondre à un dysfonctionnement CASPER replanifie par réparations successives pour répondre aux situations conflictuelles détectées. CASPER semble aussi réévaluer un plan lorsque de nouvelles opportunités se présentent, par exemple si des ressources sont à nouveau utilisables.

la replanification : Elle permet la poursuite de la mission en identifiant le plan le plus adapté pour achever la mission en cours. Elle peut intervenir sur échec du plan courant ou périodiquement pour éviter l'apparition de situations adverses. Dans l'architecture LAAS, la replanification est demandée par le composant R2C et exécutée par le IXTET. Ce dernier a été modifié dans [Lussier, 2007] et accompagné d'un composant FTPLAN permettant de prendre en compte l'occurrence de fautes au niveau du planificateur (chien de garde, erreur de plan), ou au niveau exécutif (vérification en ligne d'objectifs). Les actions de recouvrement induites font en fait appel à plusieurs planificateurs, s'appuyant sur des modèles et heuristiques différents, exécutés de façon concurrente ou successive.

Dans [Nana, 2007], après un échec, la nature interprété du langage PILOT permet à l'opérateur de replanifier en ligne la mission pour pouvoir l'achever ou de mettre le robot dans un état sûr.

Evidemment cette classe de mécanismes de recouvrement, s'il étend encore la capacité de tolérance aux fautes en proposant une autre alternative que le contrôle d'exécution, ne reste envisageable que pour les architectures de contrôle disposant intrinsèquement d'un planificateur. Cependant, comme le souligne [Verfaillie, 2006], ce mécanisme de recouvrement est entaché de quatre défauts majeurs :

- Le choix d'un horizon de planification approprié reste une question ouverte.
- Si les réactions aux échecs sont souvent prises en compte, le rétablissement de fonctionnalités est très rarement considéré.

– En présence d'un environnement très dynamique pouvant engendrer de nombreux échecs d'actions, les replanifications peuvent devenir très nombreuses et inutilement couteuses.
– Rien ne garanti que la replanification puisse être réalisée dans un temps compatible avec les exigences de réactivité du système robotisé.

Ainsi, on a vu que les méthodes de recouvrement autonomes sont nombreuses et permettent une prise en compte locale par contrôle d'exécution, ou globale par replanification des erreurs. Nous avons cependant vu qu'un certain nombres de limitations ne leur permettent pas de répondre à certaines situations. En l'absence de solutions alternatives, la plupart des architectures de contrôle proposent de positionner le système robotisé dans un état sûr et sécuritaire mettant fin à la mission en cours. Cependant, il nous semble particulièrement pertinent, dans ce cas de figure, d'ajuster le niveau d'autonomie du robot de façon à pouvoir engager une interaction Homme-Robot où les capacités cognitives ou d'actions à distance de l'opérateur permettront, si c'est possible, d'apporter ou de proposer une réponse à la situation d'échec. C'est pourquoi nous allons maintenant nous attarder sur les rares exemples proposant un ajustement du niveau d'autonomie comme solution de recouvrement.

3.5.2.3 Recouvrement par autonomie ajustable - Interaction Homme-Robot

L'ajustement de l'autonomie du robot peut permettre, lorsque ce dernier n'a plus les capacités décisionnelles ou fonctionnelles nécessaires à la réalisation de sa mission, à un opérateur distant d'apporter ou proposer une solution de recouvrement.

Le paragraphe 2.5.3 a présenté rapidement les concepts d'autonomie, de niveaux d'autonomie et les différents types d'interaction Homme-Robot envisageables. Si comme le souligne [Goodrich and Schultz, 2007] l'interaction est indispensable dans de nombreux domaines applicatifs comme les robots assistants pour la maison, l'industrie, les actions policières et militaires, l'enseignement ludique (*edutainment*) ou les robot spatiaux, peu de travaux s'inscrivent ouvertement dans une logique de tolérance aux fautes. Nous allons présenter ici les quelques cas applicatifs d'ajustement du niveau d'autonomie, relevant du recouvrement des fautes, que nous avons identifiés.

Dans le domaine de la recherche et du sauvetage en milieu urbain USAR (*Urban Search and Rescue*), les travaux de [Bruemmer et al., 2002] proposent de définir 4 niveaux d'autonomie dont la sélection est initiée par l'opérateur en fonction du contexte d'exécution de la mission : la téléopération, la téléopération avec évitement d'obstacle (*safe mode*), le contrôle partagé (*shared mode*) et le mode autonome (*autonomous mode*). Dans [Baker and Yanco, 2004] les auteurs adjoignent deux modes de fonctionnement, le mode évasion (*Escape*) pour sortir d'un couloir sans issue, et le mode poursuite (*pursuit*) pour traquer une cible, qui permettent de dégager le robot lorsqu'il se trouve dans des configurations d'évolution spécifiques. Certaines commutations de modes peuvent être proposées par le robot à l'opérateur lorsque des situations particulières sont détectées. Par ailleurs l'opérateur peut encore, s'il le souhaite, choisir

le mode de fonctionnement. Le mécanisme de recouvrement intègre donc ici une légère part d'adaptativité. Dans [Zimmel et al., 2004] les auteurs proposent d'utiliser l'architecture SFX-EH [Murphy and Hershberger, 1999] pour la recherche de victimes. Seulement trois types d'interaction Homme-Robot sont implémentées : la supervision (*monitor*) dans lequel le robot est autonome, la résolution de problème (*solving problem*) et la téléopération (*teleoperation*). Au sein de l'architecture, le gestionnaire d'erreur (*Error handler*) qui met en œuvre la tolérance aux fautes peut initier, sur occurrence de défauts capteurs, une interaction avec l'opérateur. L'interface Homme-machine bascule alors directement du mode de supervision au mode de résolution de problème. Les expérimentations menées révèlent que dans leur contexte, le recouvrement autonome est plus rapide qu'en sollicitant l'intervention humaine en raison du temps nécessaire à la prise en compte du contexte d'évolution du robot. Cependant le contexte expérimental retenu ne permet pas de statuer sur la pertinence de l'interaction Homme-Robot.

On trouve aussi quelques exemples de recouvrement par ajustement de l'interaction Homme-Robot pour des applications avioniques. [Taylor, 2001] présente au sein d'un cockpit cognitif l'environnement PACT (*Pilot Authorization Control of Tasks*) où ils utilisent dans un premier temps trois niveaux d'interaction homme-robot : commandé (*commanded*), automatique (*automatic*), et assisté (*assisted*). Ce dernier est lui-même scindé en quatre sous-niveaux pour permettre de décomposer le partage d'autonomie selon les besoins de la mission en allant du support direct (*direct support*) de l'homme au robot autonome à l'inverse ou l'homme est aidée par le robot lors de la téléopération lorsqu'il en fait l'appel (*at call*).

A l'inverse dans [Mercier et al., 2009] les auteurs proposent une autonomie ajustable sans niveau prédéfini d'autonomie. Leur approche est basée sur une gestion des ressources, modélisées à l'aide de réseau de Petri, et potentiellement gérées de façon autonome ou téléopéré. Dans leurs expériences, la perte de la localisation GPS est compensée par l'allocation de la tâche de navigation à l'opérateur. Lors du rétablissement du système GPS celle-ci est à nouveau gérée de façon autonome.

Ces rares exemples de recouvrement mettant en œuvre un ajustement du niveau d'autonomie du robot démontrent que ce moyen d'amélioration de la tolérance aux fautes est peu envisagé. Les pannes prises en compte relèvent souvent de dysfonctionnements liés à l'environnement du robot plutôt que de fautes matérielles ou logicielles. Ces travaux mettent en évidence le développement d'interfaces adaptées au problème de recouvrement rencontré. Cela suppose donc la mise à disposition de l'opérateur de l'ensemble des informations contextuellement pertinentes, tant internes que sur l'environnement du robot, qui lui permettront d'engager la réaction la plus adéquate. Réciproquement, le recouvrement vers le mode autonome nécessite que l'opérateur puisse fournir au robot les données permettant à ce dernier de reconstruire un état stable et cohérent. Cette dernière constatation est loin d'être évidente à mettre en œuvre.

Les travaux trouvés semblent attester que le recours à l'Homme par interaction distante, pour pallier aux insuffisances des robots, est souhaitable notamment dans des domaines d'applications critiques (USAR - avionique).

3.6 Conclusion

Après avoir présenté dans le premier chapitre les solutions proposées par la sûreté de fonctionnement pour mettre en œuvre des systèmes sûrs, nous nous sommes attachés dans ce chapitre à présenter les architectures de contrôle robotiques et les atouts de celles-ci pour mettre en œuvre la sûreté de fonctionnement. Les architectures de contrôle étant le centre névralgique des robots autonomes, il semble primordial de mettre en œuvre la tolérance aux fautes en leur sein.

Les structures puristes, réactives ou délibératives, sont très contraignantes et ne mettent pas en œuvre de façon complémentaire les qualités de réactivité et de délibération et donc des solutions hybrides sont adoptées. L'utilisation du concept de modularité et d'outils de conception sophistiqués permettent de produire des cadres de développement sûr pour prévenir l'apparition des fautes dans les architectures de contrôle.

L'élimination des fautes basée sur la vérification formelle est également un accent important mis en avant dans certaines architectures de contrôle pour promouvoir un développement sûr de fonctionnement. Le test et la simulation apportent aussi à l'élimination des fautes mais ne permettent pas d'avoir une approche exhaustive du comportement du système et de son environnement. Seul le test intensif réalisé dans le cadre du projet DS1 de la NASA a permis d'avoir cette approche globale de l'élimination des fautes dans le système sans pour autant prétendre à l'exhaustivité.

Malgré tous les efforts de développement mis en place la réalisation d'un robot sans fautes est illusoire. La prévision des fautes a pour but l'étude du robot et de son architecture pour en définir les points critiques et soit apporter des solutions correctives, soit des solutions permettant de poursuivre le fonctionnement malgré l'occurrence de fautes. Plusieurs solutions ont été évoquées mais trop peu sont utilisées dans le cadre des architectures de contrôle. Cependant l'utilisation de démarches structurantes telle l'AMDEC provenant de la productique permet d'améliorer la prévision des fautes mais leurs applications restent quasi inexistantes dans le domaine de la robotique.

A l'inverse de la prévision des fautes, la robustesse et la tolérance aux fautes sont souvent utilisées dans les architectures de contrôle. Par contre leur utilisation est éparpillée dans les sous-systèmes des architectures et ne permet pas de mettre en œuvre une approche globale complète de tolérance aux fautes. L'utilisation d'observateurs spécifiques pour la détection se révèle intéressante car elle permet de structurer l'intégration de la tolérance dans l'architecture. Cependant, la seconde étape du processus de tolérance aux fautes, le diagnostic, est majoritairement passé sous silence. Des solutions très simples semblent être réalisées conjointement à la détection. Le recouvrement est quant à lui souvent mis en avant par l'utilisation à la fois du contrôle d'exécution et de stratégies de planification. Il fait partie majoritairement du processus décisionnel des architectures de contrôle. Un troisième type de solutions pourrait être le recouvrement par l'autonomie ajustable. L'interaction Homme-Robot est un thème porteur de recherche visant à tirer profit de la relation entre les capacités complémentaire des Hommes et

des Robots pour améliorer la réalisation de tâches. Cependant, à notre connaissance très peu d'exemples d'interaction sont utilisés dans le cadre de la sûreté de fonctionnement.

Pour conclure on peut remarquer qu'il manque une démarche globale, complète et structurée de mise en œuvre de la tolérance aux fautes dans les architectures de contrôle robotiques partant de l'identification des fautes pertinentes jusqu'à la réalisation des processus de détection et diagnostic correspondants et des stratégies de recouvrement adaptées. Nous pensons que seule une prise en compte de toutes les étapes de la sûreté de fonctionnement dans une méthodologie globale pourra permettre le développement de systèmes robotiques fiables dans des environnements complexes et dynamiques.

Conclusion de l'état de l'art

Dans cette première partie, nous avons, dans un premier temps, fait un tour d'horizon des nombreuses méthodes et mécanismes proposés et mis en œuvre pour concevoir des systèmes de contrôle tolérants aux fautes. Puis dans un second temps nous nous sommes focalisés sur les systèmes robotiques où, nous l'avons constaté, les principes relevant de la sûreté de fonctionnement sont implémentés le plus souvent de façon assez empirique, locale et parcellaire. L'analyse croisée de ces études démontre à l'évidence qu'il n'existe pas de démarche globale, structurée et générique permettant la conception de systèmes robotiques tolérants aux fautes et évoluant dans des environnements dynamiques. L'élaboration d'une telle démarche pour des raisons de rapidité, flexibilité et efficacité doit s'appuyer sur l'architecture logicielle au sein de laquelle est mis en œuvre le contrôleur du robot. Elle doit satisfaire aux objectifs suivants :

- Guider l'utilisateur dans l'identification de l'ensemble des fautes pertinentes pouvant affecter le robot et de leur sévérité, sans occulter celles qui pourraient se manifester au sein même de l'architecture de contrôle et du contrôleur déployé.
- Proposer des mécanismes permettant d'observer au sein de l'architecture support l'occurrence de défaillances.
- Mettre en œuvre une approche de diagnostic générique permettant d'identifier, lorsque cela est possible, les fautes à l'origine des défaillances affectant le robot.
- Proposer une stratégie de recouvrement complète et adaptative faisant appel, en fonction du contexte, à des mécanismes permettant au robot, autant que faire ce peut, de poursuivre la mission.
- Assurer un déploiement en temps réel des mécanismes de contrôle et de tolérance aux fautes de façon à préserver la stabilité du système et la réactivité vis-à-vis de l'environnement dynamique du robot.
- Assurer la réussite de la mission, même d'une façon non optimale, ou en faisant appel à une intervention Humaine pour pallier aux limites décisionnelles et fonctionnelles de l'architecture embarquée.

La suite de ce document propose, met en œuvre et expérimente une méthodologie de conception d'architecture de contrôle tolérante aux fautes visant à remplir les différents objectifs qui viennent d'être énoncés.

Deuxième partie

Proposition d’une méthodologie pour la conception d’architecture de contrôle tolérante aux fautes

Chapitre 4

Méthodologie pour une architecture de contrôle tolérante aux fautes

4.1 Proposition

La partie précédente nous a permis de voir qu'il n'existe pas de démarche générique, globale et structurée pour la gestion de la sûreté de fonctionnement dans les systèmes robotiques. Nous allons donc proposer une méthodologie permettant d'intégrer d'une façon systématique des mécanismes pour la tolérance aux fautes au sein d'une architecture de contrôle.

Cette méthodologie est décomposée en plusieurs étapes :

- Adoption d'une démarche permettant d'identifier les fautes pertinentes devant être traitées dans un contexte expérimental donné. Cette étape permettra de classifier ces erreurs en fonction de leur type et de leur sévérité. L'utilisation d'une démarche systématique d'identification à ce stade du processus de conception permettra de ne pas se limiter aux fautes les plus courantes (capteurs, actionneurs, temporelles,..) mais de lister les fautes d'une façon aussi exhaustive que possible. Cette étape relève de la prévision des fautes dans le cadre de la sûreté de fonctionnement.
- Développement de mécanismes d'observation spécifiques, implémentés au sein de l'architecture, permettant de détecter l'occurrence des défaillances en faisant appel à des techniques existantes. Ces mécanismes d'observation sont directement reliés aux différentes fautes précédemment identifiées.
- Utilisation de mécanismes de diagnostic efficaces permettant d'isoler et de localiser les fautes à l'origine des défaillances observées.
- Développement de mécanismes décisionnels permettant de réagir à la détection des défaillances. Afin de garantir le fonctionnement en présence de fautes, différents types de mécanismes de recouvrement seront mis en œuvre pour apporter une réponse adaptée au contexte de la mission et à la sévérité des fautes détectées.

Cette méthodologie sera déployée au sein d'une architecture de contrôle modulaire ce qui per-

met de s'inscrire dans une logique de prévention de fautes dans le cadre de la sûreté de fonctionnement.

Avant de décrire en détails les différentes étapes de la méthodologie proposée, la section suivante permet de préciser les grands principes de fonctionnement d'une telle architecture de contrôle.

4.2 Architecture de contrôle : Principes généraux

Conformément à l'analyse des architectures robotiques menée dans la partie précédente nous supposerons dans cette partie que le contrôleur est implanté au sein d'une architecture hybride modulaire composée de deux couches, pour préserver un comportement à la fois déclaratif et réactif.

La couche décisionnelle reçoit directement ses ordres de l'opérateur sous la forme d'une mission à accomplir. Son rôle est d'une part de décomposer la mission en un ensemble d'objectifs qui devront être exécutés concurremment ou séquentiellement et, d'autre part, de superviser l'exécution de la mission en fonction de son évolution.

$$Mission = \{Objectif\} \tag{4.1}$$

La couche décisionnelle a aussi en charge l'exécution des objectifs sélectionnés à un instant donné. Un objectif est alors décomposé en un ensemble de sous-objectifs pouvant être mis en œuvre concurremment ou séquentiellement.

$$Objectif = \{Sous\ Objectif\} \tag{4.2}$$

Un sous-objectif représente une tâche robotique. Il correspond à une suite de traitements informatiques réalisés séquentiellement selon un motif fixe répété cycliquement et devant satisfaire à des contraintes temps réel liées au contexte expérimental. Chaque traitement est implanté dans un module spécifique appartenant à la couche exécutive.

$$Sous\ Objectif = Tache\ robotique = \{Module\} \tag{4.3}$$

Ainsi, la couche fonctionnelle contient l'ensemble des modules pouvant être employés pour réaliser les fonctionnalités robotiques nécessaires à la mise en œuvre des sous-objectifs pouvant composer la mission.

En général la structure d'une tâche robotique peut être schématiquement représentée par la figure 4.1. Le premier traitement correspond à la lecture des informations capteurs. Les données obtenues sont alors traitées puis utilisées pour le calcul de la commande. Enfin les ordres actionneurs sont émis vers le robot. Le temps d'exécution de l'ensemble de ces modules doit rester inférieur au cycle d'exécution (`Tcycle` sur le schéma).

Des mécanismes de communication adaptés permettent d'échanger les flots de données et de contrôle entre les modules et la couche décisionnelle. La modularité affichée de l'architecture

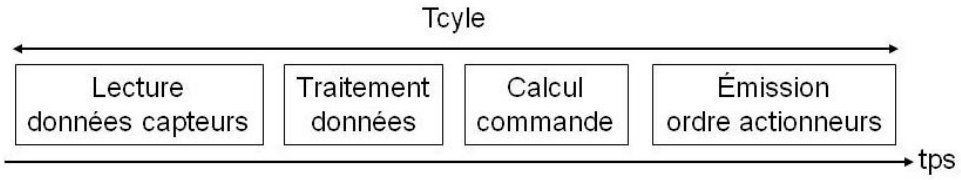


Figure 4.1 – Structure générique d'une tâche robotique

permet d'assurer les principes de réutilisabilité, maintenabilité, évolutivité, flexibilité souhaitables pour s'inscrire dans une logique de prévention de faute nécessaire pour construire des applications logicielles d'envergure.

Après avoir énoncé les principes soutendants l'architecture de contrôle qui supportera le déploiement de notre approche, nous allons maintenant détailler la méthodologie que nous proposons pour concevoir des robots mobiles autonomes tolérants aux fautes.

4.3 Méthode de prévision des fautes : notre vision de l'AMDEC

Le déploiement de mécanismes de tolérance aux fautes nécessite la mise en œuvre préalable d'une méthodologie de prévision des fautes. Parmi celles présentées dans la section 2.3 nous avons opté pour la démarche AMDEC. En effet même si elle ne permet de considérer qu'une seule faute à la fois et nécessite souvent un temps d'analyse non négligeable, elle reste une méthode performante et flexible adaptée à l'intégration de la sûreté de fonctionnement dans les architectures robotiques.

Nous allons maintenant détailler comment nous proposons d'adapter chacune des étapes de l'AMDEC à notre problématique en nous appuyant sur le cas d'un robot mobile contrôlé par une architecture modulaire. Commençons donc par définir les limites fonctionnelles de notre système.

4.3.1 Étape 1 : Détermination des limites du système

La première étape de la démarche AMDEC est d'identifier les limites du système que l'on souhaite analyser. Dans le cas d'un robot mobile pouvant faire appel à différents niveaux d'autonomie, il est clair que les limites du système vont évoluer en fonction du type d'interaction Homme-Robot. Ces limites pouvant inclure ou non, et de façon ponctuelle ou permanente, les moyens de communication et d'interaction nécessaires à la relation Homme-Robot. De plus, puisque nous nous plaçons dans une logique architecturale, il semble essentiel de déployer l'analyse des modes de défaillances et de leur criticité en fonction des modules impliqués et donc pour chacune des tâches robotiques envisagées.

Pour les raisons que nous venons d'évoquer, l'analyse AMDEC commencera donc par la décomposition de la mission en un ensemble d'objectifs et de sous-objectifs. Chacun de ces derniers

pouvant être déployé au niveau architectural de différentes façons, en fonction des modules opérationnels (non défaillants), et du niveau d'interaction Homme-Robot souhaitable pour pouvoir poursuivre avec succès la mission entamée. La décomposition fonctionnelle AMDEC sera donc conduite au niveau de chacune des tâches robotiques associées aux différents sous-objectifs du robot.

4.3.2 Étape 2 : Décomposition fonctionnelle du système

Un robot mobile contrôlé par une architecture logicielle peut être décomposé en deux parties distinctes. D'une part les éléments mécaniques fixes (carcasse, base, ...) et mobiles (roues, bras, ..), ainsi que de ses composants électromécaniques tels que les actionneurs (moteur, vérin, ..) et les capteurs (encodeur, caméra, sonars, ..). D'autre part, le contrôleur composé par le système informatique supportant l'architecture de contrôle logicielle.

Cette première décomposition ne permet pas d'analyser fonctionnellement le robot dans notre logique architecturale. Nous proposons donc d'utiliser une technique d'analyse fonctionnelle.

4.3.2.1 Analyse fonctionnelle SADT

La méthodologie systémique SADT (*Structured Analysis and Design Technique*) [Ross, 1977] permet une analyse graphique, descendante, hiérarchique et détaillée des différentes fonctions du système.

Par exemple, la tâche robotique *se déplacer* associée au niveau d'autonomie **téléopération** (figure 4.2 peut être décomposée à l'aide de la méthode SADT.

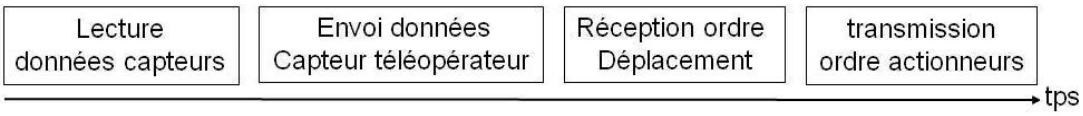


Figure 4.2 – Structure de la tâche robotique **se déplacer** en **téléopération**

La figure 4.3 présente la décomposition de la fonction **Se déplacer en téléopération** d'un robot mobile. Dans ce formalisme, celle-ci est décrite dans le haut de la figure avec ses entrées et sorties et le support sur lequel elle agit, le *robot mobile*. Les données de contrôle ne sont ici pas représentées mais proviennent de la couche décisionnelle de l'architecture. Les entrées du système sont les données capteurs de l'interface de téléopération (**position joystick**) et de la caméra du robot (**données camera**). Les sorties du système sont les vitesses fournies aux roues qui détermine le *déplacement du robot* et les *images envoyées vers le téléopérateur*. Le bas de la figure met en évidence les flux de données reliant les sous-fonctions : **Percevoir**, **Communiquer**, **Traduire les commandes** et **Rouler** mis en œuvre.

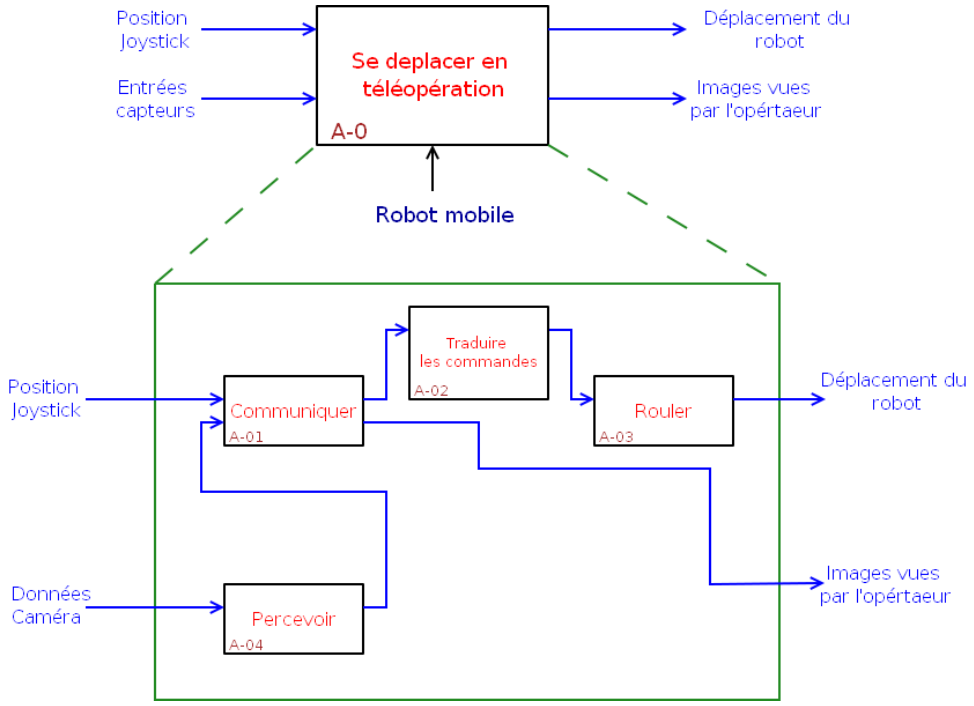


Figure 4.3 – Décomposition SADT de la fonction **Téléopérer**

4.3.2.2 Analyse fonctionnelle : lien avec le point de vue architectural

4.3.2.2.a De la fonction aux modules

L'utilisation de la méthodologie SADT permet d'avoir une vue de l'ensemble des fonctions du système, mais il est possible de la projeter plus précisément sur la structure modulaire de l'architecture.

Il existe un niveau de décomposition pour lequel chacune des fonctions identifiées correspond à l'exécution d'un module dans l'architecture de contrôle. Par exemple, la fonction **Percevoir** de l'exemple précédent correspond à l'exécution du module chargé de la lecture du capteur webcam (figure 4.2).

4.3.2.2.b Des modules aux modes de fonctionnement

Un **mode de fonctionnement** d'un robot peut être défini comme l'exécution ordonnée d'un ensemble de modules permettant de réaliser un sous-objectif (tâche robotique) d'une mission. Il est possible de distinguer le **mode de fonctionnement nominal** où l'ensemble des capacités du robot sont utilisables et les **modes de fonctionnement dégradés** où le robot ne dispose que d'un sous-ensemble de ses capacités.

Il existe pour un type d'interaction Homme-Robot et pour une même fonction réalisée, un « classement » des modes de fonctionnement basé sur une notion d'efficacité, dont les critères seront précisés dans la suite du document. La décomposition fonctionnelle de notre robot nous

guide pour identifier l'ensemble des fonctionnalités, et donc des modules, permettant la réalisation de chaque sous-objectif de la mission.

4.3.3 Étape 3 : Identification des modes de défaillances et de leur sévérité

La seconde étape de la méthodologie AMDEC consiste à identifier les modes de défaillance de l'ensemble de nos modes de fonctionnement, les fautes causant ces défaillances ainsi que leurs sévérités.

4.3.3.1 Modes de défaillance

Dans notre approche, nous considérons que les modes de défaillances définis traduisent la disponibilité et la qualité du service délivré par la fonctionnalité étudiée. Les modes de défaillances sont donc définis selon leur type qui représente l'impact de la faute sur le service fourni par le module, alors que la classe du mode de défaillance est liée à sa durée d'activation.

Nous définissons deux types de modes de défaillances :

- **Complète** : le service à délivrer ne peut plus l'être.
- **Partielle** : la qualité du service délivré est dégradée. Ces défaillances permettent de prendre en compte la déviation lente de la qualité de service, comme par exemple la surcharge réseau d'une communication sans fil.

La durée d'activation du mode de défaillance permet de discerner ces deux classes :

- **Permanente** : l'altération du service est définitive.
- **Transitoire** : le service est altéré pendant une période donnée.

La classe intermittente est souvent rencontrée dans la littérature, mais nous ne la différencierons pas car elle ne correspond qu'à la réactivation répétitive de la classe transitoire.

4.3.3.2 Identification des causes des défaillances

Après avoir précisé les modules intervenants dans les différents modes de fonctionnement, leurs modes de défaillances potentielles, l'étape suivante de la méthodologie AMDEC est d'identifier les causes des défaillances, c'est-à-dire les fautes.

L'objectif ici est d'identifier, pour chaque module de l'architecture, les fautes pouvant potentiellement engendrer les défaillances observées. Pour y parvenir nous allons utiliser une méthodologie d'analyse cause-effet (diagramme d'Ishikawa) qui s'appuiera sur une taxonomie des fautes potentielles permettant de diriger notre analyse.

4.3.3.2.a Taxonomie de fautes potentielles

En s'inspirant des taxonomies de fautes utilisées couramment en sûreté de fonctionnement et particulièrement en robotique (cf. section 1.1.1), celle que nous proposons retient les classes de fautes suivantes :

Capteur : fautes relevant de l'ensemble des capteurs et des dispositifs d'interface entre ces derniers et l'architecture de contrôle.

Actionneur : fautes relevant de l'ensemble des actionneurs et des dispositifs d'interface entre ces derniers et l'architecture de contrôle.

Énergie : fautes relevant de l'ensemble des éléments associés à la gestion de l'énergie du robot.

Environnement : ensemble des fautes liées à l'interaction entre le robot et son environnement.

Conception : ensemble des fautes liées à la conception, à l'implémentation et à l'exécution des algorithmes de contrôle et de gestion des composants matériels.

Architecture de contrôle : ensemble des fautes liées au non-respect des contraintes d'exécution d'un module.

Cette taxonomie est utilisée pour nous guider dans l'analyse cause-effet des défaillances.

4.3.3.2.b Diagrammes d'Ishikawa

Pour identifier les fautes potentielles provoquant les défaillances des modules nous proposons d'utiliser les diagrammes d'Ishikawa [Ishikawa, 1985], structurés sous forme d'arêtes de poisson, qui permettent d'identifier récursivement les causes possibles d'un effet constaté. Dans notre cas, chacune des arêtes principales du diagramme correspondra à une classe de la taxonomie de fautes que nous venons de présenter. De plus, nous proposons de rajouter à ces diagrammes une information relative aux modes de défaillances.

La figure 4.4 illustre la mise en œuvre de ces techniques sur un exemple de module de capteur odométrique. Les défaillances de celui-ci peuvent avoir pour cause les différentes fautes attachées aux arêtes du diagramme. La relation de certaines de ces fautes avec les modes de défaillances est traduite par un code de couleur. On retrouve en vert les causes de défaillance partielles, en bleu les causes de défaillances transitoires et en noir les causes de défaillances complètes.

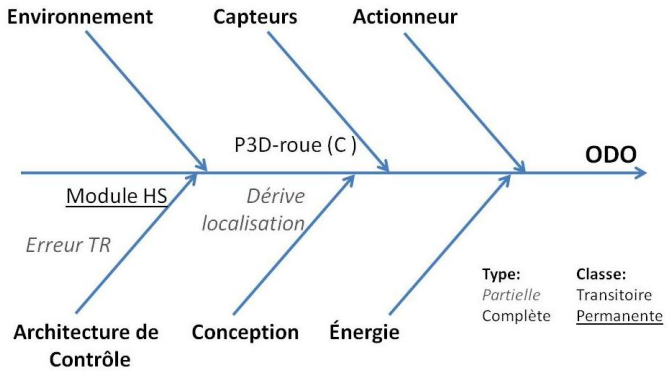


Figure 4.4 – Exemple de diagramme Ishikawa - Module Odomètre

4.3.3.3 Sévérité des défaillances

Le déploiement de mécanismes de tolérance aux fautes implique une évaluation de la sévérité des défaillances rencontrées pour permettre de choisir de façon pertinente la solution de recouvrement la plus adaptée.

Le degré de sévérité est défini en rapport à la réussite de la mission, ainsi que la sécurité du déplacement du robot vis à vis de lui-même et de son environnement. La sévérité des modes de défaillances prend donc en compte les capacités décisionnelles et fonctionnelles du robot à un instant donné pour déterminer la sévérité de la perte d'une fonctionnalité.

En respect avec l'état de l'art effectué dans la section 3.4.3, les niveaux de sévérité définis sont fonction des capacités de recouvrement du robot et au nombre de quatre :

- **Fatale** : La défaillance implique un échec de la mission, par exemple si une ressource indispensable est défaillante.
- **Grave** : La défaillance implique que le robot ne peut poursuivre sa mission en l'état. Les capacités du robot ne lui permettent pas de répondre de façon autonome à la situation.
- **Moyenne** : La défaillance perturbe le mode de fonctionnement courant qui ne peut être poursuivi. Par contre les capacités du robot lui permettent de répondre de façon autonome à la situation en utilisant un mode de fonctionnement dégradé.
- **Faible** : La défaillance ne perturbe que localement un module et il existe une solution locale, par exemple la re-configuration de l'algorithme du module défaillant.

En outre, certaines défaillances peuvent aussi être ignorées de par le retour d'expérience des personnes mettant en œuvre la démarche AMDEC.

Il faut remarquer qu'une même défaillance d'un module correspondra à des degrés de sévérité différents en fonction du mode de fonctionnement courant et des fonctionnalités déjà défaillantes du robot. Par exemple, la perte de capteurs ultrasonique est peu sévère si une solution d'évitement d'obstacle par laser ou par caméra est disponible. Dans le cas contraire, la sévérité sera plus élevée.

L'analyse de la sévérité des défaillances nécessite donc une vue globale du robot et des différents modes de fonctionnement. Elle est donc réalisée lors du regroupement de l'ensemble des informations collectées au sein de chaque tableau récapitulatif *Amdec* de l'ensemble des modes de fonctionnement accessibles pour la mission considérée.

4.3.4 Résultat : Tableau récapitulatif de l'AMDEC

La méthodologie AMDEC se termine usuellement par un tableau synthétisant les résultats de l'analyse développée [Bouti and Ait Kadi, 1994, Chevance, 2010]. Il permet d'identifier pour chacune des fonctions étudiées, ses modes de défaillances, les causes possibles ainsi que les effets et conséquences potentiels sur la fonction ou le système (table 4.1). Il évalue aussi la sévérité de la défaillance, et propose à la fois des moyens de détection envisageables pour la mettre en évidence et des actions potentielles de recouvrement.

Fonction (Composant)	Mode de défaillance	Causes	Effets Conséquences	Moyen de détection	Sévérité	Actions

Table 4.1 – Structure classique d'un tableau de synthèse AMDEC

Au gré des applications ce tableau peut être adapté aux besoins des utilisateurs. Par exemple dans [Nikam and Hall, 1999] qui réalise un diagnostic de fautes sur un robot mobile, seules les caractéristiques relevant du mode de défaillance, des potentielles causes et effets et des actions remédiatrices sont préservés. Guiochet [Guiochet, 2003] dans son travail sur la maîtrise de la sécurité en robotique de service propose des échelles d'évaluation adaptées au domaine investigué. Par exemple l'effet d'une défaillance est caractérisé par un impact local, de niveau supérieur ou système, et sa gravité est évaluée sur une échelle allant de 1 (catastrophique) à 5 (négligeable).

Pour notre part nous allons chercher à projeter notre analyse dans la dimension de l'architecture de contrôle en nous attachant à ne retenir que les informations qui seront intéressantes pour identifier les fautes pertinentes ainsi que les modules fautifs correspondants et pour guider le processus de diagnostic (détection et localisation) et le recouvrement. La table de synthèse que nous proposons est présentée table 4.2.

Mode_de_fonctionnement

MODULE	MODE DEFAILLANCES		CAUSES DES DEFAILLANCES			SEVERITE	ACTIONS		
			Fautes internes		Modules liés fonctionnellement		Observation		Informations Recouvrement
Fonction	classe	Type	Identificateur	Information			Identificateur	Principe	
MOD1(Fct_2)	Partielle		ID_faute_i	Faute_i	{Mod1(fct_i)[Pp+Cp], Mod2(fct_j)[Ct], ..., Modn(fct_k)[Cp]}	Ignorée	OBS_Faute_i	Principe_Faute_i	Recouvrement Faute_i
	Complète	trans.	ID_faute_j	Faute_j	{Mod1(fct_i)[Ct+Pt], Mod2(fct_j)[Pp], ..., Modn(fct_k)[Pt]}	faible	OBS_Faute_j	Principe_Faute_j	Recouvrement Faute_j
		perm.		Faute_l		fatale			
MOD1(Fct_2)	Partielle					faible			
	Complète	trans.	ID_faute_i	Faute_i	{Mod1(fct_i)[Ct+Pt], Mod2(fct_j)[Pp], ..., Modn(fct_k)[Pt]}	grave	OBS_Faute_i	Principe_Faute_i	Recouvrement Faute_i
		perm.		Faute_m		fatale			
MOD2(Fct_1)	Partielle					Ignorée			
	Complète	trans.	ID_faute_k	Faute_k	{Mod1(fct_i)[Ct+Pt], Mod2(fct_j)[Pp], ..., Modn(fct_k)[Pt]}	faible	OBS_Faute_k	Principe_Faute_k	Recouvrement Faute_k
		perm.		Faute_n		grave			

Table 4.2 – Tableau de synthèse AMDEC

Une différence importante avec la démarche classique AMDEC vient du fait que nous ancrions notre analyse dans la dimension architecturale. En effet, notre étude ne peut se restreindre à l'analyse des fonctions du système pris dans sa globalité. Nous devons développer la démarche AMDEC pour chacun des modes de fonctionnement envisagés du robot. En effet, la sévérité d'une défaillance et donc la présence d'un module fautif au sein de la tâche robotique courante dépendent, au moins en partie, des fonctionnalités encore opérationnelles pour ce mode de

fonctionnement. En conséquence nous déploierons une analyse AMDEC au niveau de chacune des fonctions impliquées dans un mode de fonctionnement donné du robot.

En ce qui concerne la notion de fonction, il faut remarquer qu'un module de l'architecture de contrôle permet de mettre en œuvre une ou plusieurs des fonctionnalités nécessaires au déploiement d'une loi de commande et des mécanismes de tolérance au fautes. Ainsi nous considérons que la déclinaison de la démarche AMDEC par fonction peut être réalisée, dans notre cas, par l'analyse de la défaillance d'un module et des fonctions qu'il supporte : `Mod1(Fonction1)`, `Mod1(Fonction2)`, etc.. Chaque ligne du tableau va ainsi correspondre à un module ou une fonction mis en œuvre dans le mode de fonctionnement analysé.

Comme nous l'avons vu, l'occurrence d'une faute peut se propager au sein du système et engendrer une défaillance complète ou partielle de la fonctionnalité considérée. A cette caractéristique s'ajoute une information relative à la dimension temporelle en qualifiant la défaillance de permanente ou transitoire. Ces caractéristiques sont indispensables à l'identification des fautes et au choix de la solution de recouvrement. Elles se retrouvent donc dans le tableau de synthèse dans les colonnes `Mode de défaillance` et `Type de défaillance`.

Pour chacun de ces modules et des fonctions qu'il implémente il est à présent nécessaire de lister l'ensemble des fautes pouvant atteindre ce module. Les diagrammes d'Ishikawa nous ont permis de supporter la démarche mentale nécessaire à l'identification des fautes. Pour chaque modes de défaillance du module, les différentes origines potentielles ont été balayées. Chacune de ces fautes est repérée par un identificateur unique $\texttt{IdFaute}_i$, et accompagnée dans le tableau d'une description textuelle de cette faute ($\texttt{Faute}_i$).

Au sein de ces diagrammes d'Ishikawa nous avons vu que les défaillances d'un module (ou fonction) pouvaient être liées aux défaillances d'un autre module (ou fonction). Nous ajoutons explicitement dans notre tableau de synthèse les défaillances des modules potentiellement génératrices de défaillances. Cette information pourra être utilisée lors de l'étape de diagnostic en remarquant que, pour un module donné $\texttt{Module}_k$, la défaillance d'un module situé fonctionnellement en amont $\text{Module}_{k-amont}$ peut entrainer un dysfonctionnement du module considéré ($\texttt{Module}_k$). Nous précisons donc dans notre tableau l'ensemble des modules amont ($\texttt{Mod1(fct}_i\texttt{)}$, .., $\texttt{Modn(fct}_k\texttt{)}$). Pour chacun d'eux les modes de défaillance associés (P (partiel) ou C (complet)), pouvant entrainer le dysfonctionnement du module considéré `Module_`k sont aussi identifiés.

$$\{\texttt{Mod1(fct}_i\texttt{)[C.t+P.t], Mod2(fct}_j\texttt{)[C.p], ... , Modn(fct}_k\texttt{)[P.p]}\}$$

Pour chacun des modes de défaillances des modules (ou fonction), nous allons alors estimer la sévérité engendrée sur le mode de fonctionnement concerné.

Nous noterons ici que nous n'introduisons pas, comme cela est usuellement fait dans la méthodologie AMDEC, une notion de criticité d'une défaillance liée à la probabilité d'occurrence de cette dernière. Nous supposons ici que l'expert en charge de l'élaboration du tableau AMDEC envisage toutes les causes pouvant expérimentalement être observées. A contrario, toutes les

causes non identifiées lors de l'expertise sont considérées comme ayant une probabilité d'occurrence négligeable.

Enfin, tout comme dans la démarche AMDEC classique, nous nous attachons à identifier un moyen d'observation de l'occurrence de la faute. Ce point revêt une importance centrale dans notre démarche car il est au cœur du processus de détection et donc de diagnostic que nous proposons. Pour ce faire, pour chaque faute identifiée, nous précisons le mécanisme (dans la colonne `Principe`) qui permettra l'observation et donc la détection de la faute. Nous lui associons là aussi un identificateur unique $\texttt{ObsFaute}_i$.

Contrairement à la démarche AMDEC nous n'identifions pas explicitement les actions à mettre en œuvre lors de l'occurrence d'une faute. En effet c'est le mécanisme de recouvrement qui aura en charge cet objectif une fois l'étape de diagnostic achevée. La colonne `Information Recouvrement` utilisée permet de donner des renseignements utiles à prendre en considération lors de la gestion du recouvrement.

L'analyse AMDEC que nous avons conduite a donc permis de mettre en évidence, pour chaque mode de fonctionnement du robot, un grand nombre d'informations qui vont être utiles au déploiement du processus de tolérance aux fautes. Nous allons maintenant détailler comment celles-ci sont employées pour construire l'architecture de contrôle du robot, en commençant par les éléments nécessaires à la détection de l'occurrence de fautes.

4.4 Intégration des mécanismes de tolérance aux fautes

Dans cette section, nous allons présenter l'impact de la méthodologie d'analyse du robot en terme de modifications sur l'architecture de contrôle. L'analyse AMDEC a permis de mettre en avant un ensemble de défaillances à détecter et identifier, ainsi qu'un ensemble de fautes à diagnostiquer. Dans un premier temps, nous décrirons les mécanismes d'observation intégrés au sein de l'architecture pour détecter l'occurrence de défaillances. Nous aborderons ensuite le diagnostic des fautes basé sur le résultat de cette observation. Pour finir, nous détaillerons les moyens de recouvrement mis en œuvre au sein de l'architecture pour répondre aux détections des défaillances.

4.4.1 Observation de l'architecture de contrôle

Les systèmes robotiques disposent en général de mécanismes d'observation permettant de vérifier l'opérationnalité de certains de leurs composants matériels (micro-contrôleur, communication, etc...). Cependant ces dispositifs ne peuvent répondre à une surveillance en temps réel du comportement de l'ensemble d'une architecture logicielle et des composants qu'elle utilise.

En effet l'observation complète de l'architecture de contrôle doit répondre à de nombreuses contraintes. Elle doit s'insérer aisément dans les modes de fonctionnement exécutés par l'architecture sans pour autant les perturber. Et, puisque les modes de fonctionnement sont adaptés au

contexte d'évolution du robot et de la mission, il est nécessaire que l'observation évolue aussi contextuellement.

Nous proposons d'intégrer l'observation au sein de l'architecture à l'aide de modules spécifiques, les **modules d'observation**, déployés au sein des modes de fonctionnement réalisés par les *modules de contrôle*. L'objectif d'un module d'observation est de monitorer le comportement d'une fonctionnalité donnée. Cela permet de détecter une déviation du comportement observé par rapport à un ou plusieurs comportements prédéfinis afin de mettre en évidence une défaillance de cette fonctionnalité. Les modules d'observation intègrent donc un ou plusieurs algorithmes relevant des techniques de détection de fautes.

Au sein de l'architecture de contrôle, deux types de modules d'observation sont identifiés selon les liens structurels avec les modules de contrôle du robot. On retrouve des exemples de ces modules d'observation en gris dans la figure 4.5, où ils viennent s'ajouter à la représentation symbolique, en blanc, de la boucle de contrôle d'un robot mobile. On distingue :

1. Un module (`Observation 1`) qui permet de monitorer un ensemble de données d'entrée (ou de sortie) d'un module. Cette classe de modules est adaptée à la mise en place d'algorithmes de détection par contrôle de vraisemblance de la valeur des données.
2. Les modules analysant plusieurs données de l'architecture :
   - Un module (`Observation 2`) qui permet de suivre le comportement d'un module en monitorant ses entrées et sorties. Cette classe de modules est typiquement adaptée à la détection basée modèle.
   - Un module (`Observation 3`) qui permet de scruter le comportement d'un ensemble de modules. Par exemple, cette classe peut être utilisée pour mettre en œuvre une banque de filtres de Kalman [Kalman, 1960, Roumeliotis et al., 1998] pour la détection de défaillances actionneurs.

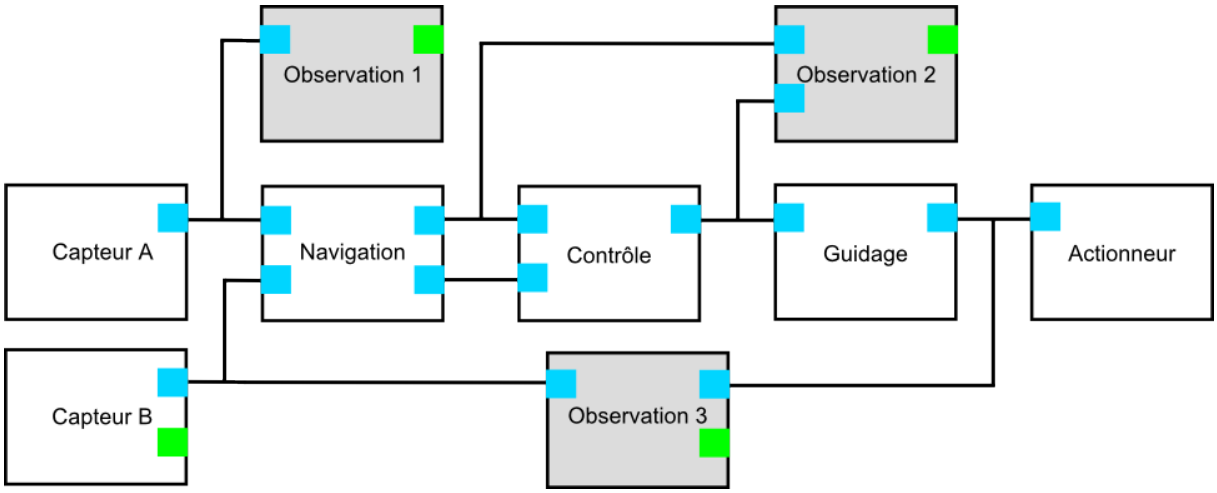


Figure 4.5 – Classes de modules d'observation

En fonction de la nature de l'algorithme implémenté au sein du module d'observation, il est

parfois possible, en plus de la détection d'une défaillance, d'en identifier l'origine (la faute). Par exemple, l'utilisation d'une banque de filtres de Kalman peut permettre d'identifier précisément l'actionneur fautif.

Il est important de noter que l'observation est adaptée dynamiquement aux besoins courants de la mission. En effet les modules d'observation peuvent être ajoutés contextuellement aux différents modules exécutés pour chaque mode de fonctionnement.

Ainsi, les modules d'observation nous permettent d'obtenir en temps réel des informations sur l'état courant du système en identifiant les fonctionnalités valides ou potentiellement défaillantes à un instant donné. Ces informations seront utilisées par le mécanisme de diagnostic pour identifier les fautes affectant le robot.

4.4.2 Le diagnostic

Les modules d'observation nous permettent donc de déterminer l'ensemble des défaillances détectables qui atteignent le robot à un instant donné. L'intérêt du diagnostic est alors de localiser aussi précisément que possible la ou les fautes à l'origine de ces dysfonctionnements pour pouvoir engager par la suite une stratégie de recouvrement pertinente permettant au robot de poursuivre malgré tout sa mission.

L'enchainement de ces différentes phases peut se synthétiser à travers la figure 4.6 que nous expliquerons en détails tout au long de notre discours jusqu'à la fin de ce chapitre en commençant par le mécanisme de diagnostic mis en œuvre.

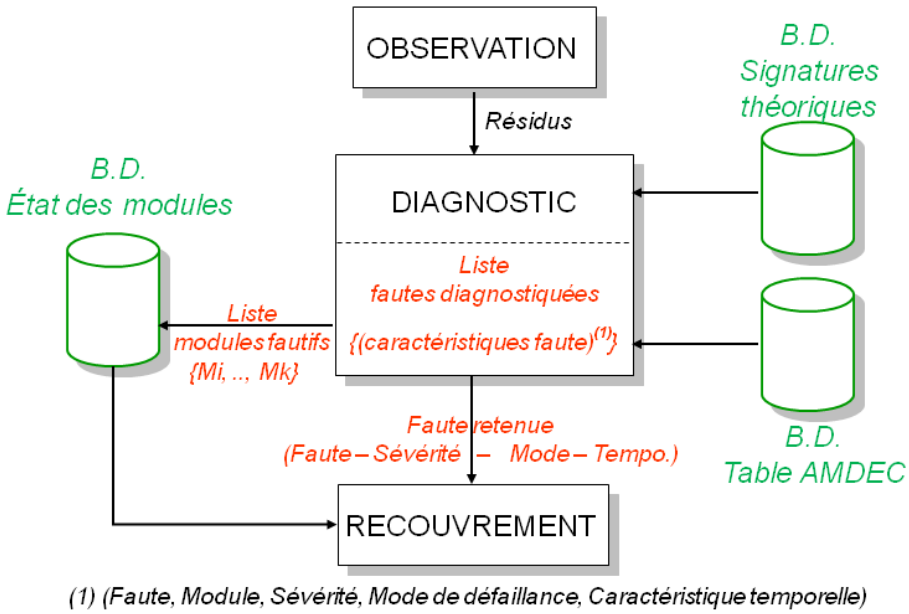


Figure 4.6 – Synoptique du processus d'observation, de diagnostic et de recouvrement

La stratégie de diagnostic mise en œuvre est basée sur l'ensemble des informations fournies par les modules d'observation. Nous allons faire appel aux notions déjà évoquées de résidu et de

signature d'une faute que de nombreux travaux ont développé pour le diagnostic de fautes dans les systèmes commandés [Thelliol et al., 2003, Coquenpot, 2004, Touaf, 2005, Dumont, 2006]. Nous allons dans un premier temps rappeler les notions de matrices de résidus et d'incidence avant de présenter notre méthodologie de diagnostic.

4.4.2.1 Construction de la matrice d'incidence

4.4.2.1.a Rappels sur les notions de résidu et de signature

Un résidu R peut être défini comme étant un signal qui reflète la cohérence (ou la consistance) des données mesurées vis-à-vis d'un modèle comportemental du système [Coquenpot, 2004]. Ainsi pour qu'un dysfonctionnement puisse être détecté, il faut qu'au moins un résidu soit affecté par un sous-ensemble des pannes pouvant toucher le système considéré [Basseville, 1999]. Les méthodes classiquement employées (identification paramétrique, observateur d'états et équations de parité) considèrent n'avoir accès qu'aux informations physiques issues des capteurs (entrées) et envoyées aux actionneurs (commandes / sorties).

Théoriquement, un résidu a une valeur nulle en fonctionnement normal si l'on dispose d'un modèle correct du système, et non nulle en présence de pannes. Cependant les incertitudes sur le modèle ainsi que les bruits de mesure conduisent un résidu à s'écarter de zéro même en fonctionnement non fautif. La difficulté revient alors à définir un seuil de quantification T pertinent permettant la détection effective du dysfonctionnement, tout en évitant les fausses alarmes et les non détection. Soit un symptôme S(R) correspondant à un résidu évalué et quantifié on a alors dans le cas d'une simple comparaison avec le seuil T :

$$S(R) = 1 \text{ si } R > T \text{ et } S(R) = 0 \text{ sinon}$$

Nous noterons ici que souvent, dans les travaux du domaine, les notions de résidu [Coquenpot, 2004] et de symptôme [Thelliol et al., 2003] sont souvent confondues. Nous utiliserons dans la suite du document la terminologie résidu.

Évidemment, au niveau d'un système complexe on dispose d'un ensemble de résidus ou vecteur de résidus. Ainsi la structure du résidu R_i par rapport à un ensemble de fautes F de dimension d est un mot binaire BR_i composé de d bits $B_{i,j}$ positionnés de la façon suivante :

$$B_{i,j} = 1 \text{ si le résidu } R_i \text{ est affecté par le } j^{eme} \text{ élément de } \{F\} \text{ et } B_{i,j} = 0 \text{ sinon}$$

Les informations relatives à la sensibilité ou à la robustesse des résidus vis-à-vis des pannes envisageables du système sont répertoriées au sein d'une table binaire (Figure 4.7) dénommée table des signatures [Coquenpot, 2004], matrice de diagnostic [Thelliol et al., 2003] ou encore matrice d'incidence [Dumont, 2006] au sein de laquelle la composante $C_{i,j}$ correspondant à la i^{eme} ligne et j^{eme} colonne se comporte de la façon suivante :

$$C_{i,j} = 1 \text{ si le } i^{eme} \text{ résidu } R_i \text{ est sensible à la } j^{eme} \text{ faute } P_j \text{ et } C_{i,j} = 0 \text{ sinon}$$

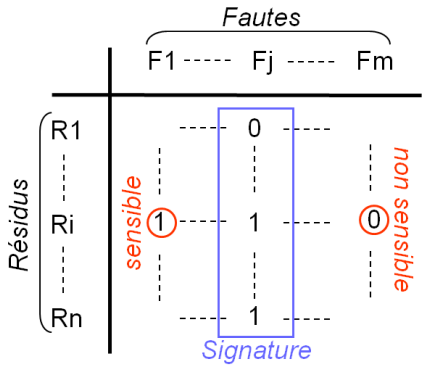


Figure 4.7 – Structure d'une matrice d'incidence

Chaque colonne de la matrice d'incidence constitue alors la signature d'une faute particulière. Celle-ci n'est donc détectable que si elle influence au moins un résidu et par conséquent si la colonne correspondante contient au moins un 1. Deux fautes ne sont pas discernables si elles engendrent des signatures identiques on parle alors de matrice d'incidence non localisante (Figure 4.8a). Celle-ci devient faiblement localisante (Figure 4.8b) et permet de localiser des fautes uniques lorsque toutes les colonnes sont distinctes deux à deux. Enfin la table devient fortement localisante (Figure 4.8c) si elle est faiblement localisante et qu'aucune de ses colonnes ne puisse être obtenue à partir d'une autre en inversant la valeur d'une de ses composantes. Cette propriété permet de s'affranchir de la sensibilité des résidus aux défauts qui peut conduire à un diagnostic erroné.

	f_1	f_2	f_3
r_1	1	1	1
r_2	1	1	1
r_3	1	0	0

(a) Non localisante

	f_1	f_2	f_3
r_1	1	1	1
r_2	1	0	1
r_3	1	1	0

(b) Faiblement localisante

	f_1	f_2	f_3
r_1	1	1	0
r_2	1	0	1
r_3	0	1	1

(c) Fortement localisante

Figure 4.8 – Exemples de matrices d'incidence d'après [Touaf, 2005]

Le diagnostic, c'est-à-dire la localisation précise de l'origine d'un dysfonctionnement nécessite de disposer de vecteurs de résidus satisfaisant des propriétés spécifiques permettant de caractériser de façon unique chaque faute. C'est pourquoi dans les travaux relevant spécifiquement de la thématique du diagnostic, les auteurs établissent au préalable la table des signatures théoriques attendues en faisant appel à des résidus structurés ou directionnels (voir chapitre 1), puis cherchent à construire les résidus permettant de l'obtenir. Le diagnostic consiste alors à comparer et interpréter la signature obtenue à partir de l'observation du système réel avec les signatures théoriques établies.

Nous allons maintenant projeter ces notions dans le cadre de notre étude.

4.4.2.1.b Structure de notre matrice d'incidence

Dans notre cas, nous pouvons transposer les méthodes de diagnostic à l'aide de résidus en posant que ces derniers correspondent aux informations issues des différents modules d'observation intégrés au sein de l'architecture. Ainsi, il est possible de construire une matrice d'incidence des résidus à partir du tableau de synthèse de l'analyse AMDEC et des différents modules d'observations définis. Nous obtenons donc un tableau tel que celui présenté table 4.3.

	Module1		Module2	Module3	
Résidus	IdF1	IdF2	IdF3	IdF4	IdF5
Résidu1	1	0	1	0	0
Résidu2	0	1	0	0	0
Résidu3	0	0	1	1	0
Résidu4	0	0	0	0	1

Table 4.3 – Structure de la matrice d'incidence pour le diagnostic

On y retrouve la forme d'une matrice d'incidence avec les résidus qui constituent les différentes lignes et les fautes touchant les différents modules qui correspondent aux différentes colonnes. Par exemple, lorsque la signature $(1\ 0\ 0\ 0)^T$ est obtenue, cela signifie que le `Module1` est touché par la faute `IdF1`. Nous remarquons qu'une faute peut être liée à plusieurs résidus telle que `IdF3` par exemple. Cette matrice d'incidence sera utilisée par le module de diagnostic pour isoler les fautes à l'origine des défaillances détectées à partir de la valeur de l'ensemble des résidus fournis par les modules d'observation.

L'étape suivante de diagnostic permettra d'identifier le mode de défaillance, les caractéristiques temporelles et la sévérité de la faute la plus pertinente, ainsi que l'ensemble des modules considérés comme défaillants (cf figure 4.3). Nous allons dans un premier temps en décrire le principe en l'illustrant sur le cas le plus simple : lorsque le système est touché par une faute unique qui provoque une défaillance unique. Puis nous argumenterons le cas des défaillances et/ou fautes multiples dans la section suivante.

4.4.2.2 Identification des fautes

4.4.2.2.a Cas d'une seule défaillance, une seule faute

Dans l'hypothèse d'une seule défaillance provoquée par une faute unique, une seule signature sera détectée lors de l'étape précédente, ce qui correspond à une faute unique qui impacte un seul module. L'identifiant de cette faute, $\texttt{IdF}_i$, et le nom du module concerné, $\texttt{Module}_i$, permettront grâce au tableau de synthèse AMDEC (tableau 4.2) d'identifier les caractéristiques de la faute. Dans cette hypothèse de faute et défaillance uniques, on constate que la sévérité de la faute diagnostiquée correspond à la sévérité de la défaillance qu'elle a provoquée.

4.4.2.2.b Fautes ou défaillances multiples

Cependant, l'hypothèse de faute / défaillance unique n'est pas toujours valide. Par exemple, une surcharge ou une perte d'un réseau sans fil de type wifi est une faute transitoire relativement fréquente dont la durée peut être assez longue. Il existe donc un risque non négligeable qu'une autre faute survienne en même temps que cette défaillance de la communication. Nous aurons dans ce cas des fautes et défaillances multiples à un instant donné. De même, il est très probable voire évident qu'une faute unique puisse provoquer la défaillance en chaine de plusieurs modules. Par exemple, une faute sur des capteurs de type sonar va entrainer la défaillance du module de gestion des valeurs de ces capteurs, mais aussi la défaillance des modules utilisant ces valeurs comme par exemple la fonctionnalité de localisation. Cette succession de défaillances est liée au fait que la détection de la défaillance initiale n'est pas forcément instantanée, ce qui laisse le temps aux erreurs de se propager avant d'entamer des actions correctrices.

Nous avons donc introduit des mécanismes de gestion des fautes et défaillances multiples au sein du processus de diagnostic. Dans l'état actuel d'avancement du développement de cette méthodologie, nous proposons les mécanismes de gestion de fautes et défaillances multiples suivants :

- **pour les fautes multiples indépendantes** : ce cas se produit lorsque plusieurs fautes sont détectées à un instant donné, et que chacune d'elles ne produit qu'une seule défaillance. Il n'y a donc pas dans ce cas de propagation en chaine d'une erreur. Le processus de diagnostic identifie chacune de ces fautes indépendamment les unes des autres, et sélectionne la faute de sévérité la plus importante pour guider le mécanisme de recouvrement. La liste des modules défaillants concernés par ces fautes sera fournie au processus de recouvrement.
- **pour les défaillances multiples liées à une faute unique** : dans ce cas toutes les défaillances détectées sont le résultat de la propagation d'une erreur liée à une seule faute. On peut donc supposer que la résolution du problème initial résoudra les défaillances impliquées. La sévérité de la faute et le nom du module sélectionné sont ceux correspondants à la défaillance initiale dans la chaine de défaillances identifiée grâce au tableau de synthèse AMDEC (section 4.3.4).

Il est évident que la combinaison des deux cas précédents peut aussi être rencontrée et donc traitée de la même façon (défaillances multiples liées à des fautes indépendantes).

La dernière phase du processus de diagnostic consiste en la mise à jour, en temps réel, d'une base de données reflétant l'état de l'ensemble des modules de l'architecture. Cette phase est indispensable à la mise en œuvre du processus de recouvrement.

4.4.2.3 Mise à jour de l'état des modules de l'architecture

Au lancement de la mission, tous les modules sont logiquement marqués comme **opérationnels**. Lorsqu'une faute qui provoque la défaillance d'un module est détectée, celui-ci sera marqué comme **non-opérationnel**. Un module non-opérationnel ne pourra plus être utilisé, et

a priori ne nécessite plus d'observation pour la détection des défaillances. A ce stade nous pouvons cependant différencier les modules touchés par des fautes permanentes, et qui seront donc non-opérationnels jusqu'à la fin de la mission, des modules touchés par une faute transitoire, et qui potentiellement pourront redevenir opérationnels. Il sera donc nécessaire de continuer à observer ces modules, et de les différencier de ceux touchés (exclusivement) par des fautes permanentes.

Pour rendre compte de ce comportement, nous introduisons un indicateur supplémentaire, traduisant le fait qu'un module non-opérationnel à l'instant considéré pourra potentiellement le redevenir : la notion de **réversibilité** d'un module. Le retour à l'état opérationnel d'au moins un module réversible sera signalé au processus de recouvrement pour que celui-ci puisse réagir le cas échéant.

La mise à jour de la base de données se fait conformément aux informations produites à l'issue du processus de diagnostic précédent. Ainsi, en cas de défaillances ou fautes multiples, un seul module sera marqué comme non-opérationnel sauf dans le cas de fautes multiples indépendantes.

Nous pouvons résumer l'étape de diagnostic en rappelant qu'elle fournit au processus de recouvrement, à l'aide de la table Amdec, les informations nécessaires au traitement de la faute détectée la plus pertinente : son mode de défaillance, ses caractéristiques temporelles, sa sévérité et l'ensemble des modules considérés comme fautifs. Elle transmet également une information lors du retour à l'état opérationnel d'un module réversible. Cette étape permet donc la mise à jour dynamique de l'état de l'ensemble des modules de l'architecture. Ces informations sont utilisées dans le processus de recouvrement que nous allons maintenant détailler.

4.4.3 Le recouvrement

Le recouvrement du système est l'étape finale de la tolérance aux fautes. Dans le cadre de notre étude, nous souhaitons proposer une large gamme de solutions permettant d'utiliser la plus adaptée à chaque situation. Nous nous appuierons donc essentiellement sur le principe de modalités évoqué dans la partie état de l'art [Lussier, 2007, Morisset et al., 2004]. Chaque tâche robotique est alors déclinée en différentes modalités parmi lesquelles la plus adaptée au contexte du robot est déployée. Ce concept de recouvrement nous semble très intéressant de par sa flexibilité et son adaptabilité aux situations pouvant être rencontrées. Il sera mis en œuvre au niveau des différents types d'interactions Homme-Robot mais aussi entre les différents modes de fonctionnement envisageables pour ces derniers. Le recouvrement sera évidemment fonction du type de faute, c'est-à-dire de sa sévérité, de l'état courant du robot (niveau d'autonomie et mode de fonctionnement) et des ressources disponibles. Afin d'améliorer la fiabilité du système et donc d'assurer la continuité de la mission, le niveau d'autonomie pourra être ajusté en impliquant de façon plus ou moins ponctuelle l'entité humaine dans la boucle de contrôle.

4.4.3.1 Niveaux d'autonomie et modes de fonctionnement

La description de la mission d'un robot mobile tolérant aux fautes nécessite la définition de l'ensemble des niveaux d'autonomie que le robot peut atteindre. Chacun d'eux correspond à un ensemble de modes de fonctionnement définis de façon spécifique. Ainsi, pour une fonction robotique donnée, chaque niveau d'autonomie aura son propre mode de fonctionnement nominal et ses propres modes de fonctionnement dégradés.

4.4.3.2 Principes de base du recouvrement

Cette section présente les solutions de recouvrement adoptées en relation avec la sévérité de la faute diagnostiquée et des modules opérationnels à un instant donné.

Recouvrement par reconfiguration, sévérité faible

Les fautes de sévérité faible perturbent localement une ressource du robot et donc un module de l'architecture. La stratégie de recouvrement permet de reconfigurer le module en modifiant simplement certains de ces paramètres, pour revenir dans un état sain sans changer le mode de fonctionnement courant, et donc le niveau d'autonomie.

Recouvrement par adaptation locale, sévérité moyenne

Les fautes de sévérité moyennes perturbent le mode de fonctionnement actuel qui ne peut être poursuivi en l'état. Une première solution, lorsque cela est possible, est d'utiliser la **redondance fonctionnelle** au sein de l'architecture pour exécuter le mode de fonctionnement dégradé adéquat. Ainsi, si une faute perturbe la fonctionnalité d'un module, et qu'un module redondant réalise une fonction équivalente, une modalité alternative utilisant ce module sera choisie. Cela minimise la dégradation des performances du robot, tout en restant dans le même niveau d'autonomie.

Une seconde solution d'adaptation locale, pour un même niveau d'autonomie, peut être utilisée dans le cas où le module fautif réalise une **fonctionnalité non-essentielle** ne pouvant être remplacée. Cette solution conduit à choisir un mode de fonctionnement (modalité) dégradé pénalisant les performances du robot. Nous définissons une fonctionnalité comme non essentielle si malgré sa disparition la tâche robotique peut toujours être réalisée mais de façon moins performante.

Ce qui vient d'être énoncé démontre implicitement que pour un niveau d'autonomie donné, il existe une liste ordonnée des modes de fonctionnement classés en fonction de la performance correspondante du robot. Celle-ci est évaluée selon l'efficacité des algorithmes mis en œuvre et des risques encourus par le robot et son environnement.

Recouvrement par ajustement d'autonomie, sévérité grave

Les fautes de sévérité graves affectent une **fonctionnalité essentielle** du robot et perturbent trop fortement le mode de fonctionnement courant pour maintenir son niveau d'autonomie actuel. Dans ce cas il n'existe aucune possibilité de mode de fonctionnement dégradé, nous proposons donc d'utiliser les capacités d'un opérateur comme **redon-**

dance fonctionnelle humaine. Le niveau d'autonomie devra donc être ajusté. L'interaction Homme-Robot pourra alors être ponctuelle ou permanente en fonction de l'analyse du problème effectuée par l'opérateur. Cela implique que la disponibilité du module de communication entre le robot et l'opérateur est critique pour cette solution de recouvrement. De même avant, il est nécessaire de fournir à l'opérateur les moyens d'appréhender le contexte d'évolution du robot ainsi que les moyens pour interagir avec le robot.

Recouvrement par mise en sécurité, sévérité fatale

Les fautes de sévérité fatales perturbent des ressources critiques indispensables à l'exécution de la mission et qui ne peuvent être suppléées par l'Homme (énergie, moyens de locomotion,..). Cela conduit donc fatalement à une fin prématurée de la mission. Cette solution de recouvrement intègre de plus la **mise en état sûr** préalable du robot avant son **arrêt** complet.

La figure 4.9 illustre ces mécanismes de recouvrement par un diagramme d'états associé à un niveau d'autonomie donné : les fautes de sévérité faible et moyenne entrainent des évolutions entre les différents modes de fonctionnement de ce dernier ; les fautes de sévérité grave conduisent à un changement de niveau d'autonomie, tandis que les fautes fatales mènent irrémédiablement à l'état d'arrêt du robot.

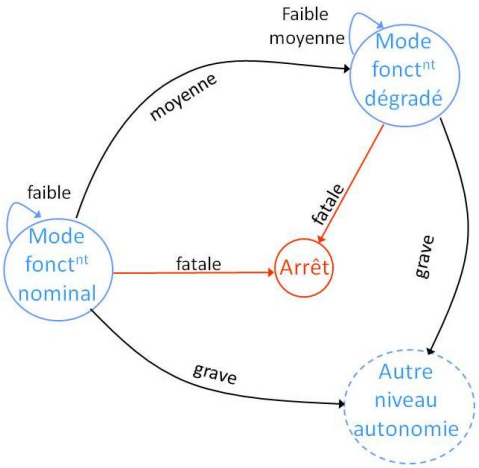


Figure 4.9 – Principes de recouvrement pour un niveau d'autonomie

L'analyse qui vient d'être réalisée n'intègre pas le critère de disponibilité des modules ni celui de leur réversibilité. Le paragraphe suivant aborde cette dimension dans l'analyse du processus de recouvrement.

4.4.3.3 Prise en compte de la disponibilité des modules

La section précédente présente les principes du recouvrement en s'appuyant uniquement sur la sévérité de la faute diagnostiquée. Cependant, la solution retenue dépend également de la disponibilité de l'ensemble des modules de l'architecture. En effet, le processus de diagnostic

met à jour la liste des modules opérationnels en fonction des fautes détectées, y compris en cas de fautes et défaillances multiples qui peuvent rendre plusieurs modules non-opérationnels au même instant.

Il faut donc prendre en compte à chaque instant l'état de l'ensemble des modules pour choisir la modalité à exécuter lors du processus de recouvrement. Si la modalité choisie ne peut être exécutée en raison de l'indisponibilité de modules, la solution de recouvrement balaiera les différentes options envisageables par ordre de sévérité croissant jusqu'à l'identification d'une modalité ayant l'ensemble de ses modules disponibles.

Par ailleurs le mécanisme de recouvrement doit aussi prendre en compte le principe de réversibilité associé à certains modules de l'architecture.

4.4.3.4 Prise en compte de la réversibilité des modules

Le processus de recouvrement présenté dans cette méthodologie va plus loin que les principes qui viennent d'être énoncés. En effet, il prend en compte explicitement l'existence de faute de type transitoire et donc l'existence de modules réversibles qui peuvent revenir dans un état opérationnel au bout d'un certain temps.

La détection de la réversibilité de certains modules peut également être utilisée pour l'optimisation des capacités du robot à tout instant. Ainsi une dégradation du mode de fonctionnement ou du niveau d'autonomie n'est pas définitive et peut être réversible si les modules nécessaires à la mise en œuvre d'une solution plus efficace redeviennent opérationnels (figure 4.10). Par exemple, la fin d'une faute transitoire de sévérité moyenne ayant mené dans un mode dégradé peut conduire à retourner dans le mode nominal (arc pointillé). De même, les fautes graves peuvent être transitoires alors que les fautes fatales sont irréversibles.

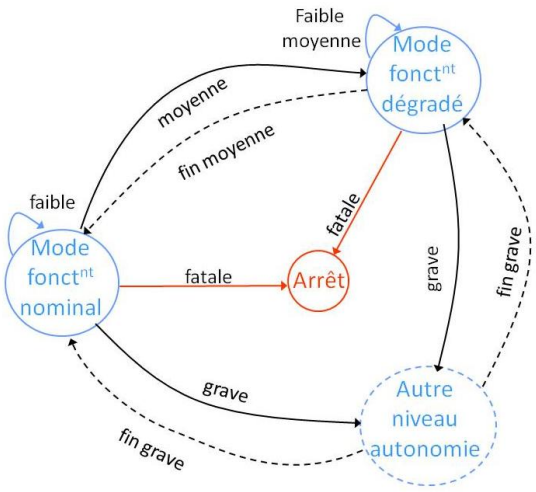


Figure 4.10 – Principes de recouvrement lors de la récupération de fonctionnalité

Notre capacité à exploiter la réversibilité de ces modules conduit de plus à enrichir le diagramme d'états précédent en introduisant un état supplémentaire d'attente (Figure 4.11). Ainsi,

pour des fautes non fatales, lorsqu'aucune solution de recouvrement n'est accessible mais que certains modules peuvent potentiellement redevenir opérationnels, cet état d'attente sera atteint plutôt que de choisir le passage en arrêt définitif. La fin d'une faute transitoire ayant mené dans l'état d'attente peut permettre un retour du robot dans un mode de fonctionnement lié à l'exécution de la mission. Cela implique donc de continuer à observer les modules réversibles en attente d'un potentiel retour en état opérationnel.

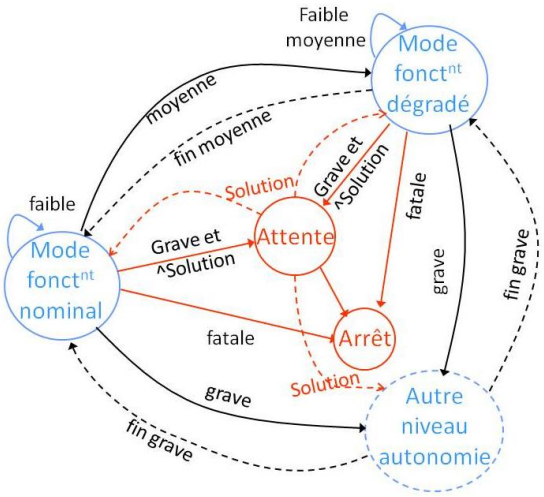


Figure 4.11 – Principes de recouvrement avec mise en état d'attente sûr

Les principes exposés ci-dessus pour la gestion des mécanismes de recouvrement vont maintenant être formalisés sous forme d'algorithme dans la section suivante.

4.4.3.5 Algorithme du processus de recouvrement

Le processus de recouvrement fait appel à deux algorithmes, l'un traduisant la mise en œuvre du principe de modalité énoncé en fonction de la sévérité de la défaillance associé à la faute diagnostiquée, l'autre dédié au traitement de la réversibilité des modules.

4.4.3.5.a Algorithme de recouvrement :

L'algorithme suivant (Figure 4.12) permet de synthétiser le processus de recouvrement dont les principes viennent d'être énoncés. Les entrées de cet algorithme sont :

- la sévérité la plus haute des fautes détectées : Sev_i
- la liste des modules fautifs : $ListeModFautif = \{ModFautif_i\}$ avec $i \in [0, M]$
- l'état courant du robot, qui est constitué de l'indice du niveau d'autonomie courant et l'indice du mode de fonctionnement courant $EtatRobot = (NA_c, MF_c)$
- La base de données de l'état des modules $BDEtatMod = \{EM_i\}$ avec $i \in [0, M]$ et $EM_i \in \{OP, NONOP, REV\}$
- la liste des modules reconfigurables : $ListeModReconfig = \{M_r\}$

– la liste ordonnée des modes de fonctionnement pour chaque niveau d'autonomie.

Si $Sev_i = Faible$ alors
    si $ModFautif_i \in ListeModReconfigurable$ alors
        Reconfig(ModuleFautif)
    sinon $Sev_i =$ Medium
Si $Sev_i = Medium$ alors
    $MF_{tmp} =$MeilleurMF(NA_c)
        si $MF_{tmp} \neq \emptyset$ alors $MF_c = MF_{tmp}$
    sinon $Sev_i =$ Grave
Si $Sev_i = Grave$ alors
    si $BDEtatMod[ModComHR] = OP$ alors
        $NA_c = HRI$ et $MF_c = 0$
    sinon $NA_c = StandBy$ et $MF_c = 0$
Si $Sev_i = Fatal$ alors
    $NA_c = Arret$ et $MF_c = 0$
Renvoi de (NA_c, MF_c)

Figure 4.12 – Algorithme de recouvrement

Dans cet algorithme, la fonction `Reconfig(ModuleFautif)` permet de reconfigurer un module (par exemple en changeant ses paramètres d'entrée, et la fonction `MeilleurMF(`NA`,`$BDEtatMod$`)`) et de rechercher le meilleur mode de fonctionnement d'un niveau d'autonomie NA donné dont les modules sont tous opérationnels. Enfin, le module `ModComHR` est le module chargé de la fonctionnalité de communication entre l'Homme et le Robot.

4.4.3.5.b Algorithme de réversibilité

L'algorithme de recouvrement est complété par celui permettant de gérer la réversibilité des modules. Sur occurrence d'une information provenant du processus de diagnostic indiquant le retour en état opérationnel d'au moins un module réversible, l'algorithme suivant (Figure 4.13) sera exécuté :

L'entrée de l'algorithme : $EtatRobot = (NA_c, MF_c, BDEtatMod)$

4.5 Conclusion

Cette section a permis de présenter une méthodologie globale qui permet d'intégrer de façon systématique des mécanismes de tolérance aux fautes au sein même du processus de conception

Si NA_c = StandBy alors
si $BDEtatMod[ModComHR] = OP$ alors
$NA_c = IHR$ et $MF_c = 0$
Sinon MF_c =MeilleurMF(NA_c)
Renvoi de (NA_c, MF_c)

Figure 4.13 – Algorithme de réversibilité

de l'architecture de contrôle d'un robot mobile. Cette méthodologie se décompose en quatre phases distinctes, dépendantes les unes des autres.

La première phase, l'identification des fautes, est effectuée hors-ligne. Elle permet l'identification des différentes fautes pouvant survenir au cours du fonctionnement du système. Cette analyse est basée sur la méthode AMDEC, qui permet en utilisant une décomposition fonctionnelle du système de lister les défaillances potentielles du système. Les défaillances seront ensuite classifiées suivant leur effet sur le service défaillant (mode de défaillance) et suivant leur conséquence sur la poursuite de la mission (sévérité). Enfin, cette première phase d'identification implique également de s'intéresser aux causes des défaillances identifiées : les fautes. Les origines potentielles de ces fautes sont alors identifiées et classifiées à l'aide de diagrammes Ishikawa. Les informations liées aux défaillances identifiées permettent de définir des politiques de recouvrement à mettre en œuvre afin de réagir de façon efficace et adaptée. L'ensemble des éléments de cette phase est synthétisé dans un tableau d'analyse AMDEC, qui regroupe également les informations spécifiques à l'architecture utilisée. En effet, la décomposition fonctionnelle est effectuée en associant une fonctionnalité à un module architectural. Toutes ces informations du tableau d'analyse seront utilisées lors de la mise en œuvre des phases suivantes : détection, diagnostic et recouvrement.

La deuxième phase concerne la détection des fautes. Elle est basée sur la conception et l'exécution de modules d'observation spécifiques qui observent le fonctionnement des modules fonctionnels en monitorant leurs entrées/sorties. Il existe différents types de modules d'observation, qui mettent en œuvre différents algorithmes spécifiques à la détection des défaillances potentielles précédemment identifiées. Ces modules d'observations permettent ainsi la récupération d'un ensemble de données dans le but de détecter l'occurrence de défaillances qui seront nécessaire à la phase suivante, le diagnostic de l'origine de ces défaillances.

La phase de diagnostic permet, en se basant sur les informations fournies par les modules d'observations et sur l'analyse préalable AMDEC, d'identifier précisément la faute qui est à l'origine de la ou des défaillances qui viennent d'être détectées. La méthode employée ici s'inspire fortement des méthodes basées sur la notion de matrice d'incidence qui permet d'identifier la signature unique d'une faute par un ensemble de résidus. La valeur des résidus, correspond ici aux informations fournies par les modules d'observations, est mise à jour en temps réel lors de la

phase de détection. La phase de diagnostic permet alors d'identifier précisément l'origine et l'occurrence d'une faute. Les cas des défaillances et/ou fautes multiples ont été considérés comme des cas potentiellement plausibles. La méthode de diagnostic présentée a donc été étendue pour le traitement de ces cas particuliers par l'implémentation d'une politique de priorité des fautes détectées selon leur gravité, et par la gestion de chaines de défaillances préalablement identifiées lors de l'analyse AMDEC. Le processus de diagnostic est associé à une étape de mise à jour de l'état interne des modules de l'architecture. Ces modules seront marqués opérationnels, réversible ou non-opérationnels suivant l'occurrence des fautes identifiées et leurs caractéristiques temporelles. Ces informations seront nécessaires à l'établissement des mécanismes de recouvrement durant la phase suivante.

Enfin, la dernière phase consiste dans la mise en œuvre d'une solution de recouvrement adaptée permettant de gérer les situations de défaillances. Notre méthodologie propose différents mécanismes de recouvrement allant de la simple reconfiguration d'un algorithme de contrôle, au changement de modes de fonctionnement qui permet d'exécuter des fonctions robotiques en modes dégradés, et même jusqu'au changement de niveau d'autonomie permettant d'utiliser les capacités humaines lorsque le robot ne peut plus poursuivre sa mission en autonomie complète. Si aucun de ces mécanismes n'est possible, et que le robot ne peut plus poursuivre sa mission dans le contexte actuel, une procédure d'arrêt est prévue qui intègre une mise en état sûr du robot. La solution de recouvrement à mettre en œuvre sera choisie en fonction de la sévérité de la faute, de l'état actuel du robot, et de ses ressources disponibles. Le processus de recouvrement prévoit également la gestion du rétablissement des capacités optimales du robot si les conséquences d'une faute ne sont que temporaires.

L'ensemble de ces mécanismes ajoute à l'architecture de contrôle du robot des capacités de tolérances aux fautes lui permettant d'améliorer la sûreté de fonctionnement d'un robot mobile, en particulier sa fiabilité et sa disponibilité.

Après la présentation de la méthodologie proposée, nous allons présenter le contexte expérimental qui nous permettra dans un second temps d'appliquer la méthodologie sur un cas concret.

Troisième partie

Contexte applicatif et expérimental

Chapitre 5

Architecture de contrôle COTAMA

Pour réaliser un contrôleur sûr de fonctionnement, la prévention des fautes préconise l'adoption d'une démarche de développement et d'outils permettant de limiter la présence de fautes au sein du système commandé et du logiciel de contrôle.

Pour répondre à ces objectifs nous utilisons une architecture modulaire hybride COTAMA développée au laboratoire. Cette architecture reprend les concepts présentés succinctement dans le chapitre 3.1. Nous allons ici les approfondir et présenter son implémentation sur le système temps réels Linux RTAI [1].

5.1 Structuration de l'architecture de contrôle

L'architecture COTAMA [El Jalaoui, 2007] (figure 5.1) s'organise autour de deux univers complémentaires, les niveaux décisionnel et exécutif :

Le niveau décisionnel asynchrone permet de répondre aux intentions formulées par l'Homme et en fonction des obligations du robot de définir le sous-objectif que le niveau exécutif mettra en œuvre.

- Les intentions de l'homme sont définies sous la forme de mission que devra accomplir le robot. Le niveau décisionnel se décompose dans cette architecture en deux sous-niveaux. Le **superviseur global** gère les objectifs de la mission alors que le **superviseur local** contrôle les sous-objectifs, étape des objectifs.
- Les obligations sont prises en compte sous la forme d'**évènements** et représentent la clef de voute de la réactivité décisionnelle dans cette architecture de contrôle [Carbou et al., 2002].

Le second univers est le **niveau exécutif synchrone**. Il permet l'exécution de l'ensemble des sous-objectifs activés par le niveau décisionnel. Chacun d'eux est construit sous la forme d'un assemblage de tâche robotique, qui sont intégrés dans des modules spécifiques présents au plus bas niveau de l'architecture. Chaque module est défini comme un processus exécuté sous le

1. The RealTime Application Interface for Linux - https ://www.rtai.org/

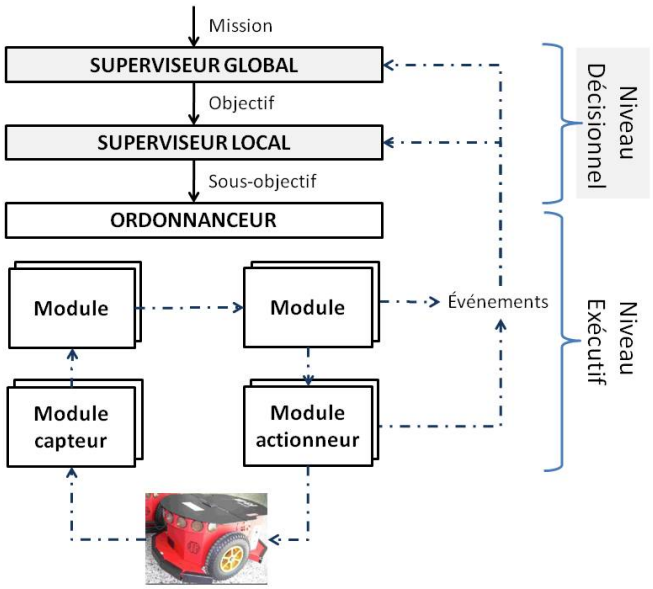


Figure 5.1 – Représentation de COTAMA

contrôle de l'**ordonnanceur**.

Ce dernier se trouve donc à la charnière entre les deux univers, il garantit l'exécution en temps réel des modules, et des sous-objectifs tout en restant réceptif aux évènements issus du niveau décisionnel prioritaire.

L'architecture de contrôle utilise un intergiciel ou middleware permettant de se distancer des problèmes d'implémentation du contrôleur en créant une couche d'abstraction entre le système d'exploitation et celui-ci. Il fournit notamment un ensemble d'outils facilitant la construction de l'architecture de contrôle, comme par exemple un module générique avec ses bibliothèques d'interaction, de configuration et d'exécution.

Nous présenterons donc dans un premier temps, de façon synthétique, la brique élémentaire de notre architecture, le module. Nous évoquerons ensuite les autres outils fournis par le middleware de l'architecture de contrôle. Pour finir nous détaillerons l'ordonnanceur qui est l'organe central et moteur de notre contrôleur.

5.2 Le module - brique élémentaire de l'architecture

5.2.1 Moyens d'interaction des modules

Chaque module est un processus du système d'exploitation. Ses moyens d'activation et de communication sont dissociés pour permettre à l'ordonnanceur d'activer chaque module à l'aide de requêtes sans interférer sur le transfert des données et des évènements entre les modules.

L'entité module (figure 5.2) met à la disposition différents types de ports d'interaction.

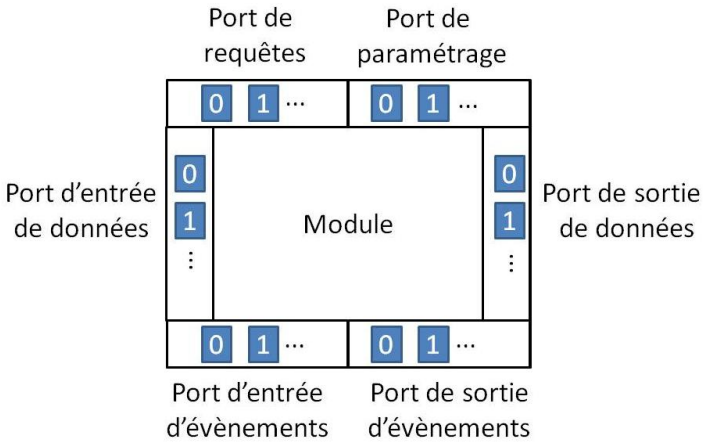


Figure 5.2 – Représentation du module et de ses ports de COTAMA

D'une part on dispose de **ports de transferts** qui ont les caractéristiques suivantes :

- **Ports d'entrée et de sortie de données** : Ces ports permettent le transfert des données entres les modules. Les modules utilisant ces ports sont nécessairement du niveau exécutif. La configuration contextuelle des liaisons inter-modules est faite à l'aide de requête de configuration (voir ci-dessous).
- **Ports d'entrée et de sortie d'évènements** : Ces ports permettent au module de produire les évènements qui seront consommés par les superviseurs au niveau décisionnel.

L'établissement des communications est réalisé à l'aide de ports de communication selon le modèle producteur/consommateur(s) avec une gestion de l'abonnement basée sur une relation publieur/souscripteur. Le module producteur de variables tient à jour la liste des abonnements des modules consommateurs de variables.

D'autre part les **ports d'activation/configuration** sont définis comme suit :

- **Ports de paramétrage (entrées et sorties)** : Ce type de ports permet aux différents superviseurs de transmettre des paramètres aux modules. Le port d'entrée de paramétrage est asynchrone au niveau des modules bas niveau et n'est activé que lorsque la requête de paramétrage est reçue. Un paramètre est une variable transmise par un superviseur vers un module du niveau exécutif uniquement lors de l'activation d'un nouveau sous-objectif.
- **Ports d'entrées/sorties des requêtes** : Ce port permet à l'ordonnanceur d'envoyer des requêtes spécifiques aux différents modules. On distingue :
  - Les **requêtes d'activation** vers les modules du niveau exécutif. L'ordonnanceur envoie une requête d'activation à un module de bas niveau et attend sur son port de réception l'acquittement de fin de traitement.
  - Les **requêtes de paramétrage** informent un module exécutif qu'il va recevoir un paramètre sur un port de paramétrage spécifié.
  - Les **requêtes de configuration** permettent en fonction du sous-objectif en cours et de la composition de la tâche de traitement d'abonner les modules consommateurs aux modules producteurs et de les désabonner lorsque le sous-objectif est arreté.

La définition des flux de communication est contextuelle. Ils seront décrits au niveau des sous-objectifs que nous présentons dans les sections suivantes. Mais avant ça nous allons présenter la structure générique d'un module de l'architecture COTAMA.

5.2.2 Structure interne du module

En plus du modèle d'interaction du module présenté précédemment, le middleware permet de différencier au sein même du module son modèle et son instance dans l'architecture. Une vue du squelette du module, et de son modèle, est présentée figure 5.3.

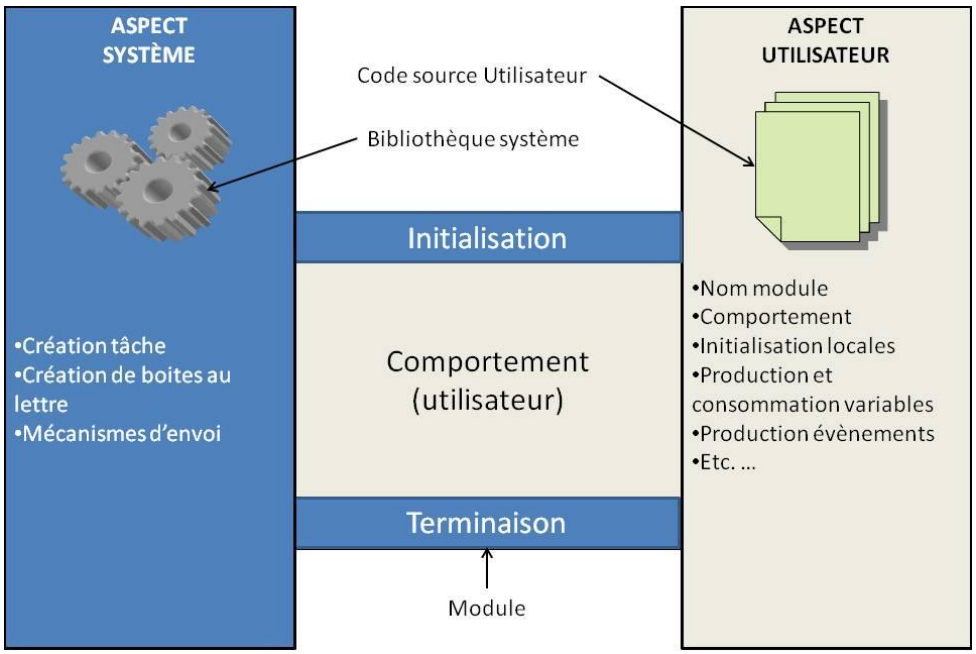


Figure 5.3 – Le squelette du module générique de COTAMA

Le squelette d'un module renferme d'une part les mécanismes communs à tous les modules (création / initialisation / envoi / réception de données / activation / etc...) composant **l'aspect système** et d'autre part l'implémentation du cœur de métier du module constituant **l'aspect utilisateur**.

L'aspect système du module permet la création du processus dans le système temps réel Linux RTAI utilisé pour accueillir l'architecture. Il implémente les ports du module en utilisant les boites aux lettres proposées par Linux RTAI, et définit les processus d'émission des messages. En fin d'exécution du module, il termine le processus temps réel et détruit les différentes boites aux lettres créées.

L'aspect utilisateur est associé, au sein du module, au cœur de métier (algorithmes robotiques, etc.) et à la gestion des interactions avec le middleware : les variables consommées et produites, les évènements produits et les paramètres. Le comportement du module est implémenté sous la forme de trois phases successives :

– une fonction permettant l'**initialisation** de l'espace mémoire des variables ou des services nécessaire à la fonction de traitement.
– la fonction de **traitement** qui sera activée périodiquement par l'ordonnanceur.
– une fonction de **terminaison** pour clore proprement les services et libérer les espaces mémoires des variables.

Les fonctions d'initialisation et de terminaison sont exécutées une seule fois lors de la création et de la suppression des services associés à l'utilisation d'un module.

Cette décomposition permet à par exemple un roboticien (contrôle, vision, etc...) de développer le module sans se soucier des aspects systèmes. L'exécutable du module peut être produit et testé de façon indépendante, seul les ports d'interactions doivent être fixés au préalable. La construction de modules peut alors se faire en parallèle sans pour autant connaitre les algorithmes implémentés dans chaque module.

Les relations entre modules sont définies lors de l'instanciation du contrôleur que nous présentons dans la section suivante où nous décrivons les superviseurs global et local.

5.3 Les superviseurs : Global et Local

5.3.1 Description générale

Le niveau décisionnel asynchrone s'articule dans l'implémentation proposée par [El Jalaoui, 2007] autour de superviseurs. De façon générale, un superviseur permet de choisir la **décision** adaptée aux contraintes définies par l'Homme (**intention**) et par l'environnement du robot (**obligations**).

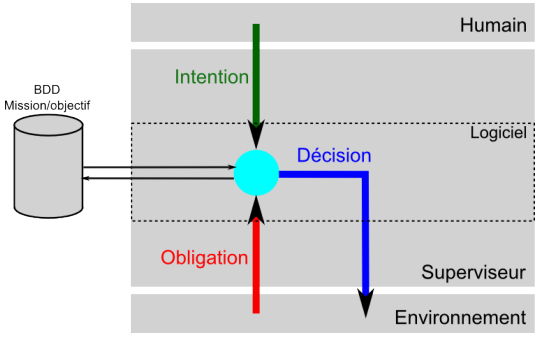


Figure 5.4 – Présentation de la structure décisionnelle des superviseurs

Pratiquement, on distingue le superviseur global qui décompose la mission, intention humaine, en une succession d'objectifs, et le superviseur local qui décline chacun des objectifs en sous-objectifs. Une mission est donc réalisée par le biais de l'exécution successive ou combinée [2] d'un ensemble de sous-objectifs.

2. voir thèse de A. El Jaloaui [El Jalaoui, 2007]

Le superviseur global gère les évènements affectant la mission dans son ensemble alors que le superviseur local limite son champ d'action à l'objectif qu'il gère.

Les prochains paragraphes décrivent la logique d'implémentation d'un superviseur.

5.3.2 Implémentation

La construction d'un superviseur s'appuie sur la structure générique d'un module. Cependant pour permettre un comportement fortement réactif aux évolutions du contexte d'une mission, les modules superviseurs ont une priorité d'exécution supérieure à celle des modules exécutifs. Pour des raisons de hiérarchie décisionnelle un superviseur global est plus prioritaire qu'un superviseur local. Par contre les algorithmes qui les composent sont identiques.

L'algorithme au cœur du superviseur joue la machine à états définie dans la base de données associées grâce à un vocabulaire dédié. Lors de la réception d'une mission (objectif) le superviseur se configure pour pouvoir être réactif aux évènements définis (en lançant des timers ou en s'abonnant aux évènements des modules de contrôle) pour les pré et post conditions des objectifs (sous-objectifs). Lorsqu'un évènement est activé l'état correspondant et lancé ou stoppé.

5.3.3 Vocabulaire

Les fichiers de configuration des superviseurs sont décrits à l'aide d'un langage reposant de façon imagée sur un alphabet, permettant d'exprimer un « raisonnement » ayant un haut niveau d'abstraction :

- L'alphabet est composé de **lettres** associées aux **Modules**.
- L'assemblage de lettres en **mots** représente la composition ordonnée des modules d'un **sous-objectif**.
- Une **phrase** est construite à l'aide d'un ensemble de mots tout comme un **Objectif** de sous-objectifs.
- Une succession de phrases forme un **texte** ce qui conduit la **mission** à être décrite en enchainant les Objectifs.

Pour que ce vocabulaire puisse transcrire le comportement de la mission un ensemble de règles grammaticales a été développé.

5.3.3.1 Règles « grammaticales »

Une base de règles « grammaticales » est définie pour permettre la description des missions et objectifs. La description des sous-objectifs et des modules est présentée dans la section suivante relative à la description de l'ordonnanceur. Nous présentons ci-dessous, la description de missions et d'objectifs.

Les règles grammaticales permettent de décrire la machine à états. Chaque état possède une pré-condition qui autorise son activation ainsi qu'une post-condition qui permet sa désactivation.

La description d'une mission (objectif) à partir d'objectifs (sous-objectifs) se fait en définissant son nom et ses paramètres. Ensuite les objectifs (sous-objectifs) paramétrés sont décrits et leur activation conditionnée par des équations logiques de pré et post-condition décrites grâce à un vocabulaire dédié.

5.3.3.2 Les équations logiques

Les équations logiques permettent de gérer l'activation et la désactivation des états qui sont sensibles à différents types d'évènements faisant référence à la description d'une mission (objectif) :

- un évènement d'un module exécutif de nom MOD et du port `x`, décrit par `[Mod:x]`.
- un évènement temporel déclenché après un temps de `y` (unité de temps) écoulé depuis le début de la mission (objectif), décrit par : `[tps=y s]`
- un évènement temporel déclenché après une durée d'exécution de `z` (unité de temps) d'un objectif (sous-objectif), décrit par : `[durée=z min]`
- un évènement déclenché lors de la terminaison d'un autre objectif (sous-objectif) `n`, décrit par : `[endof(n)]`

Un exemple de description est proposé dans la table 5.1.

ma_mission()
{
1 : [tps=1ms] mon_objectif_1() [MOD :0] ;
2 : [endof(1)] mon_objectif_2() [durée=10min] ;
}

Table 5.1 – Exemple de description d'une mission(objectif)

La mission **ma_mission()** est donc décrite ici à l'aide de deux objectifs. Le premier **mon_objectif_1()** est lancé une milliseconde après le début de la mission. Dès son arrêt, provoqué par l'évènement du port 0 du module **MOD**, la pré-condition **endof(1)** de **mon_objectif_2()** est validé. Cet objectif sera donc actif pendant une durée de dix minutes.

La composition des évènements est faite à l'aide des opérateurs logiques ET (AND) et OU (OR). Tous les évènements sont discernables et seront traités selon leur ordre d'apparition.

Donc la composition ET(AND) de plusieurs évènements nécessite d'associer une durée de mémorisation à chacun d'entre eux [3].

5.3.4 Conclusion

Le niveau décisionnel asynchrone de l'architecture COTAMA est décrit à l'aide de deux machines à états qui évoluent en fonction des évènements issus du niveau exécutif. Ces machines à

3. voir thèse de A. El Jaloaui [El Jalaoui, 2007]

états sont construites à l'aide de fichiers de configuration appelés vocabulaire.

Le niveau décisionnel de l'architecture choisit l'ensemble des sous-objectifs que le niveau exécutif doit mettre en œuvre. Leur exécution est gérée par la clef de voute de l'architecture, l'ordonnanceur.

5.4 L'ordonnanceur - clef de voute de l'architecture

Comme nous l'avons déjà vu dans la figure 5.1, l'ordonnanceur est un pont entre l'univers exécutif et l'univers décisionnel. Il doit donc répondre aux demandes asynchrones formulées sous la forme de sous-objectifs par le niveau décisionnel. Il doit aussi manager le niveau exécutif en gérant l'activation temporelle de chacun de ses modules.

L'ordonnanceur contrôle l'exécution des modules, le respect de leurs contraintes d'enchainement, ainsi que celui des contraintes temporelles associées aux modules et au sous-objectif dans son ensemble. Il est notamment important d'assurer une activation périodique des modules et de la boucle de contrôle-commande pour respecter les hypothèses faites, lors de la conception, au niveau algorithmes implantés (loi de commande, etc.). De plus, parfois, l'utilisation d'une instrumentation spécifique peut imposer des contraintes supplémentaires à l'ordonnanceur notamment en milieu sous-marin [El Jalaoui, 2007].

Nous présentons maintenant les contraintes que l'ordonnanceur prend en compte avant de préciser la stratégie d'ordonnancement des modules exécutifs choisie. Nous évoquerons ensuite l'implémentation de l'ordonnanceur et notamment les fichiers de configuration qu'il utilise.

5.4.1 Les contraintes de l'ordonnancement

5.4.1.1 Définition des tâches : impact des contraintes d'instrumentation

L'architecture COTAMA a été développée pour contrôler un robot sous-marin torpille de type Taipan [El Jalaoui et al., 2005]. Ce milieu d'évolution induit des contraintes spécifiques par exemple pour l'utilisation des capteurs acoustiques. D'une part ils ne peuvent être mis en œuvre simultanément de façon à ne pas engendrer d'interférences (figure 5.5). D'autre part, la vitesse de propagation des ondes acoustiques dans l'eau induit un temps de réponse non négligeable. Pour répondre à ces problèmes et utiliser au mieux le temps processeur il a été choisi de distinguer les modules en tâches d'acquisition et tâches de traitement.

- Une **tâche de traitement** est potentiellement préemptible et retardable. Elle est notée $T :< r, C, R, P >$ et définie par quatre paramètres (r date de réveil, C durée de traitement, R délai critique maximal, P période d'activation).
- Une **tâche d'acquisition** ne peut être préemptée ni retardée car elle a un lien direct avec l'instrumentation. Elle est notée $A :< r, C-, L, C+, R, P >$ et fait en plus appel aux paramètres ($C-$ durée d'insonification du capteur, L durée de latence, $C+$ durée d'attente pour l'écoute de l'onde acoustique).

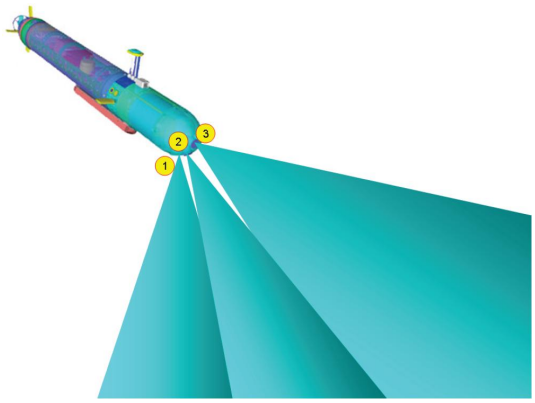


Figure 5.5 – Interférence des capteurs acoustiques d'un véhicule sous-marin torpille

Cette définition de la tâche d'acquisition permet de récupérer du temps processeur pendant la durée L. Par contre le traitement devra être achevé lors de la reprise de la tâche d'écoute de l'onde.

5.4.1.2 Les contraintes d'exécution des sous-objectifs

Il est aussi indispensable d'enchainer l'exécution des modules conformément à l'ordre logique imposé par la boucle de commande robotique pour que chaque module utilise les données les plus récentes. Ces contraintes d'enchainement sont formalisées à l'aide d'un graphe de précédence. Il faut définir l'ensemble de contraintes liées à l'exécution même d'un sous-objectif. Ce dernier est associé, dans l'ordonnanceur, à une sous-configuration $Sc :< r, R, P, E >$ où les paramètres représentent :

- r date de réveil,
- R délai critique maximal,
- P périodicité d'activation du sous-objectif
- $E = A_1, A_2, ..., A_p, T_1, ..., T_q$ ensemble des modules du sous-objectif

La configuration de l'ordonnancement correspond alors à l'ensemble de sous-configurations actives dans l'ordonnanceur. Un algorithme d'ordonnancement [El Jalaoui et al., 2006], rendant actif les tâches ayant la date de terminaison potentielle la plus proche de l'instant courant a été développé pour contrôler leur activation en prenant en compte les contraintes qui viennent d'être évoquées.

5.4.2 Implémentation et vocabulaire

L'implémentation de l'ordonnanceur s'appuie elle aussi sur la structure d'un module générique. Sa priorité d'exécution est supérieure à celle des modules du niveau exécutif mais reste inférieure à celle des superviseurs.

L'ordonnanceur reçoit du superviseur l'ensemble des sous-objectifs à exécuter. Il assurera leur management jusqu'à ce qu'il reçoive l'ordre de retirer l'un d'entre eux de sa configuration.

Les sous-objectifs et les modules activables sont décrits à l'aide de fichiers de vocabulaire.

Le fichier de description des modules (présenté dans la table 5.2) permet de définir les caractéristiques des modules à savoir, son nom **MODx**[4] son temps d'exécution nominal `t` en milliseconde ainsi que les types des différents paramètres qu'il est susceptible de recevoir **(int,float)**.

Fichier modDef.voc
MODx(5ms)({int}) ;
MODy(12ms)() ;
MODz(6ms)({int},{float}) ;

Table 5.2 – Description des modules

Un sous-objectif est caractérisé par son nom **mon-sous-obj** et les paramètres **param1** et **param2** qu'il reçoit. Son exécution est décrite grâce aux informations suivantes et un exemple est présenté dans la table 5.3 :

Fichier soDef.voc

mon-sous-obj(int param1, float param2)
{
// paramètres du sous objectif
PERIODE = 100 ms ;
DELAI_CRITIQUE = 90 ms ;
// module du sous-objectif
MODx(1) ;
MODy() ;
MODz(param1,param2) ;
// graphe de précédence
MODx->MODy ;
MODy->MODz ;
// communication inter-module
MODx :0 -> MODy :0 *1 ;
MODx :1 -> MODy :1 *1 ;
MODx :2 -> MODz :0 *1 ;
MODy :0 -> MODz :1 *1 ;
}

Table 5.3 – Description des sous-objectifs

– Les paramètres temporels du sous-objectif : sa période **PERIODE** et son délai critique d'exécution **DELAI_CRITIQUE**.

4. pour des raisons d'implémentation les noms des modules sont limités à trois caractères.

– L'ensemble des modules exécutés par le sous-objectif ainsi que leurs paramètres. Ces derniers peuvent être statiques et définis au sein du sous-objectif (**MODx(1)**) ou transmis par l'objectif (**MODz(param1,param2)**).
– Le graphe de précédence inter-module. Si l'exécution du module **MODx** doit précéder celle du module **MODy**, on définira alors ce lien de précédence par : `MODx->MODy`
– Les liaisons de communications inter-modules. Si le port `i` du module `MODx` produit une variable sur le port `j` du module `MODy` avec une périodicité `p`, cela sera noté `MODx:i->MODy:j * p`

5.5 Conclusion

L'architecture COTAMA permet de construire un contrôleur temps réel basé sur une approche hybride à deux couches. D'une part le niveau décisionnel composé des superviseurs global et local qui assurent la décomposition de la mission (ou objectif) en objectifs (ou sous-objectifs). D'autre part le niveau exécutif composé de l'ordonnanceur et des modules utilisés pour réaliser les sous-objectifs sélectionnés. Ces derniers sont exécutés par l'ordonnanceur qui garantit la satisfaction des contraintes (précédence, temporelles) définies.

La modularité et l'ouverture de cette architecture permettent de s'inscrire dans une logique de prévention des fautes. Nous l'avons donc retenue dans le cadre de nos travaux pour supporter notre méthodologie de tolérance aux fautes.

Dans la suite de ce manuscrit, pour en alléger la lecture, nous utiliserons la terminologie suivante pour désigner les différentes classes de modules :

– Les modules du bas niveau seront dénommés **Modules**,
– Le module ordonnanceur sera appelé **Ordonnanceur**
– Les modules du niveau décisionnel seront nommés **Superviseurs**.

Le fonctionnement du contrôleur tolérant aux fautes que nous avons développé à partir de cette architecture a été testé dans le cadre d'une mission expérimentale présentée dans le chapitre suivant.

Chapitre 6

Mission de livraison de courrier au sein du laboratoire

Après avoir présenté le fonctionnement de l'architecture de contrôle cible sur laquelle sera déployée notre méthodologie de tolérance aux fautes, nous allons détailler dans cette section, la mission de livraison de courrier que nous souhaitons mettre en œuvre. Dans un premier temps, nous en dresserons le cahier des charges, avant de préciser l'instanciation du contrôleur permettant de réaliser la mission avec l'architecture de contrôle COTAMA mais sans pour l'instant utiliser la démarche de tolérance aux fautes proposée qui sera abordée dans la suite du document.

6.1 Cahier des charges de la mission

Le but de cette mission est de fournir un service de livraison de courriers (ou d'objets) au sein du laboratoire LIRMM. L'environnement retenu est une partie du laboratoire, plus précisément le rez de chaussé de l'extension du bâtiment principal présenté en figure 6.1.

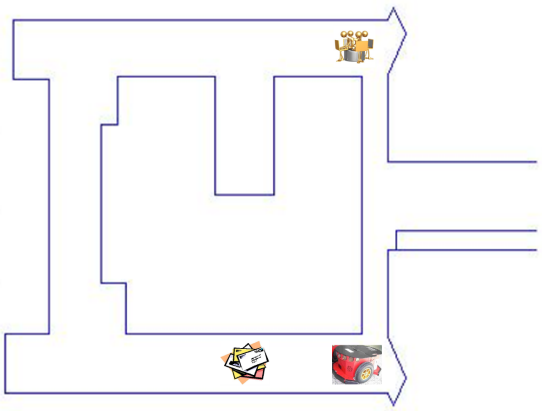


Figure 6.1 – Carte du LIRMM - extension rez-de-chaussée

L'objectif visé à terme au sein de l'équipe de recherche NERO est de disposer d'un ensemble

de robots indépendants sur lesquels les chercheurs pourront expérimenter leurs nouveaux algorithmes robotiques de commande ou de coordination de flottille. Un exemple représentatif de ce type de plate-forme peut être la réalisation de missions de livraison de courrier par une flottille de robots. Ceux-ci pourront être supervisés dans l'accomplissement de leurs tâches par différents opérateurs au travers d'un superviseur de contrôle distant (figure 6.2). Par exemple, lors d'une mission de livraison, le superviseur de contrôle distant réceptionne les demandes formulées par les utilisateurs et les affecte aux robots en fonction de leur disponibilité.

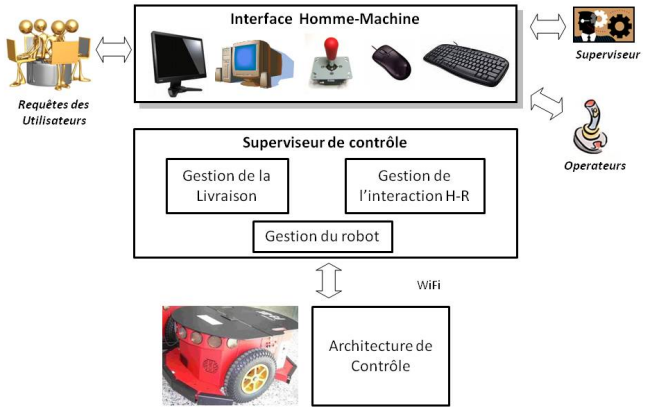


Figure 6.2 – Infrastructure logicielle mise en place pour le service de livraison

Le superviseur de contrôle est composé de trois services :

- Le service *gestion de l'interaction Homme-Robot* qui permet d'une part à l'opérateur de lancer ou d'arrêter la mission et d'autre part de répondre, lorsque cela est possible, aux problèmes du robot.
- Le service *gestion de la livraison* global qui assure l'optimisation des commandes de livraison pour fournir un service correct et performant.
- Le service *gestion des robots* qui gère les interactions entre les précédents services et les robots.

Les différents composants nécessaires à la réalisation de cette mission sont présentés ci-dessous.

6.1.1 Description du superviseur de contrôle

Le superviseur de contrôle, développé sous linux, est composé de différents serveurs de données et algorithmes intégrés au sein de processus communicants à l'aide de boites aux lettres. La figure 6.3 présente un schéma de l'implémentation souhaitée du superviseur de contrôle dont le fonctionnement détaillé est présenté ci-dessous.

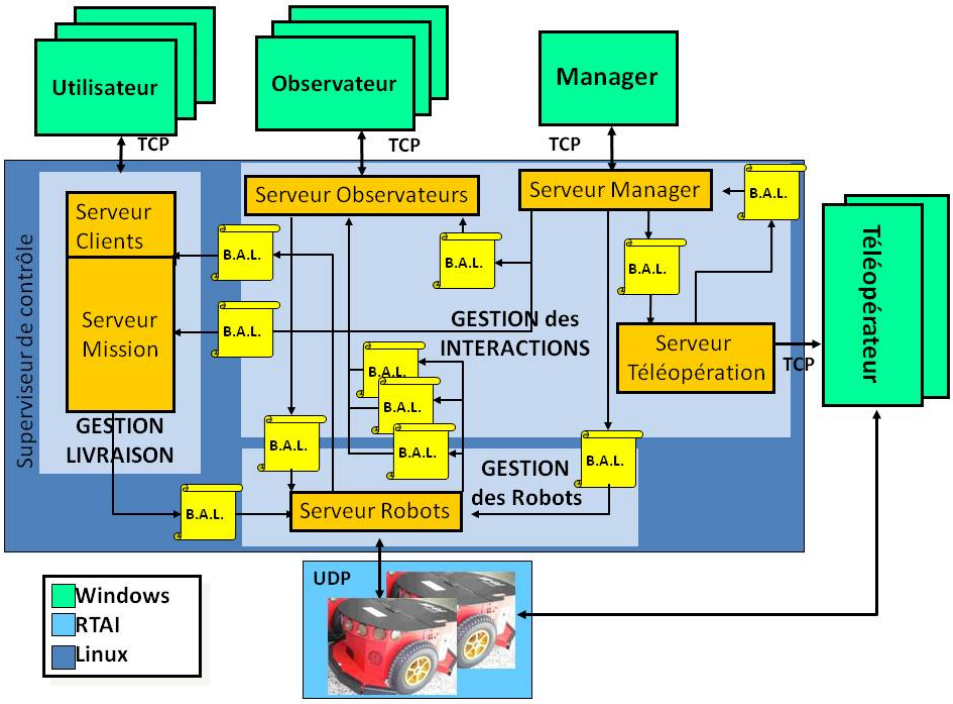


Figure 6.3 – Représentation du Superviseur de contrôle

6.1.1.1 Le service de Gestion des robots

La communication entre le *serveur de robots* et chaque robot fait appel au protocole UDP. Au niveau applicatif, du côté superviseur de contrôle, un vérificateur assure que l'ensemble des informations transmises aux robots ont bien été reçues.

6.1.1.2 Le service de Gestion de la livraison

L'algorithme *Mission* gère l'affectation des missions aux robots en fonction des demandes formulées par les utilisateurs grâce à une interface spécifique, connectée par le biais du *serveur Clients*, permettant aux Utilisateurs Expéditeurs de formuler leurs requêtes vers un *Utilisateur Destinataire*, et d'être informés de l'avancement des livraisons demandées. Dans ce cas, le temps d'interaction étant moins critique, le protocole TCP est utilisé pour gérer la communication et vérifier le bon acheminement des données.

6.1.1.3 Le service Gestion des interactions Homme-Robot

Ce service permet de gérer les échanges d'informations entre les robots et les différentes catégories d'*Opérateurs* impliqués dans la gestion de la mission et pouvant interagir avec le robot. Cette classe, qui vient s'ajouter à celle *Utilisateur* implicitement définie dans le paragraphe précédent, se décline de la façon suivante (figure 6.4), en fonction des rôles joués par les différents intervenants :

Le manager supervise et gère seul le fonctionnement du service et des robots. Il lance les services et répond, en cas de problème, aux requêtes des robots avant de faire appel, si nécessaire, à un téléopérateur.

Le téléopérateur contrôle un robot à distance. Il utilise des interfaces spécifiques selon l'opération à effectuer. Pour optimiser la vitesse de communication entre téléopérateur et robot une connexion directe dédiée est établie. Le protocole de communication est adapté au type de téléopération choisi, de la quantité d'information échangée, et des contraintes temporelles imposées.

L'observateur se contente de monitorer le comportement du service, d'un robot, de celui de certains algorithmes du robot. L'interface permet à l'observateur de définir les informations qu'il souhaite scruter. Il peut donc définir en fonction de son métier (informaticien, roboticien, électronicien, ergonomiste, etc.) les données qu'il souhaite traiter pour analyser le déroulement de la mission. Sa seule interaction avec un robot est la demande d'information. Il reste purement passif par ailleurs.

Par ailleurs, un **mainteneur** peut intervenir physiquement sur le robot sur requête du manager, pour le rechercher, le sortir d'une situation délicate ou effectuer une opération de maintenance.

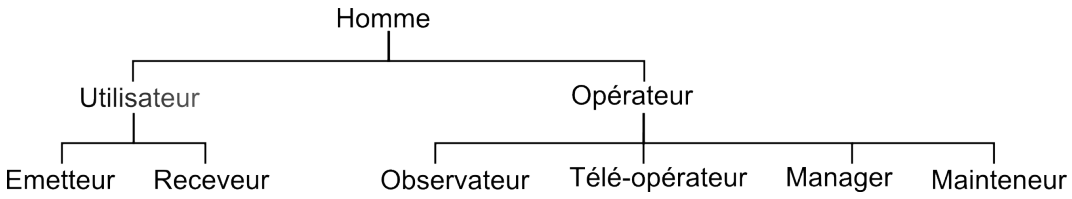


Figure 6.4 – Classification des différents intervenants d'une mission

Après avoir présenté les différentes classes d'intervenants humains impliqués dans la mission, nous allons présenter les robots qui permettent sa réalisation.

6.1.2 Description des robots

Le robot mobile utilisé pour cette mission est de type Pioneer 3-DX, de la société MobileRobots Incorporation ©. Il possède trois roues : deux d'entre elles sont contrôlables séparément à l'aide de commandes différentielles, la troisième est une roue libre (roue folle) placée à l'arrière. Il peut porter jusqu'à 25 kg, et dispose de 16 sonars fonctionnant à une distance maximale théorique de 5 m ainsi que de 10 bumpers entourant le robot, pour détecter d'éventuelles collisions. Il est possible d'ajouter des équipements supplémentaires tels qu'un gyroscope, une caméra embarquée, ou encore un télémètre laser. Le robot, ses capteurs et actionneurs ainsi que l'ordinateur embarqué sont présentés sur la photo 6.5.

La gestion interne des capteurs et des actionneurs est réalisée à l'aide d'un micro-contrôleur H8S relié par port USB avec l'ordinateur embarqué qui accueille l'architecture de contrôle. Un

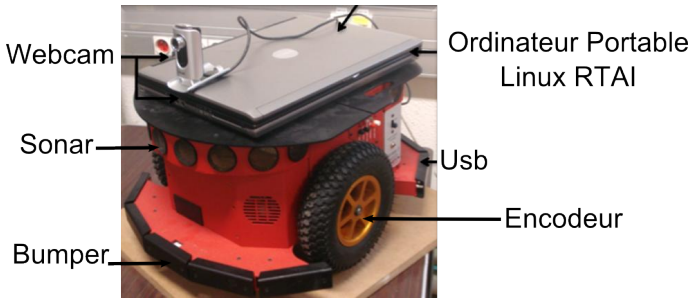


Figure 6.5 – Le robot Pioneer 3 DX et l'ordinateur portable embarqué

langage de communication propriétaire permet à l'architecture de contrôle d'une part de configurer le robot et de piloter ses actionneurs et d'autre part de recevoir à une fréquence choisie, les informations des capteurs sous la forme de paquets dénommés Standard Information Packet (SIP).

Le micro-contrôleur permet de piloter les déplacements du robot avec soit une commande différentielle soit une commande en vitesse. Il permet aussi de récupérer à partir des capteurs les informations suivantes :

- la vitesse des roues à l'aide d'encodeur. Le micro-contrôleur tient à jour la localisation odométrique du robot.
- les distances aux obstacles relevées par les différents sonars.
- les collisions à l'aide des bumpers.
- le niveau de charge de la batterie du robot.

L'ordinateur portable embarqué DELL LATITUDE ou SONY VAIO, supporte le contrôleur du robot. Il utilise le système d'exploitation linux ubuntu 8.10 RTAI [1] temps réel . Une webcam USB Philips de résolution 480x640, lui est connectée pour les besoins de la téléopération.

Nous venons de fixer l'environnement de travail, en cours de développement dans lequel nous nous inscrivons et qui vise à construire, à terme, une plate-forme expérimentale opérationnelle permettant aux différents chercheurs de l'équipe d'évaluer leurs travaux. Cependant, dans le cadre de ce manuscrit, nous nous restreindrons au cas du contrôle d'un seul robot livreur de courrier pour déployer et évaluer notre approche pour le développement d'une architecture de contrôle tolérante aux fautes. Celle-ci s'appuiera sur le contrôleur que nous allons maintenant décrire.

6.2 Instanciation du contrôleur

Cette partie détaille les principes architecturaux et les différents éléments du contrôleur construit à partir du cahier des charges qui vient d'être fixé. Il convient tout d'abord de décomposer la mission envisagée en un ensemble d'objectifs et de sous-objectifs.

1. Ubuntu 8.10 rtai - http://www.linuxcnc.org/

6.2.1 Description de la mission d'un robot

La mission réalisée par un robot autonome consiste donc à récupérer un objet au niveau de son Expéditeur pour le transmettre à son Destinataire. L'algorithme qui contrôle le déplacement du robot a été choisi en collaboration avec nos collègues roboticiens. Celui-ci s'appuie sur un paradigme de suivi de chemin pour atteindre le but fixé. La génération d'un chemin est donc nécessaire pour le mettre en œuvre. Cette dernière s'appuie sur la connaissance de l'environnement du robot représentée par une carte du laboratoire. Mais cet environnement peut être fortement perturbé par la présence d'hommes ou d'obstacles non prévus dans les couloirs. Les déplacements du robot doivent évidemment se faire en totale sécurité pour lui et son environnement.

La mission Livraison choisie est décomposée en quatre objectifs qui seront exécutés séquentiellement :

- **Déplacement du robot vers (Expéditeur)**
- **Réception du courrier**
- **Déplacement du robot vers (Destinataire)**
- **Livraison du courrier**

On peut remarquer que le même objectif déplacement du robot est utilisé deux fois, avec des paramètres différents, la première fois pour aller jusqu'à l'Expéditeur, la deuxième fois pour aller jusqu'au Destinataire.

Cette mission est complétée par une autre Mission Attente-Mission dédiée à l'attente d'une nouvelle demande de livraison.

La décomposition des trois objectifs de la mission Livraison conduit à définir les 4 sous-objectifs suivants pour le robot :

- Génération de chemin (Utilisateur)
- Suivi de chemin (chemin)
- Livraison de courrier
- Réception de courrier

Les deux premiers sous-objectifs composent l'objectif *Déplacement du robot vers* précédemment identifié. La décomposition de la mission *Livraison* peut être synthétisée par la figure 6.6.

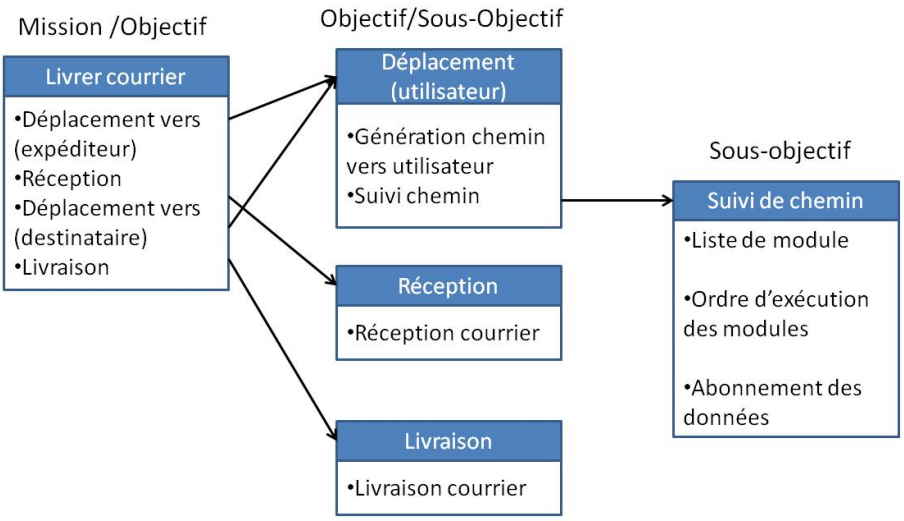


Figure 6.6 – Décomposition de la mission *Livraison*

Dans la suite, nous nous focaliserons sur l'objectif le plus complexe et critique à mettre en œuvre, le déplacement du robot vers un utilisateur (expéditeur ou destinataire).

6.2.2 Déplacement du robot vers un Utilisateur

Cet objectif se décompose en deux sous-objectifs : *Génération de chemin* et *Suivi de chemin*.

6.2.2.1 Génération de chemin

Au cœur de ce sous-objectif, au niveau exécutif de l'architecture, on ne présente ici qu'un seul module celui de planification automatique de chemin PAC. Les modules P3D et NAV sont aussi exécutés lors de la génération de chemin mais seront présentés dans la section suivante lors de la description du *Suivi de chemin*.

6.2.2.1.a Le module PAC : Planification automatique de chemin

Le planificateur de chemin utilise un algorithme de type *probabilistic roadmap methods* (PRM) [Sánchez et al., 2006] pour générer le fichier de points qui sera lu par le module de suivi de chemin (SMZ ou GUI présentés par ailleurs). Le module PAC (table 6.1) reçoit d'une part la position actuelle du robot (**posinitiale**) fournie par le module navigation NAV (détaillé plus loin) et d'autre part la position finale (**position_finale**) du chemin, transmise en paramètre. Par ailleurs le paramètre **carte** permet de définir l'environnement dans lequel doit être généré le chemin.

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiEtatRobot	posinitiale
Sorties			
Paramètres			
	0	TPositionRobot	position_finale
	1	int	carte

Table 6.1 – Ports de communication du module PAC

L'algorithme fonctionne par itération. Un ensemble de points est tiré aléatoirement dans l'environnement. De ceux-ci sont extraits les points que le robot peut rejoindre directement en ligne droite. Enfin, parmi ces derniers, le point le plus proche de la cible est sélectionné pour construire de proche en proche, le chemin menant à la **position_finale** choisie. Pour finir, la succession de segments de droites obtenue est lissée de façon à construire le chemin final qui est mémorisé dans une base de données.

6.2.2.2 Le suivi de chemin avec évitement d'obstacle

La figure 6.7 présente le schéma de commande utilisé pour décrire ce sous-objectif qui est associé au mode de fonctionnement et donc aux différents modules employés pour l'exécuter.

- **P3D** Un module communication gère l'envoi des commandes aux moteurs ainsi que la réception des données capteurs.
- Les données capteurs sont ensuite transmises aux modules :
  - **UST** de gestion des sonars,
  - **ODO** de localisation odométrique,
  - **MCL** de localisation Monte-Carlo.
- **NAV** Un module de navigation fusionne les informations des modules de localisation et construit l'état du robot.
- **SMZ** Un module combiné d'évitement d'obstacle et de guidage définit la cible que doit rejoindre le robot.
- **CTL** Pour finir le module de contrôle calcule en fonction des positions du robot et de la cible les vitesses à appliquer aux roues.

Les modules mis en œuvre dans cette loi de commande sont décrits dans la suite, en détaillant leurs interfaces de communication et le fonctionnement de leurs algorithmes. On trouvera dans les tableaux présentés d'une part les numéros des ports d'entrée, de sortie et de paramétrage utilisés dans l'architecture COTAMA. D'autre part les types et les noms des variables associés à ces données dans les algorithmes employés.

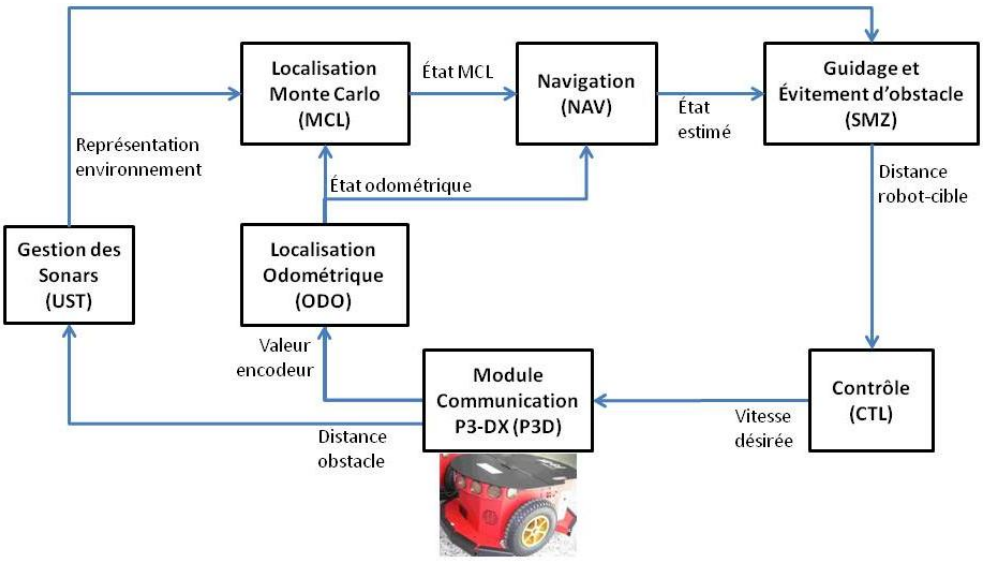


Figure 6.7 – Schéma de contrôle du suivi de chemin avec évitement d'obstacle

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiVitesseRobot	vitesse
Sorties	0	TArchiDonneeUltraSon	ultrason
	1	TArchiEtatRobot	etat
Parametre	0	int	flush_com

Table 6.2 – Ports de communication du module P3D

6.2.2.2.a Le module P3D : Communication avec le Pioneer 3-DX

Le module de communication avec le Pioneer3-DX permet la liaison vers le robot et son micro-contrôleur par une liaison série sur câble **usb**. Il gère l'envoie des commandes et la réception des données provenant de celui-ci en adoptant le format propriétaire imposé (SIP) à une fréquence de 100 ms reconfigurable. D'une part, Les valeurs reçues sont mises en forme avant d'être transmises aux autres modules (tableau 6.2).

- Les valeurs des ultrasons rafraichies permettent de mettre à jour la donnée **ultrason**
- Dans **etat** on construit l'état du robot (position x, y et θ ainsi que les vitesse des 2 roues) à l'aide des informations odométriques fournies par le robot.

D'autre part, le module P3D reçoit les valeurs de vitesses à appliquer aux roues du robot. La commande **vitesse** est donc envoyée au micro-contrôleur qui contrôle cette consigne à l'aide d'un PID jusqu'à la réception d'une nouvelle valeur.

La donnée **flush_com** reçue sur le port de paramétrage permet de demander une réinitialisation de la communication série, et de supprimer les paquets les plus anciens pour lire uniquement le dernier émis par le micro-contrôleur.

6.2.2.2.b Le module ODO : Gestion des odomètres du robot

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiEtatRobot	capteur
Sorties	0	TArchiEtatRobot	etat

Table 6.3 – Ports de communication du module ODO

Le module ODO permet de récupérer les valeurs **capteur** (table 6.3) fournies par le robot au travers du module P3D et vérifie leur fréquence de rafraichissement. La variable **etat** qui définit la localisation odométrique est donc mise à jour à partir de ces informations capteurs.

6.2.2.2.c Le module UST : Gestion des sonars du robot

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiDonneeUltraSon	ultrason
Sorties	0	TArchiUS	us

Table 6.4 – Ports de communication du module UST

Ce module réceptionne les valeurs des capteurs ultrasons, **ultrason** (table 6.4) rafraichies lors du dernier cycle de contrôle par le module P3D.

La position des obstacles dans le repère centré sur le robot (angle et distance) est calculée pour mettre à jour la variable **us**. Pour cela l'algorithme utilise la disposition des sonars sur le robot et les distances à l'obstacle relevé par les capteurs et reçues dans **ultrason**.

6.2.2.2.d Le module MCL : Localisation Monte Carlo du robot par filtre particulaire

Pour répondre aux problèmes de localisation liés aux erreurs inhérentes à l'odométrie, une localisation de Monte Carlo de type filtre particulaire a aussi été mise en œuvre.

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiEtatRobot	etatcapteur
	1	TArchiUS	proximity
Sorties	0	TArchiMCL	robot
Parametre	0	int	reinit

Table 6.5 – Ports de communication du module MCL

La Localisation de Monte-Carlo fusionne les relevés odométriques **etatcapteur** et les informations proximétriques **proximity** (ultrasons) pour déterminer la position du robot à l'aide de

la carte de l'environnement.

Le paramètre **réinit** (table 6.5) permet de réinitialiser l'ensemble des particules de l'algorithme sur la valeur fournie en entrée **etatcapteur**.

L'algorithme de localisation Monte-Carlo utilisé, inspiré de [Zhang and Zapata, 2009], fait appel à un nuage de i particules de position (P_i) qui se déplacent dans l'environnement à la même vitesse que le robot. Chacune d'entre elles représente un robot virtuel. Deux phases successives sont réalisés à chaque itération t de l'algorithme :

Phase d'échantillonnage La phase d'échantillonnage permet de propager sur l'ensemble des positions,de l'itération précédente, des particules P_i, le déplacement réel du robot V_r sur lequel un bruit gaussien δ est ajouté. Cela permet d'obtenir la position simulée à l'instant t de la particule P_i.

$$P_i(t) = P_i(t-1) + (V_r + \delta)dt \tag{6.1}$$

Un facteur de ressemblance Σ_i associé à la particule P_i est calculé en comparant les valeurs des capteurs ultra-sons simulés S_i des particules et les valeurs des ultrasons réels du robot S_r.

La particule ayant le meilleur facteur de ressemblance Σ_i est choisie comme étant la position estimée du robot.

Phase de ré-échantillonnage La phase de ré-échantillonnage à pour but de recentrer les particules. L'ensemble des particules P_i est regénéré par un tirage au sort aléatoire parmi les particules P_i pondérée par le facteur de ressemblance Σ_i. La méthode de ré-échantillonnage est présenté dans [Zhang, 2010]. Les particules ayant une forte ressemblance Σ_i seront donc favorisées lors de cette phase alors que les autres seront, grâce à la succession des itérations, progressivement ignorées.

La variable **robot** contient donc la position de la particule ayant reçu la meilleure « note » ainsi que l'écart type des particules ré-échantillonnées. Elle contient aussi les vitesses de roues fournies par l'odométrie.

6.2.2.2.e Le module NAV : Gestion de la navigation du robot

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiEtatRobot	etatODO
	1	TArchiMCL	etatMCL
Sorties	0	TArchiEtatRobot	etat_actuel
Paramètres	0	int	para_etat

Table 6.6 – Ports de communication du module NAV

Le module de navigation reçoit les informations provenant des deux modules de localisation (**etatODO** et **etatMCL**) pour définir la position du robot **etat_actuel** (table 6.6) qui sera considéré comme état de référence par l'ensemble de l'architecture.

Le paramètre **para_etat** permet lorsque les deux solutions de localisation sont actives de choisir au niveau de ce sous-objectif la localisation prioritaire au sein du module :

– 0 - ODO - Localisation odométrique
– 1 - MCL - Localisation de Monte-Carlo

6.2.2.2.f Les modules SMZ et CTL : Suivi de chemin avec évitement d'obstacle

La méthodologie de suivi de chemin et d'évitement d'obstacle employée fait appel à une approche développée par [Lapierre et al., 2010]. Ces derniers proposent aussi une solution pratique d'évitement d'obstacle qu'ils combinent avec un algorithme de suivi de chemin sans singularité [Lapierre and Indiveri, 2007]. Les auteurs prennent en compte la saturation des actionneurs et prouvent la convergence de la solution proposée.

Nous présentons maintenant l'algorithme de suivi de chemin avant d'aborder la solution d'évitement d'obstacle retenue.

Le suivi de chemin L'algorithme de suivi de chemin permet à un robot de suivre un chemin à une vitesse donnée. Il prend en compte la distance entre le robot et le chemin mais aussi l'angle formé dans le repère global par les angles θ et θ_c (figure 6.8). θ et θ_c représentent les orientations respectives du robot et de la cible.

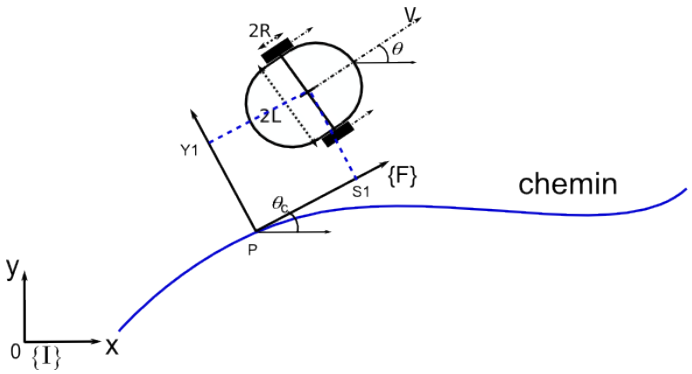


Figure 6.8 – Présentation du suivi de chemin

Dans l'approche retenue, la cible du robot ne sera pas simplement le point du chemin le plus proche du robot, mais une cible virtuelle (dénommée *rabbit*) qui se déplace de façon appropriée sur le chemin. La fonction d'évolution de cette cible dépend de la vitesse du robot et de sa position courante (s_1, y_1) dans le repère de Serret-Frenet $\{F\}$ attaché à la cible virtuelle.

Le suivi de chemin est donc décomposé en deux étapes successives :

– la définition de la position de la cible virtuelle.

– l'élaboration des commandes à envoyer au robot en fonction de sa position courante par rapport à celle du rabbit.

L'algorithme d'évitement d'obstacle que nous allons maintenant présenter s'appuie sur celui de suivi de chemin que nous venons de détailler.

L'évitement d'obstacle La solution pratique [Lapierre et al., 2010] d'évitement d'obstacle utilisée est dénommée SMZ (*Safe Manoeuveing Zone*). Elle définit un chemin local lorsqu'un obstacle est détecté sur le chemin initial. Ce chemin local est un cercle dont le centre est la position de l'obstacle le plus proche du robot et dont le rayon correspond à une distance de sécurité avec l'obstacle. Sur ce chemin, une seconde cible virtuelle est créée et suivie par le robot lors de la stratégie d'évitement d'obstacle (figure 6.9).

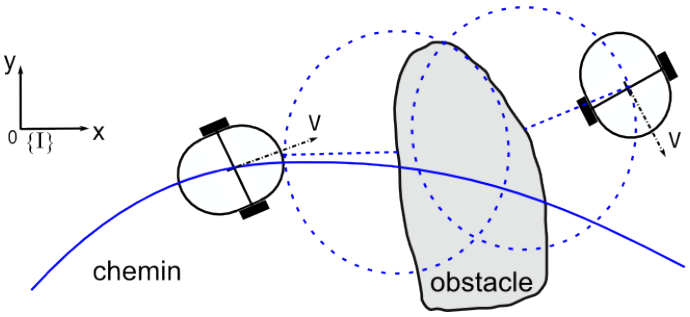


Figure 6.9 – Évitement d'obstacle par SMZ

Pendant la phase d'évitement, à chaque itération de l'algorithme, les deux cibles virtuelles sont calculées : celle sur le chemin initial et celle sur le chemin local. La condition de sortie de cette phase de contournement, qui se base sur l'évolution des orientations des cibles virtuelles, est présentée dans [Lapierre et al., 2010].

Les modules L'algorithme de suivi de chemin et d'évitement d'obstacle est mis en œuvre sous la forme de deux modules dans l'architecture de contrôle.

– Le module SMZ contient les algorithmes de suivi de chemin et d'évitement d'obstacle. La position du robot **etat** et l'environnement représenté par la variable **sonar** permettent de définir la cible virtuelle que le robot doit suivre. Les informations relatives à cette cible sont contenues dans la variable (**Rabbit**) (table 6.7).

	Port	Type de la donnée	Nom de la donnée
Entrée	0	TArchiEtatRobot	etat
	1	TArchiUS	sonar
Sorties	0	TArchiRabbit	Rabbit
Évènement	0	Fin de chemin atteint	event_distance
Paramètres	0	int	UpdatePath

Table 6.7 – Ports de communication du module SMZ

L'évènement **event_distance** est émis lorsque la fin du chemin est atteinte.

– Le second module est le module de contrôle, appelé CTL, qui calcule les vitesses à appliquer aux roues du robot en fonction de la position du robot par rapport à la cible définie par le module SMZ (table 6.8).

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiRabbit	Rabbit
Sorties	0	TArchiVitesseRobot	vitesse

Table 6.8 – Ports de communication du module CTL

Les vitesses des roues souhaitées sont donc émises vers le module P3D à l'aide de la variable **vitesse**.

6.2.2.2.g Les modules redondants pour le suivi de chemin

De façon à pouvoir disposer d'un plus large éventail de choix dans les modes de fonctionnement du robot en cas de défaillances de certains modules ou de difficultés expérimentales nous avons développé un ensemble de modules dit redondants. Les modules GUI pour le suivi de chemin, DVZ pour l'évitement d'obstacle et SIM pour la simulation comportementale du robot sont maintenant décrits.

Le module GUI : Gestion du suivi de chemin Contrairement au module SMZ qui implémente en son sein à la fois l'algorithme de suivi de chemin et d'évitement d'obstacle, le module GUI ne traite pour sa part, que de la problématique de suivi de chemin [Lapierre and Indiveri, 2007].

Ce module (table 6.9) reçoit en paramètre le chemin **UpdatePath** à suivre. Il récupère alors la position courante **etat** du robot pour définir la position de la **cible** virtuelle à poursuivre. Lorsque la fin du chemin (**event_distance**) est atteinte il génère l'évènement correspondant.

	Port	Type de la donnée	Nom de la donnée
Entrée	0	TArchiEtatRobot	etat
Sorties	0	TArchiRabbit	cible
Évènement	0	Fin de chemin atteint	event_distance
Paramètres	0	int	UpdatePath

Table 6.9 – Ports de communication du module GUI

Ce module peut être utilisé en combinaison avec le module DVZ pour développer une stratégie d'évitement d'obstacle s'appuyant sur le principe de zone virtuelle déformable (Deformable Virtual Zone).

Le module DVZ : Évitement d'obstacle Les informations nécessaires au module DVZ sont présentées dans le tableau 6.10. Il reçoit les commandes **vitesse_CTL** générées par le module de contrôle CTL pour suivre la cible virtuelle définie par GUI. Celle-ci est détectée par le biais des informations sonars (**ultrason**). La nouvelle commande **vitesse_P3D** est alors construite par l'algorithme de DVZ pour être ensuite adressée au module P3D qui l'appliquera aux actionneurs.

Ce module fait appel à l'algorithme d'évitement d'obstacle proposé par [Sánchez et al., 2006, Lapierre et al., 2007]. A l'inverse de celui de la SMZ, il est décorrélé du suivi de chemin et permet une réaction en aval de celui-ci pour modifier les commandes envoyées aux actionneurs de façon à éviter les collisions. L'algorithme de la DVZ est procède comme suit :

- La Zone virtuelle déformable du robot est construite.
- Si il y a une intrusion dans cette zone, la réaction nécessaire pour éviter l'obstacle est calculée en fonction des paramètres suivants :
  - La position de l'intrusion par rapport au déplacement désiré,
  - L'aire de l'intrusion.

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiUS	ultrason
	1	TArchiVitesseRobot	vitesse_CTL
Sorties	0	TArchiVitesseRobot	vitesse_P3D

Table 6.10 – Ports de communication du module DVZ

Le module SIM : Simulateur du Pioneer Pour pouvoir pallier à certains problème expérimentaux le module SIM, permettant de simuler le comportement du robot, a été développé. Il calcule les données capteurs que devrait produire le pioneer3-DX en situation réelle.

Plusieurs comportements peuvent être choisis en fonction de la valeur du paramètre **para**. Soit le robot n'est pas physiquement connecté et le module SIM remplace totalement le module

P3D. Soit il simule uniquement un des ensembles de capteurs : les odomètres ou les sonars et il sera possible d'intégrer le robot dans la boucle de contrôle *(Hardware In the Loop - HIL)*.

Le module SIM reçoit en entrées (tableau 6.11) les informations nécessaires à la simulation, **vitesse** pour construire les valeurs des odomètres (**etat**) et **etatcapteur** pour simuler à partir de la position du robot et de la carte de l'environnement, les valeurs sonars (**ultrason**) qui devraient être observées.

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiVitesseRobot	vitesse
	1	TArchiEtatRobot	etatcapteur
Sorties	0	TArchiDonneeUltraSon	ultrason
	1	TArchiEtatRobot	etat
Paramètre	0	int	para
	1	int	carte

Table 6.11 – Ports de communication du module SIM

Nous venons de décrire dans cette partie l'ensemble des modules pouvant être utilisé dans l'objectif *Déplacement du robot vers (Utilisateur)*. A celui-ci s'ajoutent d'autres objectifs et d'autres niveaux d'autonomie qui nécessitent d'établir une communication distante entre le robot et l'Homme par le biais d'un module dédié.

6.2.3 Un module pour l'interaction Homme-Robot : Le module UDP

Lorsque le robot est en attente d'une mission ou d'une confirmation de réception ou de livraison, un module de communication nommé UDP est exécuté pour gérer le dialogue avec le superviseur de contrôle. Il permet, à distance, de lancer une mission et de valider les étapes de réception et livraison d'un courrier. Il permet par ailleurs de réaliser la téléopération.

Ce module est aussi utilisé pour transmettre aux opérateurs les informations nécessaires à la supervision de la mission. Il fournit aux observateurs les informations nécessaires à l'analyse de la mission en cours. Pour les téléopérateurs, il communique les informations nécessaires à leurs prises de décision (image vidéo, état du robot, etc.) et sert aussi d'interface de téléopération pour transmettre au robot les données de contrôle des actionneurs déduites de la position du (ou des) joystick(s).

Dans ce but le module UDP doit potentiellement être abonné à l'ensemble des variables produites dans l'architecture (non explicitées dans la figure 6.12) qui seront émises selon le protocole de communication UDP. Il transmet aux modules qui lui sont abonnés (P3D ou SIM) les commandes **vitesse_joystick** à appliquer. Par ailleurs il peut émettre les évènements **fin téléopération**, **réception** et **livraison** vers les superviseurs lorsqu'il reçoit les informations correspondantes depuis les utilisateurs distants.

	Port	Type de la donnée	Nom de la donnée
Entrées		...	...
Sorties	0	TArchiVitesseRobot	vitesse_joystick
Évènement	0	int	fin téléopération
	1	int	réception
	2	int	livraison

Table 6.12 – Ports de communication du module UDP

Après avoir décrit l'ensemble des modules nécessaires à la mission de livraison du courrier nous allons aborder la description des superviseurs qui lui sont associées.

6.3 Description des superviseurs

La description des superviseurs fait appel à des fichiers de vocabulaires dont la syntaxe a été définie au paragraphe 5.3.3.

Conformément à la description de la mission énoncée au paragraphe 6.2.1 la table 6.13 la décompose en une série d'objectifs. Pour chacun d'entre eux ont retrouvent les conditions d'activation (pré-condition) et les conditions d'arrêts (post-condition).

livraison(int carte, (double exp_x, double exp_y),(double dest_x, double dest_y))		
{		
1 : [tps=1ms]	déplacement(carte,{exp_x,exp_y})	[SMZ :0] ;
2 : [endof(1)]	réception-courrier()	[UDP :1 or duree=10min] ;
3 : [endof(2)]	déplacement(carte,{dest_x,dest_y})	[SMZ :0] ;
4 : [endof(3)]	livraison-courrier()	[UDP :2 or duree=10min] ;
}		

Table 6.13 – Description de la mission - fichier missionDef.voc

Nous retrouvons l'enchainement des objectifs : déplacement du robot de sa position courante vers l'expéditeur (à la position **(exp_x,exp_y)**),réception du courrier, à nouveau déplacement vers le destinataire de position **(dest_x,dest_y)**, avant la livraison finale du courrier.

La réalisation de ces objectifs et leur décomposition en sous-objectif est décrites au sein du fichier de vocabulaire `oDef.voc`. Deux d'entre eux sont présentés dans la table 6.14. Ils suivent la même logique syntaxique que la table précédente.

réception-courrier()
{
1 : [tps=1ms] attente() [duree=10min] ;
}

déplacement(int carte, double x, double y)
{
1 : [tps=1ms] generation_chemin(carte,x,y) [PAT :0] ;
2 : [endof(1)] suivi_de_chemin(carte,0) [SMZ :0] ; *//paramètre=0 : automatique*
}

Table 6.14 – Description des objectifs réception-courrier et déplacement - fichier oDef.voc

- L'objectif **réception-courrier()** utilise le sous-objectif **attente()**. Il permet d'attendre pendant une durée maximale de 10 minutes le signal Utilisateur envoyé par UDP indiquant que le courrier est sur le robot. L'objectif **livraison-courrier()** est similaire à celui-ci.
- L'objectif **déplacement(int carte,{ double x, double y })** gère le déplacement du robot. Dans un premier temps le sous-objectif **generation_chemin(carte,x,y)** construit le chemin à suivre. Dans un second temps le sous-objectif **suivi_de_chemin(int carte,int chemin)** lui succède pour diriger le robot vers sa destination.

La dernière étape de description des superviseurs permet d'associer aux sous-objectifs un ensemble ordonné de modules communicants. Par exemple, le sous-objectif de suivi de chemin est décrit dans la table 6.15.

La description du sous-objectif (section 5.4.2) commence par la définition des **modules** exécutés et des paramètres qui leur sont associés à l'initialisation. L'ordre d'exécution des modules est ensuite précisé grâce au **graphe de précédence**. Pour finir les **communications inter-modules** créées par abonnement sont décrites.

suivi_de_chemin(int carte,int chemin)
//modules du sous-objectif
P3D(1) ;
UST() ;
MCL(carte) ;
NAV(1) ; *// 1 : navigation MCL*
SMZ(chemin) ; *//chemin généré par le module PAC*
CTL() ;
// graphe de précédence
P3D->UST ;
UST->MCL ;
MCL->NAV ;
NAV->SMZ ;
SMZ->CTL ;
// communication inter-module
P3D :1 -> NAV :0 *1 ; *// état*
P3D :1 -> MCL :0 *1 ; *// état*
P3D :0 -> UST :0 *1 ; *// ultrason*
UST :0 -> SMZ :1 *1 ;
UST :0 -> MCL :1 *1 ;
MCL :0 -> NAV :2 *1 ;
NAV :0 -> SMZ :0 *1 ; *// état*
SMZ :0 -> CTL :0 *1 ;
CTL :0 -> P3D :0 *1 ;

Table 6.15 – Description du sous-objectif Suivi de chemin - fichier soDef.voc

6.4 Conclusion

Le contrôleur proposé dans cette partie permet la réalisation de la mission de livraison de courrier en s'appuyant sur les principes architecturaux de COTAMA. Dans cette présentation le mode de fonctionnement autonome de suivi de chemin avec évitement d'obstacle a été mis en avant de par son rôle central dans la réalisation de cette mission.

Nous présentons dans la suite de ce manuscrit l'application de la méthodologie de tolérances aux fautes que nous avons développée et qui vient compléter le contrôleur venant d'être décrit.

Quatrième partie

Application de la méthodologie de tolérances aux fautes

Chapitre 7

Prévision des fautes : Application de la méthodologie sur le cas d'étude

Après avoir présenté la mission ainsi que l'architecture logicielle robotique qui soutiendra sa réalisation, nous allons détailler sur un cas applicatif la mise en œuvre de notre approche pour la tolérance aux fautes. Nous abordons dans un premier temps, dans le cadre du suivi de chemin, l'aspect prévision des fautes et stratégies de recouvrement. Dans un second temps nous décrivons comment cette analyse préalable permet de construire une architecture de contrôle plus sure de fonctionnement.

7.1 Application de l'AMDEC

Conformément au chapitre 4 l'application de la méthode AMDEC est réalisée au niveau de chacun des sous-objectifs envisageables pour une mission. Ces sous-objectifs conduisent à définir implicitement, aux niveaux architectural et physique, des limites d'analyse différentes pour le système considéré.

En raison de son importance pour la mission considérée, nous nous focalisons une fois de plus sur le suivi de chemin autonome SMZ.

7.1.1 Décomposition SADT de la fonction suivi de chemin autonome SMZ

La figure 7.1 présente la décomposition SADT de la fonction **Suivre le chemin en autonomie**.

Elle fait apparaitre une large mise en correspondance des fonctionnalités identifiées avec les modules architecturaux développés. Quelques différences marginales peuvent être cependant observées en raisons des contraintes matérielles ou de facilité d'implémentation.

On peut identifier les correspondances suivantes :

– la fonction **Représenter l'environnement** est réalisée par le module UST.

– la fonction **Localiser par Odométrie** est mise en œuvre par le module ODO.

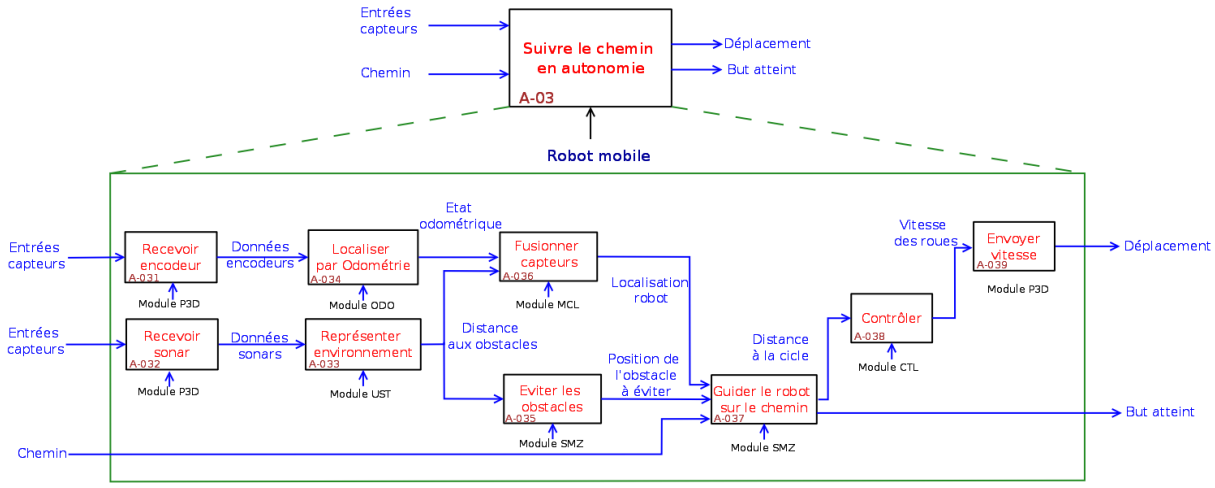


Figure 7.1 – Décomposition SADT de la fonction suivi de chemin en autonomie

- La fonction **Fusionner les capteurs** est déployée par le module MCL.
- Les fonctions **Éviter les obstacles** et **Guider le robot sur le chemin** sont réalisées au sein d'un unique module SMZ.
- La fonction **Contrôler** est assurée par le module CTL.

À ces correspondances évidentes s'ajoutent celles moins directes imposées par la spécificité du lien de communication entre le robot et l'ordinateur embarqué (module P3D) et par le souhait de disposer à tout instant d'un module permettant de connaitre la localisation courante retenue du robot (module NAV).

Le module P3D , centre de communication avec le robot, concentre les fonctionnalités de réception des données capteurs et d'application des commandes aux roues. En fait, il est décomposé en plusieurs entités pouvant être associées à différentes fonctionnalités identifiées dans la figure :

- La fonction P3D-mod représente à la fois la fonction du module et la fonctionnalité **Envoyer vitesse** permettant de commander les roues.
- La fonction P3D-roue associée à la fonction **Recevoir Encodeurs** récupère les informations issues des moteurs et des encodeurs.
- La fonction P3D-sonar liée à la fonctionnalité **Recevoir Sonars**, collecte les données des ultra-sons pour reconstituer l'environnement proche du robot.

Le module de Navigation NAV centralise les informations sur la position du robot issues des modules ODO et MCL. Il permet de disposer de données de localisation de référence. Ce module ne correspond pas à une fonctionnalité identifiée dans la décomposition SADT, mais il doit être considéré dans la suite de l'analyse AMDEC puisqu'il est intégré dans l'architecture de contrôle.

L'analyse fonctionnelle du robot a donc permis de définir les différentes fonctionnalités nécessaires à la réalisation de la mission visée ainsi que les modules architecturaux correspondants.

La méthodologie de tolérance aux fautes proposée nous amène maintenant à analyser les modes de défaillances de ces fonctionnalités puis à définir leurs sévérités vis à vis du fonctionnement du robot.

7.1.2 Identification des modes de défaillance

L'utilisation des diagrammes d'Ishikawa va permettre d'identifier les fautes (internes au module) ou les défaillances (externes au module) pouvant affecter le service rendu par chaque module de l'architecture. En présence d'un dysfonctionnement de ce dernier, il sera inutilisable ou les données qu'il génère ne seront plus cohérentes avec le fonctionnement attendu de l'architecture.

Dans la suite, cette analyse est déployée selon les axes d'études retenus et définis dans le paragraphe 4.3.3.2.a. Les figures construites précisent en légende les modes de défaillances, le type (Partielle, Complète) et la durée d'activation (transitoire, permanente) des défaillances pouvant être envisagées. Du fait de notre vision architecturale et modulaire, en présence de dysfonctionnements engendrés par l'occurrence d'une faute externe à un module, ces diagrammes établissent soit un lien direct avec la faute « physique », soit avec les modules de l'architecture qui ont propagé la faute et qui sont directement connectés au module analysé. Dans le premier cas nous indiquerons le nom de la faute physique, dans le second nous spécifierons le type de défaillance ((C) pour Complète et (P) pour Partielle) et le nom du module propageant la faute [1]. L'enchainement de la lecture des différents diagrammes d'Ishikawa permettra s'il y a lieu de remonter, de proche en proche, vers la faute originelle.

On peut remarquer que puisque nous nous focalisons sur l'aspect logiciel chacun des modules de l'architecture est un processus système. Il peut donc, suite à un dysfonctionnement interne (erreur de blocage), rester bloqué dans un état non-contrôlable. Ainsi chaque module analysé peut être défaillant sur l'occurrence d'un tel évènement (**Module HS**) qui sera présent suivant l'axe architecture de contrôle du diagramme Ishikawa.

De la même façon, l'exécution d'un module peut s'avérer trop longue par rapport au temps imparti et engendrer une erreur temps réelle. Celle-ci peut être due soit à une erreur de blocage, soit à un algorithme ayant une durée d'exécution trop importante. Cela induira à nouveau la défaillance **erreur TR** au niveau de l'axe architecture de contrôle du diagramme Ishikawa de chacun des modules.

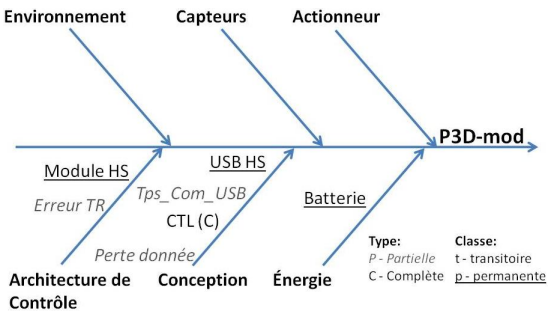
Nous présentons maintenant l'analyse des autres défaillances pouvant affecter les modules associés au mode de fonctionnement suivi de chemin autonome SMZ.

7.1.2.1 Module de communication avec le Pioneer-3DX : P3D

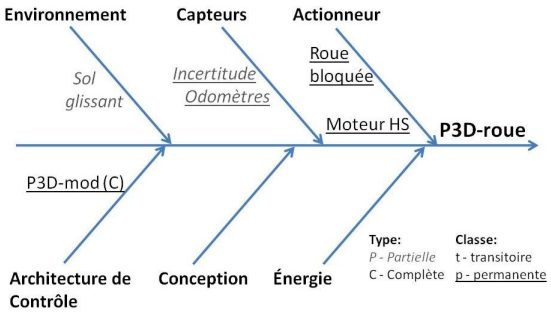
Les diagrammes d'Ishikawa associés à chacune des fonctionnalités du module P3D sont présentés dans la figure 7.2.

1. La durée d'activation de la défaillance n'est pas ici significative

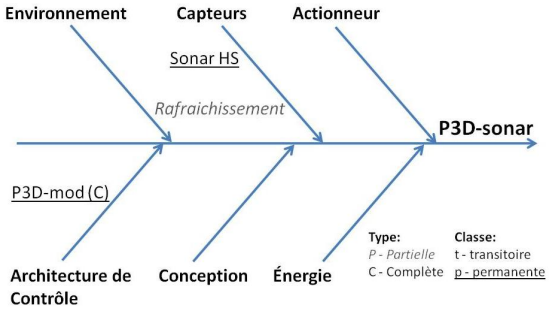
– La fonction P3D-mod (figure 7.2a) est sensible à la défaillance complète de la communication avec le micro-contrôleur **USB HS** et à la perte d'énergie fournie par la **Batterie** du robot. La non-présence de la commande fournie par le module CTL perturbe complètement P3D-mod, même si un système de watch-dog interne permet de stopper le robot lorsqu'aucune commande n'est présente en entrée. La gestion de la liaison USB avec le robot peut engendrer une faute temps réel liée à un retard temporel **Tps_Com_USB** causant une défaillance partielle de la fonction. De même si le paquet transmis du micro-contrôleur contient des données corrompues une **perte de donnée** est détectée. Si la fonctionnalité P3D-mod du module P3D est défaillante, alors les autres fonctionnalités du module P3D (P3D-roue et P3D-sonar) le seront aussi nécessairement.
– La fonction P3D-roue (figure 7.2b) est complètement sensible à une **roue bloquée** ou à la mise hors service d'un des moteurs (**moteur HS**). Les capteurs encodeurs créent une défaillance partielle due à l'(**Incertitudes odomètres**). De même la perte de motricité des roue sur un **sol glissant** peut perturber transitoirement et partiellement le robot.
– La fonction P3D-sonar (figure 7.2c) est complètement défaillante si les capteurs ultrasoniques sont hors-service (**Sonar HS**).



(a) Fonctionnalité P3D-mod



(b) Fonctionnalité P3D-roue



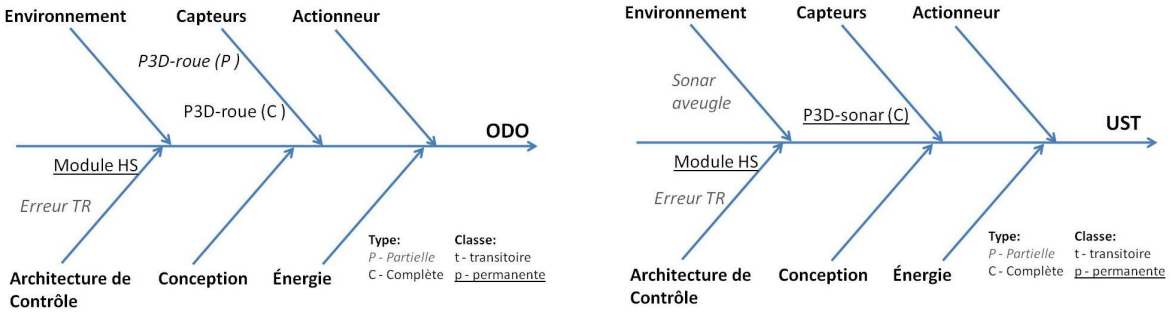
(c) Fonctionnalité P3D-sonar

Figure 7.2 – Diagrammes d'Ishikawa des fonctionnalités du module P3D

7.1.2.2 Modules de perception : ODO et UST

Le module ODO Le module ODO 7.3a est sensible aux défaillances complètes de la fonction **P3D-roue**. Il est aussi influencé par les défaillances partielles de la fonction P3D-roue.

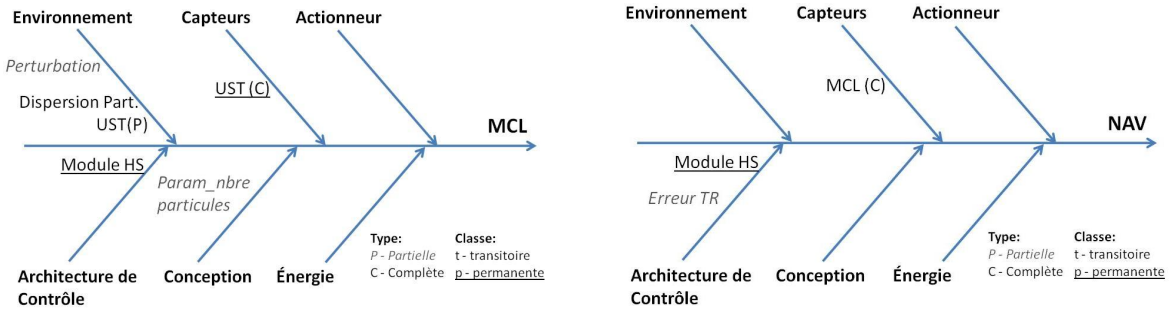
Le module UST Le module UST 7.3b est lui sensible aux défaillances des capteurs proximétriques (sonars) au travers de P3D-sonar. Les capteurs utrasoniques ont une portée de lecture faible (1,50m à la fréquence de travail) et ne permettent donc pas de retranscrire l'environnement si le robot est trop éloigné des murs. Dans ce cas, les (**sonar aveugle**) induisent donc une défaillance partielle du module UST.



(a) Module ODO (b) Module UST

Figure 7.3 – Diagrammes d'Ishikawa des modules ODO et UST

7.1.2.3 Modules de localisation : MCL et NAV



(a) Module MCL (b) Module NAV

Figure 7.4 – Diagrammes d'Ishikawa des modules MCL et NAV

Le module MCL (figure 7.4a) est complètement défaillant si les données provenant des capteurs sonar sont erronées :

- Cette défaillance est permanente si le module **UST(C)** est complètement défaillant.
- Elle est transitoire et entraine une dispersion des particules si la défaillance du module **UST(P)** est partielle. Les capteurs ne sont pas à même de représenter l'environnement

donc l'algorithme ne peut pas évaluer les particules en fonction de celui-ci pour définir la position réelle du robot.

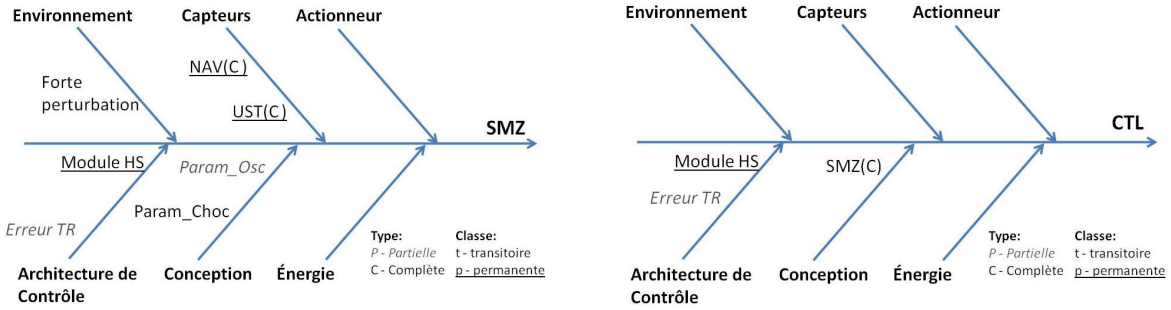
Une perturbation dans l'environnement, comme par exemple un grand nombre d'obstacles dynamiques non répertoriés, peut induire une **perturbation** de la localisation du robot induisant une défaillance partielle de MCL. Une défaillance partielle due à une mauvaise paramétrisation du nombre de particules (**Param_nbre particules**) entrainera un temps de calcul excessif et donc un retard temporel.

Le module NAV (figure 7.4b) est évidement sensible aux fautes des modules capteurs. Une défaillance complète et transitoire du module NAV est détectée si le module MCL est complètement fautif.

7.1.2.4 Modules de contrôle du robot : SMZ et CTL

Le module SMZ (figure 7.5a) est défaillant si les capteurs proximétriques **UST(C)** le sont. Une défaillance complète du module est détectée si une mauvaise paramétrisation de l'algorithme ne permet pas d'éviter un choc avec un obstacle **Param_choc**. Une **forte perturbation** de l'environnement conduisant le robot à retourner sur ses pas conduira a définir une défaillance complète du module. De même, une paramétrisation inadaptée de l'algorithme de suivi de chemin peut être à l'origine d'oscillations **Param_Osc** incontrôlées de la trajectoire du robot et engendre donc une défaillance partielle du module SMZ.

Le module CTL (figure 7.5b) est défaillant complètement et transitoirement si le module de SMZ ne peut lui fournir les données nécessaires au calcul de la loi de commande.



(a) Module SMZ

(b) Module CTL

Figure 7.5 – Diagrammes d'Ishikawa des modules SMZ et CTL

Les diagrammes Ishikawa réalisés ont permis d'identifier l'ensemble des défauts pertinents auxquels sont sensibles les différents modules de l'architecture. L'analyse de sévérité sera conduite globalement dans le tableau de synthèse qui va être maintenant présenté.

7.1.3 Analyse de sévérité et tableau récapitulatif de l'AMDEC

Le résultat de la méthodologie AMDEC est présenté dans le tableau 7.1. Pour chacune des fonctionnalités des modules du mode de fonctionnement analysée (colonne Fonction), nous y retrouvons les fautes pertinentes (colonne fautes) à observer dans le fonctionnement de l'architecture et que nous venons d'identifier. La colonne *Modules liés fonctionnellement* fait apparaitre les liens inter-module de propagation des défaillances ayant des conséquences sur la qualité de service d'un module considéré. La colonne *principe d'observation* propose une technique permettant de diagnostiquer un dysfonctionnement. La colonne *Informations Recouvrement* suggère une action à mettre en œuvre lors du dysfonctionnement avéré d'un module. Enfin, le niveau de sévérité du mode de défaillance est précisé dans la colonne *sévérité*. Aux niveaux Faible, Moyen, Grave et Fatale définis dans 4.3.3.3 nous ajoutons aussi dans ce tableau les fautes que nous avons ignorées en raison de leur impact négligeable sur le fonctionnement du robot.

Nous allons maintenant illustrer sur un exemple, le processus d'évaluation du niveau de sévérité des modes de défaillances d'une fonction d'un module. Celui-ci doit d'une part prendre en compte globalement l'influence que ce dysfonctionnement peut induire dans l'exécution de la boucle de contrôle exécutée pour le mode de fonctionnement considéré. Il doit aussi, d'autre part, prendre en considération les capacités de redondance logicielle et fonctionnelle pouvant être employées pour pallier au dysfonctionnement supposé. Cette analyse suppose donc de disposer d'une vue d'ensemble de tous les modes de fonctionnement qui peuvent être déployés. Bien évidemment, à ce point de l'étude, cette analyse ne peut être que statique et devra être complétée, lors du processus décisionnel réel de recouvrement, par la prise en compte dynamique des moyens opérationnels de recouvrement.

Par exemple, si l'on considère la fonction P3D-mod du module P3D, les sévérités choisies pour les différents modes de défaillances sont explicitées ci-dessous :

- *Partiel et transitoire* : Ce mode de défaillance est généré sur des dépassements temporels engendrés par le module ou la gestion de la communication USB. Ceux-ci étant très rares, ils sont donc **Ignorés**.
- *Complet et transitoire* : Ce mode de défaillance est induit par dysfonctionnement du module CTL. Il n'existe a priori pas de solution possible faisant appel à de la redondance logicielle. Nous qualifions donc la sévérité de mode de défaillance de **Grave** puisqu'alors une intervention Humaine distante est nécessaire pour poursuivre la mission.
- *Complet et permanent* : Ce mode de défaillance engendre un arrêt complet du robot soit parce que la communication est rompue, soit parce que le robot n'a plus d'énergie, soit en raison d'un dysfonctionnement du seul module de communication du robot. Le niveau de sévérité associé est donc **Fatal** car il n'existe aucune solution autre que de clore prématurément la mission et de faire appel à la maintenance.

MODULE	MODE DEFAILLANCES		CAUSES DES DEFAILLANCES			SEVERITE	ACTIONS	
	classe	Type	Fautes		Modules liés fonctionnellement		Principe d'observation	Informations Recouvrement
			Identificateur	Information				
P3D(mod)	Partielle	transitoire	P3D-mod1	Tps_Com_USB		ignorée	Watch dog variable	
			P3D-mod2	Perte donnée				
			P3D-mod3	Erreur TR			Watchdog activation	
	Complète	transitoire			CTL(C)	grave		
			P3D-mod4	Batterie			Information niveau de batterie robot	Arret
		permanent	P3D-mod5	Module HS		fatale	Watchdog activation	Arret
			P3D-mod6	USB HS			Watchdog communication	Arret
P3D(roue)	Partielle	transitoire	P3D-roue1	Sol glissant		ignorée	Perturbation ponctuelle non detectée	Utilisation préventive du MCL pour correction passive
		permanent	P3D-roue2	Incertitude odométres			Perturbation permanente	
	Complète	permanent			P3D(mod)(C)	fatale		Arret
			P3D-roue4	Roue bloqué			Banque filtre de kalman	Arret
			P3D-roue5	Moteur HS			Banque filtre de kalman	Arret
P3D(sonar)	Complète	transitoire	P3D-sonar1	Rafraichissement		moyen	Watchdog activation	MCL non utilisable
		permanent			P3D(mod)(C)			
			P3D-sonar2	Sonar HS			Watchdog - vivacité des valeurs	Aucun sonar
UST	Partielle	transitoire		Erreur TR		ignorée	Watchdog activation	
			UST1	Sonar aveugle			Limite d'un nombre de sonar actif	MCL non utilisable
	Complète	permanent	UST2		P3D(sonar)(C)	moyen		
			UST3	Module HS			Watchdog activation	Mode sans US
ODO	Partielle	transitoire			P3D(roue)(P)	faible		
			ODO1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent			P3D(roue)(C)			
			ODO2	Module HS		fatale	Watchdog activation	Arret
MCL	Partielle	transitoire	MCL1	Perturbation		faible	Différence entre position ODO et MCL	reconfig NAV avec odométrie
			MCL3	Param_nbre Particules			Watchdog activation	reparam nb de particules
	Complète	transitoire	MCL4	Dispersion particules	UST(P)	grave	Analyse Ecart type des particules	
		permanent			UST(C)			
			MCL5	Module HS			Watchdog activation	Mode sans MCL
NAV	Partielle	transitoire	NAV1	Erreur TR		ignorée	Watchdog activation	
	Complète	transitoire			MCL(C)	grave		
		permanent	NAV3	Module HS		fatale	Watchdog activation	Arret
SMZ	Partielle	transitoire	SMZ1	Erreur TR		ignorée	Watchdog activation	
			SMZ2	Param - Osc		faible	Observateur d'oscillation	
		transitoire	SMZ3	Param-choc		grave	Information bumper robot	Appel Opérateur
	Complète		SMZ4	Perturbation forte			Retour arrière	Appel Opérateur
		permanent			(NAV(C) - UST(C))	moyen		Changement d'algorithme de suivit de chemin
			SMZ5	Module HS			Watchdog activation	
CTL	Partielle	transitoire	CTL1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent			SMZ(C)			
			CTL2	Module HS		grave	Watchdog activation	

Table 7.1 – Tableau récapitulatif de l'AMDEC appliquée au suivi de chemin autonome SMZ

Nous venons de voir que la définition des niveaux de sévérité des différents modes de défaillances dépend de la connaissance de l'ensemble des modes de fonctionnement envisageables pour réaliser la mission. Nous allons donc balayer ceux que nous avons considérés dans le cadre de notre travail.

7.2 Les modes de fonctionnement pour le recouvrement

Ici nous allons présenter les différents modes de fonctionnement qui ont été développés pour le recouvrement lorsqu'une des défaillances identifiées d'un mode de fonctionnement est détectée.Ils sont présentées ici suivant le niveau d'autonomie auquel ils sont rattachés. Tout d'abord les modes autonomes sont décrits, puis ceux faisant appel à l'interaction Homme-robot sont abordés.

7.2.1 Les modes de fonctionnement autonomes de l'objectif suivi de chemin

Le mode de fonctionnement optimal est celui décrit dans la section 6.2.2.2, le suivi de chemin avec évitement d'obstacle SMZ et localisation de Monte-Carlo. Cependant en présence d'une défaillance, de façon à assurer une continuité de service ce sous-objectif peut être décliné en plusieurs modes de fonctionnement autonomes grâce à l'utilisation de la redondance logicielle.

7.2.1.1 Le Suivi de chemin avec évitement d'obstacle DVZ

Ce mode de fonctionnement possède les mêmes caractéristiques que le suivi de chemin SMZ mais en adoptant simplement une stratégie d'évitement d'obstacle différente.

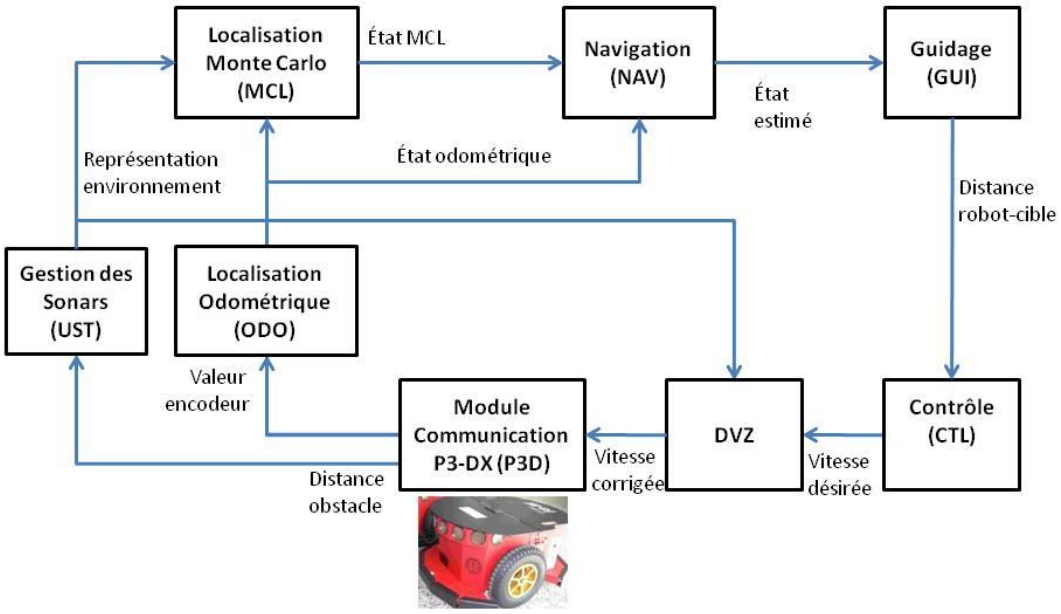


Figure 7.6 – Schéma de contrôle du suivi de chemin avec DVZ

On observe, dans la figure 7.6 que le module SMZ n'entre plus en jeu et qu'il est remplacé par le module de guidage GUI et par le module d'évitement d'obstacle réactif DVZ. La table 7.2 énumère l'ensemble des modules impliqués dans ces deux modes de fonctionnement.

Sous-objectif	Niveau Autonomie	Liste des modules actifs								
Suivi de chemin autonome SMZ	Autonome	P3D	SIM	UST	ODO	MCL	NAV	SMZ	CTL	
Suivi de chemin avec DVZ	Autonome	P3D	SIM	UST	ODO	MCL	NAV	GUI	CTL	DVZ

Table 7.2 – Modules du suivi de chemin DVZ

7.2.1.2 Le suivi de chemin SMZ sans MCL

Le suivi de chemin SMZ sans MCL est de toute évidence un mode dégradé du mode suivi de chemin SMZ. Puisque la fusion des capteurs n'est plus utilisable pour définir la position du robot. Dans ce cas, l'odométrie est donc le seul moyen de localisation disponible, ce qui induit une dérive de la localisation du robot au cours du temps. Le schéma de commande est donc celui du suivi de chemin SMZ (figure 6.7) où le module MCL a été supprimé. Le module NAV sera configuré pour définir la localisation odométrique comme prioritaire. Les modules impliqués dans ce mode de fonctionnement sont précisés dans le tableau 7.3.

Sous-objectif	Niveau Autonomie	Liste des modules actifs						
Suivi de chemin SMZ sans MCL	Autonome	P3D	SIM	UST	ODO	NAV	SMZ	CTL

Table 7.3 – Modules du suivi de chemin SMZ sans MCL

7.2.1.3 Le suivi de chemin DVZ sans MCL

Le suivi de chemin DVZ sans fusion des capteurs est l'équivalent du mode précédent où l'évitement d'obstacle est réalisé par la DVZ. Ainsi le module MCL disparait de la liste des modules impliqués 7.4.

Sous-objectif	Niveau Autonomie	Liste des modules actifs							
Suivi de chemin sans MCL avec DVZ	Autonome	P3D	SIM	UST	ODO	NAV	GUI	CTL	DVZ

Table 7.4 – Modules du suivi de chemin DVZ sans MCL

7.2.1.4 Le suivi de chemin sans utilisation des sonars

Le suivi de chemin sans utilisation des sonars est de toute évidence le mode le plus dégradé pour réalisé le suivi de chemin autonome. Le robot se déplace en effet en aveugle dans l'environnement sans avoir la possibilité d'utiliser ses sonars pour assurer ses déplacements et éviter les obstacles. Aucun des modules traitant des sonars (UST,MCL, DVZ et SMZ) n'est donc utilisés 7.5.

Sous-objectif	Niveau Autonomie	Liste des modules actifs				
Suivi de chemin sans sonar	Autonome	P3D	ODO	NAV	GUI	CTL

Table 7.5 – Modules du suivi de chemin sans information sonar

Lorsque la redondance logicielle ne peut plus être utilisée car il n'y a plus de solution alternative, le recours à une redondance cognitive distante peut être envisagée pour mener à bien la mission. On fait alors appel à des modes de fonctionnement non autonomes en intégrant l'Homme dans la boucle.

7.2.2 Les modes de fonctionnement non-autonomes

Plusieurs modes de fonctionnement impliquant une interaction Homme-robot ont été développés dans le but de tenter de répondre aux problèmes que le dispositif robotique n'a pu résoudre de lui-même (permettant ainsi d'achever la mission en cours). Deux niveaux d'autonomie interactifs ont été envisagées : la **téléprogrammation** et la **téléopération** que nous allons maintenant présenter.

7.2.2.1 Les modes de fonctionnement de téléprogrammation

La téléprogrammation permet à l'opérateur d'apporter s'il le peut une aide ponctuelle au robot. Le principal problème de ce type d'interaction est que l'opérateur a à priori une mauvaise connaissance du contexte d'évolution du robot. Pour intervenir de façon efficace il devra donc disposer d'informations pertinentes sur le fonctionnement antérieur du robot, son état courant et sur l'environnement dans lequel il se trouve actuellement.

La phase d'interaction homme robot en téléprogrammation présente donc dans un premier temps une phase de prise de connaissance pour l'opérateur. Celle-ci peut être faite en analysant le comportement antérieur du robot et/ou en prenant connaissance l'environnement réel du robot avec l'aides des moyens de téléopération et notamment d'une caméra. Dans un second temps, l'opérateur pourra définir l'action corrective qu'il souhaite mettre en place pour pallier les dysfonctionnements que le robot n'a pu surmonter.

Dans notre cadre expérimental, même si la liste des actions correctives en téléprogrammation reste évidement ouverte, les solutions adoptées sont les suivantes :

Localisation du robot - L'opérateur re-localise le robot et lui fournit sa nouvelle position sur la carte.

Définition d'un nouveau chemin - L'opérateur modifie le chemin que le robot doit suivre pour par exemple éviter un obstacle non prévu et lui communique les nouveaux points de passage.

Diagnostic distant de modules fautifs - L'opérateur à partir des informations dont il dispose est arrivé à déterminer la ou les origines du dysfonctionnement. L'état des modules est mis à jour en conséquence dans la base de donnée.

Ces deux alternatives conduisent à définir un seul mode de fonctionnement téléprogrammation qui comme nous le voyons dans le tableau ne fait appel qu'à trois modules. Le module UDP qui en fonction de l'ordre de l'opérateur viendra soit mémoriser le chemin reçu, soit communiquer directement à NAV la nouvelle position du robot. Le module NAV mémorise la position courante du robot. Ces opérations étant effectuées robot à l'arrêt, le module P3D n'est présent que pour maintenir la liaison avec le micro-contrôleur.

Sous-objectif	Niveau autonomie	Liste des modules actifs		
Téléprogrammation	Téléprogrammé	NAV	UDP	P3D

Table 7.6 – Module des modes de fonctionnement téléprogrammé

Après une opération de téléprogrammation l'opérateur devra choisir le nouveau mode de fonctionnement (autonome ou non) qui lui semble le plus judicieux pour que le robot achève sa mission avec les nouveaux paramètres fournit.

7.2.2.2 Les modes de fonctionnement de téléopération

Deux modes de fonctionnement en téléopération ont été développés (figure 7.7) :

- Dans le premier l'évitement d'obstacle réactif par DVZ est maintenu tandis que l'opérateur définit le déplacement du robot. Le module DVZ utilise les données sonars fournies par le module UST pour valider le déplacement souhaitée par l'opérateur. On peut remarquer que dans ce cas de figure l'algorithme d'évitement d'obstacle SMZ ne peut être utilisé car il nécessite de connaitre à priori le chemin que doit suivre le robot.
- Dans le second seules les directives de l'opérateur sont appliquées au robot.

Pour chaque mode de fonctionnement de téléopération deux stratégies sont possibles pour l'opérateur. L'une, en mode articulaire, commande directement la vitesse de rotation de chacune des roues du robot à partir de deux joystick. L'autre, en mode cartésien, n'utilise qu'un seul

Sous-objectif	Niveau Autonomie	Liste des modules actifs					
Téléopération avec évitement d'obstacle	Téléopération	P3D	UST	ODO	NAV	UDP	DVZ
Téléopération sans DVZ	Téléopération	P3D	ODO	NAV	UDP		

Table 7.7 – Module des modes de fonctionnement téléopéré

joystick qui défini le cap et la vitesse du robot qui sont convertis par le module UDP pour commander les roues.

L'ensemble des modes de fonctionnement pour le recouvrement viennent d'être définis. L'annexe B présente les tableaux d'analyse AMDEC récapitulatifs de tous ces modes de fonctionnement où l'on trouvera pour chacun des modules utilisés les sévérités des différents modes de défaillances. Il est important de remarquer que les fautes rendant un module intrinsèquement fautif se retrouvent dans tous les modes de fonctionnement où ce module est impliqué. En revanche, les défaillances propagées sont dépendantes du mode de fonctionnement considéré et donc des modules impliqués.

7.3 Conclusion

Cette première phase de notre méthodologie de développement d'une architecture de contrôle tolérante aux fautes, a permis de définir l'ensemble des modes de fonctionnement que nous avons retenu pour mener à bien la mission de livraison de courrier au sein du laboratoire. Au total douze modules ont été décrits et développés. Ils ont été projetés à travers trois niveaux d'autonomie pour définir huit modes de fonctionnement différents plus ou moins performants. La démarche AMDEC qui a été déployée sur chacun d'eux a permis de mettre en évidence, pour chacun des modules impliqué, leurs modes de défaillance ainsi que les niveaux de sévérité associés.

Avec l'aide des informations rassemblées nous allons maintenant pouvoir aborder le développement proprement dit de l'architecture tolérante aux fautes pour laquelle nous mettrons en place, au sein de COTAMA, les mécanismes nécessaires à la détection d'une défaillance, au diagnostic de la faute et au processus de recouvrement.

Chapitre 8

Tolérance aux fautes : Mise en œuvre Architecturale

La phase préliminaire de notre méthodologie a permis de mettre en avant, d'étudier, et d'analyser l'impact des dysfonctionnement potentiels de notre système robotique sur son comportement. La seconde phase que nous allons maintenant détailler, va permettre de détecter et diagnostiquer puis de réagir à l'occurrence de ces défauts lors de la réalisation de la mission pour construire une architecture tolérante aux fautes.

L'architecture logicielle COTAMA dispose des mécanismes fondamentaux nécessaires au déploiement de la tolérance aux fautes. D'une part, la modularité déployée au niveau exécutif lui permet d'ors et déjà de supporter des processus de détection et de diagnostic « au fil de l'eau », sans pour autant les intégrer dans une logique structurante. D'autre part, ses capacités décisionnelles, déployées sous la forme de machines à états, autorisent la mise en œuvre de la réaction nécessaire. Cependant celle-ci reste pour l'instant noyée au milieu des moyens permettant de gérer l'évolution de la mission proprement dite.

Pour pallier ces insuffisances, nous proposons de modifier la composition de l'architecture COTAMA pour la structurer au regard des moyens de détection, de diagnostic et de réaction nécessaires à la mise en œuvre d'une architecture tolérante aux fautes.

Ces modifications, que nous détaillerons plus en avant tout au long de ce chapitre, se traduisent au travers de la figure 8.1 par l'introduction d'un ensemble de nouveaux modules (grisés) :

- La phase de détection de défaillances sera assurée, au niveau exécutif, par un ensemble de modules dédiés, les **Modules d'Observation** qui permettront de générer les informations qui seront utilisées lors de la phase de diagnostic.
- La phase de diagnostic est réalisée, toujours au niveau exécutif, par le **Module de Diagnostic**. Celui-ci centralise les informations collectées par les Modules d'Observation et détermine l'origine des défaillances.
- Sur sollicitation du Module de Diagnostic, la phase de recouvrement est déployée au ni-

veau décisionnel, à travers plusieurs superviseurs. Tout d'abord le **Superviseur Contextuel** détermine, en fonction du contexte actuel de la mission (modules opérationnels) et de la sévérité des défaillances détectées, le type de réaction à mettre en œuvre. Puis, en fonction de la gravité de la situation précédemment identifiée, plusieurs superviseurs (Global, Local et Adaptation) peuvent être activés pour choisir le recouvrement à mettre en œuvre. Cette nouvelle décomposition de l'architecture met en évidence la présence de deux circuits évènementiels concomitants. Le premier (partie gauche de la figure) correspond aux évènements directement liés à la gestion de la mission. Le second (partie droite) correspond au processus de tolérance aux fautes mis en place.

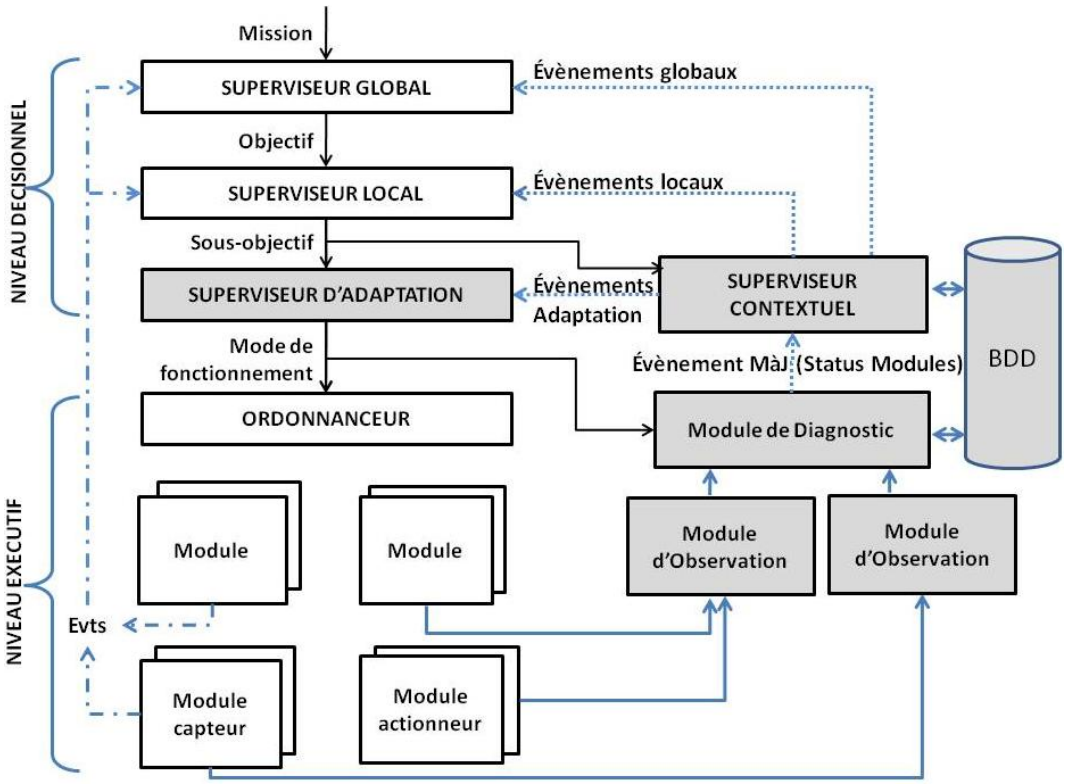


Figure 8.1 – Architecture COTAMA après modification

Nous allons maintenant aborder le déploiement du processus de détection au travers de la mise en place d'un ensemble de Modules d'Observation.

8.1 Phase de détection : Les Modules d'Observation

Le concept de module d'Observation a déjà été employé dans plusieurs travaux du domaine [Brandstötter et al., 2007, Steinbauer and Wotawa, 2005, Grosclaude, 2004, Borrelly et al., 1998]. Cependant, dans notre étude, nous nous inscrivons dans le cadre d'une démarche globale et intégrée. Celle-ci s'appuie sur le résultat de l'AMDEC qui a permis d'identifier l'ensemble des défaillances pertinentes à scruter et qui sont rassemblées dans la colonne

(Causes des défaillances/fautes/informations) des tableaux de synthèse 7.1. Il faut donc trouver, pour chacune d'entre elles, le mécanisme permettant sa détection qui sera ensuite déployé au sein du Module d'Observation correspondant. Chaque module d'observation monitore un ensemble de défaillances potentielles et permettra donc de diagnostiquer l'occurrence des fautes identifiées par la démarche AMDEC.

Dans la suite de l'exposé nous allons balayer les différents Modules d'Observation nécessaires. Classés selon la typologie structurelle définie dans la section 4.4.1, nous préciserons pour chacun d'eux la stratégie de détection mise en œuvre ainsi que ses entrées/sorties.

8.1.1 Les modules d'observation de données

Les modules d'observation de données se basent sur les données produites par un seul module pour évaluer le comportement d'un algorithme, du robot ou de ses capteurs.

8.1.1.1 Module d'observation des actionneurs : ACT

Ce module (table 8.1a) scrute les informations odométriques fournies périodiquement par le module P3D (**etat**) qui fournie les valeurs des vitesses et d'accélérations des deux roues. Il vérifie alors en faisant appel à du contrôle de vraisemblance que ces données respectent les plages de valeurs imposées par le constructeur. En cas de défaillance, les indicateurs correspondants (**vit_roueG** et **vit_roueD**) et (**acc_roueG** et **acc_roueD**) du tableau 8.1b sont mis à 1.

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiEtatRobot	etat
Sorties	0	OErrorACT	actionneur

(a) Les ports de l'observateur ACT

OErrorACT	
Type	Champ
int	vit_roueG
int	vit_roueD
int	acc_roueG
int	acc_roueD

(b) Structure OErrorACT

Table 8.1 – Module d'observation ACT

8.1.1.2 Module d'observation des capteurs : SNS

Les valeurs des capteurs proximétriques (**sonar**) du robot, fournies par le module P3D, sont évaluées par ce module dont la structure est décrit table 8.2a.

- Les 16 capteurs entourant le robot sont observés tout d'abord pour déterminer si la périodicité de mise à jour des informations est respectée. Dans le cas contraire, l'indicateur **actualisation** (table 8.2b) est mis à 1.
- De plus, le nombre de capteurs produisant un relevé exploitable (capteur non aveugle) est calculé en permanence. Cette information est nécessaire pour fiabiliser la fusion de

données (localisation de Monte Carlo). L'indicateur **USaveugle** détecte une dégradation de la représentation de l'environnement par les sonars lorsque plus de la moitié d'entre eux sont aveugles.

– Enfin un algorithme s'assure que les valeurs fournies par les capteurs varient effectivement sur une plage temporelle assez large. Dans le cas contraire, l'indicateur **UShors_service** est mis à un.

Les données **actualisation** et **UShors_service** sont évaluées à l'aide d'une stratégie de type contrôle temporel (watchdog). L'information **USaveugle** utilise un simple seuillage pour détecter la défaillance.

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiDonneeUltraSon	sonar
Sorties	0	OErrorSNS	capteur

(a) Les ports de l'observateur SNS

OErrorSNS	
Type	Champ
int	USaveugle
int	UShors_service
int	actualisation

(b) Structure OErrorSNS

Table 8.2 – Module d'observation SNS

8.1.1.3 Module d'observation du suivi de chemin : OSC

Ce module (table 8.3) scrute l'évolution du module de suivi de chemin SMZ. Il fait appel à la variable **rabbit** produite par SMZ pour évaluer la qualité du suivi de chemin par rapport aux contraintes en s'assurant que la progression du robot continue et qu'elle ne présente aucune oscillation.

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiRabbit	rabbit
Sorties	0	OErrorOSC	errorOSC

(a) Les ports de l'observateur OSC

OErrorOSC	
Type	Champ
int	oscillation
int	retour_chemin

(b) Structure OErrorOSC

Table 8.3 – Module d'observation OSC

Dans la table 8.3b les indicateurs de qualité du suivi de chemin sont construits de la façon suivante :

– L'évolution du rabbit sur le chemin permet de déterminer si le robot se déplace ou non dans le bon sens. L'indicateur **retour_chemin** est donc activé si le robot parcours le chemin dans le mauvais sens sur une distance dépassant un seuil fixé empiriquement (1 mètre).

– La différence $TmoinsD = \Theta - \Theta_c$ représente l'angle entre l'orientation θ du robot et θ_c du

rabbit (voir paragraphe 6.2.2.2.f) permet d'identifier la présence d'oscillations anormales. Cette donnée est une des clefs du fonctionnement de l'algorithme de suivi de chemin qui cherche à minimiser cet écart. Ce dernier présente un signe constant pour un fonctionnement correct. Le changement fréquent de son signe met en évidence, soit des problèmes de réglages de paramètres internes à l'algorithme de suivi de chemin, soit des problèmes dans l'actualisation de l'état du robot [Lapierre et al., 2007]. Ainsi dans l'algorithme de détection, on décompte le nombre de changement de signe observés durant un horizon temporel glissant donné (de l'ordre 1 minute). L'indicateur **oscillation** est positionné à 1 lorsque ce décompte dépasse 30 oscillations (seuil définis empiriquement).

Ces deux défaillances ne permettent pas de détecter directement des fautes impactant des modules du mode de fonctionnement courant. En revanche elles pointent des comportement anormaux du système pouvant avoir plusieurs origines : l'incertitude des odomètres, le sol glissant ainsi que des perturbations de l'environnement.

8.1.1.4 Module d'observation du réseau Wifi : OWF

Le module OWF (table 8.4) scrute les informations fournies en permanence par le système linux (fichier wireless) pour analyser le comportement du réseau Wifi (charge du réseau, force du signal et niveau de bruit). Ce module a été réalisé lors d'un projet tuteuré par deux élèves d'IUT [Saraiva and Crocquefer, 2010]. Des seuils de qualité sont définis empiriquement pour pratiquer du contrôle de vraisemblance permettant d'identifier une mauvaise **Qualité** de réseau ou une perte totale de **connexion**.

	Port	Type de la donnée	Nom de la donnée
Entrées			
Sorties	0	OerrorOWF	OerrorOWF

(a) Les ports de l'observateur OWF

OErrorOWF	
Type	Champ
int	Qualite
int	connexion

(b) Structure OErrorOWF

Table 8.4 – Module d'observation OWF

8.1.2 Les modules d'observation de module

Cette classe structurelle de Module d'Observation permet de détecter un dysfonctionnement en analysant les données présentes en entrée et en sortie d'un même module. L'analyse AMDEC ne permet de dégager qu'un seul composant nécessitant ce type d'observation, le module de localisation MCL.

8.1.2.1 Module d'Observation de la localisation : LOC

Pour évaluer le comportement de l'algorithme de localisation de Monte Carlo, le module d'observation LOC (figure 8.5) a été développé. Il évalue l'écart entre la localisation odométrique **LocODO** (entrée de MCL) et celle **LocMCL** obtenue par le filtre particulaire (sortie de MCL). La table 8.5b décrit les deux indicateurs construits, la différence de localisation **ecart_odo_mcl** (en mètres) et le comportement temporel **dot_ecart_odo_mcl** de celle-ci évaluée en mètre par seconde. Des seuils empiriques ont été définis pour détecter un dysfonctionnement. Dans le premier cas l'indicateur **ecart_odo_mcl** est positionné au delà d'un seuil de 1 mètre. Dans le second cas, sa dérive **dot_ecart_odo_mcl** est mise à 1 lorsqu'elle dépasse 0,2 m/s.

La **Dispersion_particules** dans le filtre particulaire est détectée à l'aide d'un seuil empirique sur l'**Ecart_type** mis à jour dans l'algorithme de localisation de Monte-Carlo. Cette dispersion des particules est provoquée par une mauvaise représentation de l'environnement par les capteurs sonars. La détection au sein du module SNS de cette faible représentation par **USaveugle** permet d'anticiper la dispersion.

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiMCL	locMCL
	1	TArchiEtatRobot	locODO
Sorties	0	OErroLOC	erreur_loca

(a) Les ports de l'observateur LOC

OErrorLOC	
Type	Champ
int	ecart_odo_mcl
int	dot_ecart_odo_mcl
int	Dispersion_particules

(b) Structure OErrorLOC

Table 8.5 – Module d'observation LOC

8.1.3 Les modules d'observation multi-modules

La dernière classe de Module d'Observation fait appel à des informations issues de plusieurs modules pour détecter une défaillance au sein du système robotique.

8.1.3.1 Module d'observation du comportement du robot : BKF

	Port	Type de la donnée	Nom de la donnée
Entrées	0	TArchiVitesseRobot	commande
	1	TArchiEtatRobot	etat
Sorties	0	OErrorBKF	errorBKF

(a) Les ports de l'observateur BKF

OErrorBKF	
Type	Champ
int	errorRD
int	errorRG

(b) Structure OErrorBKF

Table 8.6 – Module d'observation BKF

Le module BKF (table 8.6) implémente un algorithme de diagnostic multi-modèles qualitatif (banque de filtres de Kalman [Roumeliotis et al., 1998]) monitorant le comportement du système physique du robot. Il simule à partir des **commandes** envoyées aux roues la banque des modèles du robot pour déterminer dans chacun des cas la localisation obtenue. Il compare alors ces résultats avec les informations fournies par la localisation courante (**etat**). Compte tenu de cet ensemble d'informations, il est alors possible de définir le modèle le plus à même de représenter le comportement actuel du système physique du robot.

Le module BKF permet de discriminer les fonctionnements suivants :

– nominal
– roue gauche bloquée
– roue droite bloquée

Il permet donc de détecter les défaillances de blocage des roues : **errorRD** et **errorRG**.

8.1.3.2 Module d'observation de l'exécution temps réel : RTC

Le module d'observation RTC (figure 8.7b) permet de détecter l'occurrence d'erreurs temps réel au niveau de l'ensemble des modules de l'architecture. Ce module récupère les informations de détection provenant de l'ordonnanceur pour signaler les fautes temporelles des modules exécutés.

En raison de sa nature, bien qu'il n'existe pas de lien explicite entre RTC et les modules exécutés, mais puisqu'il centralise des informations qui leur sont liées, nous avons choisi de le classer parmi les modules d'observation multi-modules.

	Port	Type de la donnée	Nom de la donnée
Paramètres	0	int	module_retardataire
Sorties	0	OerrorRTC	errorRTC

(a) Les ports de l'observateur RTC

OErrorRTC	
Type	Champ
int	retard P3D
int	P3D HS
int	retard SMZ
int	SMZ HS
...	...
int	retard SSObj

(b) Structure OErrorRTC

Table 8.7 – Module d'observation RTC

Ce module construit, pour chaque module, deux types d'indicateurs de défaillance temporelle : d'une part, les retards fréquents d'un module sont identifiés par **retard MOD** et d'autre part un module hors service est marqué **MOD HS**. A ces indicateurs s'ajoute une information plus globale qui détecte des retards fréquents au niveau de la boucle de contrôle robotique c'est à dire du sous-objectif **retard SSOBJ**.

Les seuils de détection retenus sont les suivants :

– un module (**retard MOD**) ou une boucle de contrôle (**retard SSOBJ**) est déclaré en retard s'il l'est plus de 4 fois sur un horizon glissant de 10 exécutions.
– un module est déclaré hors service (**MOD HS**) s'il ne renvoie aucun accusé de réception de fin d'exécution.

8.1.4 Les observations dans les modules de contrôle

Certains modules de contrôle de l'architecture intègrent directement dans leur fonctionnement des indicateurs permettant de détecter des situations ou des comportements anormaux pour permettre d'y répondre. Un exemple récurrent en robotique est la vérification que les angles définis en radian respectent bien les bornes mathématiques $[-\pi, \pi]$ ou$[0, 2\pi]$.

8.1.4.1 Le module P3D

Le module P3D qui gère la liaison série avec le robot fourni, en plus des variables présentées dans le paragraphe 6.2.2.2.a, un ensemble de données (table 8.8) associées à la gestion de cette connexion , à la détection de collision et au niveau d'énergie.

OErrorP3D	
Type	Champ
int	no_data
int	erreur_usb
int	usb_HS
int	bumpers
int	batterie

Table 8.8 – Résidu fourni par les algorithmes d'observation au sein du module P3D

– Lorsque l'indicateur **no_data** est actif le module P3D ne reçoit aucune commande à appliquer sur les roues, ce qui caractérise un dysfonctionnement notable de l'algorithme de contrôle.
– L'indicateur **erreur_usb** est mis à 1 lorsque les informations adressées par le micro contrôleur sont corrompues (test de parité).
– Si aucun paquet SIP n'est reçu par P3D, l'indicateur **usb_HS** est aussi activé.
– Sur un choc détecté par les **bumpers** du robot l'indicateur correspondant est levé.
– Un niveau d'énergie trop faible dans la batterie (seuil sur la tension) place l'indicateur **batterie** à 1.

8.1.5 Conclusion

Cette partie vient de décrire l'ensemble des Modules d'Observation construits pour détecter les défaillances mises en évidence à l'issue de l'analyse AMDEC. Ils permettent de réifier des mécanismes de programmation, souvent utilisés au sein des architectures, mais noyés dans le code des algorithmes de contrôle. Beaucoup d'entre eux s'appuient sur l'expérience empirique de l'expérimentateur pour définir des seuils de réaction adaptés. Quelques uns seulement utilisent des mécanismes de type contrôle de vraisemblance, de commande ou temporel. Enfin, certains font appel à des approches plus élaborées (banque de filtres de Kalman) pour détecter un dysfonctionnement.

Ces Modules d'Observation forment le socle du mécanisme de diagnostic, qui constitue la seconde phase du processus de tolérance aux fautes, que nous allons maintenant présenter.

8.2 Phase de diagnostic : Le Module de Diagnostic

Le but du diagnostic des fautes est de déterminer à un instant donné, quels sont, parmi l'ensemble des modules touchés par un dysfonctionnement de l'architecture, ceux qui sont fautifs ou défaillants. Nous distinguons les modules fautifs qui présentent un dysfonctionnement interne, des modules défaillants qui sont intrinsèquement sains mais dont le comportement est perturbé par un module autre fautif.

Le processus de diagnostic est pris en charge, au niveau exécutif de l'architecture, par un module dédié, le module de Diagnostic (figure 8.3) qui centralise en permanence toutes les informations d'observation pouvant être collectées. Les principes de son fonctionnement vont maintenant être présentés.

8.2.1 De l'observation au diagnostic : la matrice des résidus

Rappelons que la notion de résidu reflète la cohérence de données mesurées vis-à-vis d'un modèle comportemental d'un système. Ainsi chacun des Modules d'Observation détecte des déviations comportementales pouvant être assimilées à un ensemble de résidus. Les informations générées par les Modules d'Observation vont alors permettre de construire un vecteur de résidus sur lequel il sera possible de s'appuyer pour identifier les modules réellement fautifs.

Lors de l'analyse AMDEC, nous avons identifié les fautes perturbant chacun des modules (colonne causes des défaillances / fautes de la table AMDEC 7.1) ainsi que, pour chaque mode de fonctionnement, les défaillances propagées de module en module (colonnes causes des défaillances / modules liées fonctionnellement de cette même table). La table des résidus est établie en utilisant les fautes identifiées et les données d'observation collectées.

La table 8.9 présente la matrice obtenue pour le mode de fonctionnement en suivi de chemin autonome SMZ qui a été détaillé section 6.2.2.2. Cette table n'inclut pas les résidus liés aux dysfonctionnements potentiels des modules d'observation.

module		P3D												UST			ODO		MCL					NAV		SMZ				CTL	
fonction		P3D						sonar		roue																					
défaillance		trans			comp			comp		trans		comp		trans		comp	trans	comp	trans				comp	trans	comp	trans			comp	trans	comp
résidu		perte donnée	Erreur donnée	erreur TR	USB HS	Batterie	Module Hs	rafraichissement	Sonar HS	Incertitude odomètres	sol glissant	roue bloqué	moteur HS	erreur TR	Sonar Aveugle	Module HS	erreur TR	Module HS	erreur TR	Dispersion particules	Perturbation	Param_nbre Particules	Module HS	erreur TR	Module HS	Param - Osc	Param-choc	erreur TR	Module HS	erreur TR	Module HS
SNS	USaveugle														1																
	UShors_service								1																						
	actualisation							1																							
BKF	roue droite bloquée											1	1																		
	roue gauche bloquée											1	1																		
ACT	vit roue G											1	1																		
	vit roue D											1	1																		
	acc roue G											1	1																		
	acc roue D											1	1																		
P3D	batterie HS					1																									
	bumper																										1				
	no_data	1																													
	erreur_usb		1																												
	usb_hs				1																										
OSC	oscillation																									1					
	retour en arrière									1	1										1										
LOC	ecart localisation									1	1										1										
	dérivée localisation										1										1										
	dispertion particules																			1											
RTC	retard P3D			1																											
	P3D						1																								
	retard UST													1																	
	UST															1															
	retard ODO																1														
	ODO																	1													
	retard MCL																		1			1									
	MCL																						1								
	retard NAV																							1							
	NAV																								1						
	retard SMZ																											1			
	SMZ																												1		
	retard CTL																													1	
	CTL																														1

Table 8.9 – Matrice des résidus pour le suivi de chemin nominal

On retrouve en ligne l'ensemble des résidus, au nombre de 32, regroupés par module d'observation. En colonne, on balaye l'ensemble des fautes identifiées, au nombre de 29, par la démarche AMDEC et regroupées par module de contrôle de l'architecture. On peut remarquer que cette matrice de résidus est fortement localisante puisque, en présence d'une faute unique, 24 résidus sur 29 permettent directement de localiser les fautes. En présence de défaillances multiples, les relations de précédence spécifiées dans l'analyse AMDEC permettront de remonter aux fautes originelles.

8.2.2 Mise en œuvre du processus de diagnostic

Le processus déployé au sein du **Module de Diagnostic** a pour objectif la mise à jour d'une structure de données permettant de connaitre l'état courant de chacun des modules pouvant être utilisé par l'architecture. L'état d'un module ou d'une fonction est caractérisé par les informations suivantes :

- Nom de la fonctionnalité du module
- Sa disponibilité : Opérationnel ou Non opérationnel
- Son mode de défaillance actuel : Sain, Partiel ou Complet
- La durée d'activation du mode de défaillance : Permanente ou Transitoire.
- La propriété de réversibilité : Un module est réversible si la défaillance qui l'affecte est Transitoire.

Le processus de diagnostic, présenté dans la figure 8.2, peut être décomposé en 3 principales étapes successives :

1. **Construction du vecteur des résidus :** L'ensemble des informations issues des modules d'observation sont collectées pour construire le vecteur des résidus représentatif de l'état courant de l'architecture.

2. **Diagnostic apparition/disparition fautes :** Lors de l'apparition d'une défaillance (résidu au niveau d'un module actif), cette étape utilise la matrice des résidus correspondante au mode de fonctionnement courant, préalablement mémorisée dans la base de données, pour identifier la ou les fautes à l'origine des dysfonctionnements détectés. Le niveau de sévérité de cette défaillance est obtenu grâce à la table AMDEC correspondante. Lors de la disparition d'une faute, les résidus associés du module réversible défaillant, sont annulés.

3. **Détermination de l'état des modules :** L'état des modules est mis à jour à partir des fautes diagnostiquées et des informations contenues dans le tableau de synthèse AMDEC. S'il y a modification des états des modules le processus de diagnostic est conclu par l'émission d'un évènement correspondant. La sévérité de la faute sur le mode de fonctionnement courant est aussi associée à cet évènement.

Nous allons maintenant illustrer plus longuement les deux dernières étapes qui constituent le cœur du processus de diagnostic.

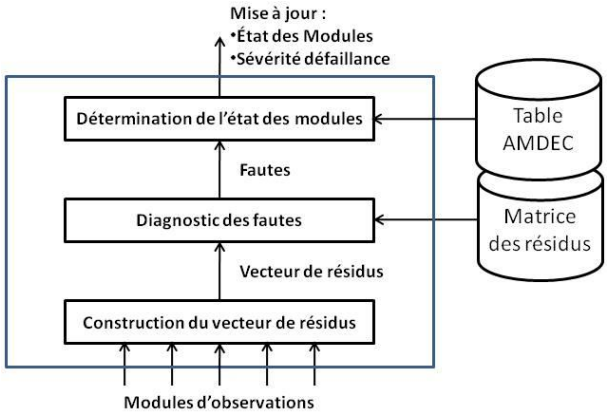


Figure 8.2 – Représentation de l'algorithme de diagnostic

8.2.2.1 Diagnostic des fautes et détermination de l'état des modules

Nous balayons ici chacun des cas de figures abordés dans la section 4.4.2.2 lors de l'apparition ou de la disparition des fautes.

8.2.2.1.a Apparition des fautes

Défaillance et faute unique : Le vecteur des résidus correspond ici à une seule signature. La faute originelle qui lui correspond est donc immédiatement localisée. Par exemple, l'activation du résidu **Batterie HS** entraine directement l'identification de la faute batterie au niveau du Module P3D. L'utilisation de la table AMDEC 7.1, permet de déterminer l'état de la fonction P3D-mod : cette faute induit un mode de défaillance complet et correspond à un niveau de sévérité fatal pour le fonctionnement du robot.

Défaillances et fautes multiples : le vecteur des résidus correspond à la composition de plusieurs signatures sur différents modules :

- **Défaillances multiples liées à une seule faute** : dans ce cas, il existe un lien causal entre les modules correspondants. Il est possible d'identifier la faute originelle en parcourant la chaine de causalité associée. Par exemple, si le vecteur contient les résidus **USaveugle** (module SNS) et **dispersion particules** (module LOC) actifs, l'analyse de signature permet d'établir que les dysfonctionnements sont localisés sur deux modules UST et MCL. Le lien causal correspondant (colonne Modules liées fonctionnellement table 7.1) permet d'induire que l'origine des perturbations du module MCL est en fait localisée au niveau du module UST. Seule la faute **Sonar aveugle** est donc diagnostiquée comme active. L'analyse de la matrice AMDEC permet de conclure que la fonction UST est partiellement défaillante associée à un niveau de sévérité moyenne. [1]
- **Fautes multiples indépendantes** : Il n'existe alors aucun lien causal entre ces fautes et

1. Cette défaillance du module UST ne permet plus d'utiliser le module MCL dans de bonnes conditions bien qu'il soit intrinsèquement considéré comme sain

donc entre les modules. L'analyse de signature conduit à identifier le vecteurs de résidus comme l'union de signatures connues sans qu'une chaine de causalité puisse être mise en évidence. Par exemple, si les résidus **UShors-service** et **Batterie HS** sont actifs, les fautes liées relèvent des fonctions P3D-mod et P3D-sonar. Puisqu'aucun lien n'est mis en évidence entre ces modules, les deux fautes sont considérées comme simultanément actives. Le tableau AMDEC permet de conclure que les fonctions P3D-mod et P3D-sonar sont complètement défaillantes et entrainent des sévérités fatale et grave sur le fonctionnement du robot. Seule la sévérité fatale, la plus pénalisante, pour le fonctionnement du robot, sera prise en compte lors du processus de recouvrement.

Plusieurs fautes avec une même signature : Dans ce cas le processus de diagnostic ne permet pas de distinguer les fautes. Les fautes indiscernables sont a priori déclarées actives. Pour lever l'incertitude il est possible de faire appel aux capacités cognitives d'un opérateur. Par exemple si les résidus **retour en arrière**, **ecart localisation** et **dérivée localisation** sont actifs en même temps, les fautes **sol glissant** du module P3D-roue et **Perturbation MCL** sont potentiellement à l'origine de défaillances, seul l'Homme pourra diagnostiquer l'origine de la faute. Cette configuration correspondra à un niveau de sévérité Grave pour pouvoir engager ce type de processus de recouvrement.

8.2.2.1.b Disparition des fautes

Lorsqu'un dysfonctionnement affecte transitoirement une partie du robot ou de l'architecture, le mécanisme de tolérance aux fautes conduit à l'éliminer par recouvrement. Cependant pour pouvoir rétablir le système robotique dans une configuration plus performante il faut être capable de détecter la fin du dysfonctionnement en continuant à l'observer. Par exemple, si le résidu **USaveugle** disparaît, la faute **Sonar aveugle** est elle aussi désactivée. Le module réversible UST partiellement défaillant est à nouveau défini comme sain. Dans cette situation, pour permettre d'engager la phase de recouvrement, la mise à jour de l'état est associée à un niveau de sévérité nul représenté par $\varnothing$.

Le processus de diagnostic mis en œuvre s'appuie largement sur l'analyse AMDEC qui a été pratiquée. Celle-ci a permis de construire la table des résidus qui sous-tend l'identification des fautes. Un tableau rassemblant toutes les informations nécessaires à la conduite du processus de diagnostic et à la détermination de l'état des modules, a été synthétisé en utilisant les informations extraites des diagrammes d'Ishikawa.

A l'issue de la phase de diagnostic, si l'état d'un module est modifié ou si le diagnostic n'a pu aboutir (signature identique), un évènement est émis vers les superviseurs pour pouvoir engager la phase de recouvrement que nous allons maintenant présenter.

8.3 Phase de recouvrement : Les superviseurs

La dernière phase du processus de la tolérance aux fautes est la phase de recouvrement. Elle doit, en fonction du contexte de la mission, choisir la réaction la mieux adaptée aux fautes perturbant le fonctionnement du robot.

La logique de l'architecture COTAMA distingue le niveau exécutif et synchrone permettant le déploiement de la boucle de contrôle et où les modules d'Observation sont déployés, du niveau asynchrone en charge de la dimension décisionnelle. Dans notre cas celle-ci recouvre, non seulement la gestion de l'évolution de la mission proprement dite, mais aussi, la prise en charge des mécanismes décisionnels de tolérance aux fautes liés à la phase de recouvrement. De façon à structurer le niveau décisionnel vis-à-vis de ces deux aspects nous proposons d'introduire une classification des évènements :

- Gestion de la mission : Ces évènements font évoluer la mission en permettant l'enchainement des objectifs et des sous-objectifs.
- Gestion de la tolérance aux fautes : Lors de la mise à jour de l'état des modules (défaillance, réversibilité), ces évènements s'enchainent pour ajuster au mieux, par recouvrement, le contrôle du robot et permettre à la mission de continuer.

Cette dichotomie nous a conduit à mieux structurer le niveau décisionnel en complétant sa décomposition en un ensemble de superviseurs (figure 8.3) aux contours clairement identifiés et qui vont être maintenant décrits dans le cadre de la mission de livraison de courrier.

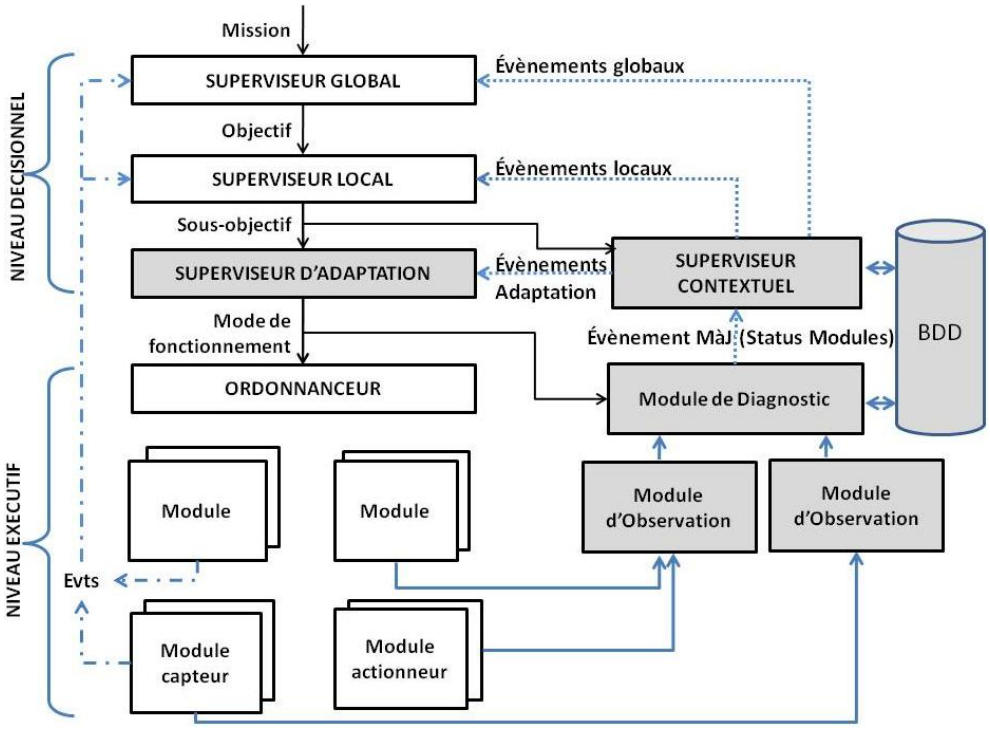


Figure 8.3 – Architecture COTAMA après modification

8.3.1 Le Superviseur Contextuel pour la gestion du recouvrement

Le superviseur contextuel est la clef de voute du mécanisme de recouvrement, dernière phase de la tolérance aux fautes. Son rôle est d'identifier, parmi les modes de fonctionnement des niveaux d'autonomie envisageables, celui qui sera le plus à même de répondre au changement d'état détecté par le Module de Diagnostic.

Il utilise pour le mode de fonctionnement courant l'état de tous les modules, et le niveau de sévérité reçu, pour exécuter l'algorithme de recouvrement présenté 4.4.3.5 et déterminer quelle est la réaction adéquate. Celle-ci permet de définir, par le biais d'autres superviseurs, le niveau d'autonomie et le mode de fonctionnement le plus adapté à la situation. En fonction de l'évaluation contextuelle réalisée, il génère trois types d'évènements :

- évènement d'adaptation lorsqu'il ne faut qu'adapter le mode de fonctionnement courant en conservant le même niveau d'autonomie.
- évènement local lorsqu'il est nécessaire de changer de niveau d'autonomie
- évènement global lorsque la mission doit être abandonnée ou lorsqu'il est nécessaire de stopper le robot dans une configuration d'attente.

Pour le niveau d'autonomie retenu par l'algorithme, le choix du mode de fonctionnement s'appuie sur une liste ordonnée, établie en utilisant, par ordre de priorité, les critères suivants :

1. Les risques pour le robot et son environnement.
2. L'efficacité subjective (choix de l'expérimentateur) d'un mode vis-à-vis d'un autre. Par exemple aucune comparaison des modes de fonctionnement avec DVZ ou SMZ n'a été faite. Cependant la SMZ est choisie prioritairement lors des expérimentations pour évaluer sa robustesse.

La liste obtenue, présentée dans la table 8.10 , précise pour chacun des modes de fonctionnement la liste des modules de contrôle utilisés.

Nous allons maintenant décrire le comportement de chacun des superviseurs impliqués par le Superviseur Contextuel dans l'action de recouvrement. Pour en faciliter la représentation nous ferons appel au formalisme réseau de Petri [Murata, 1989] qui autorise une description graphique et comportementale facilement interprètable d'un système à évènements discrets. Nous commençons ici par détailler le superviseur global.

Sous-objectif	Autonomie	Liste des modules actifs								
Suivi de chemin optimal SMZ	Autonome	P3D	SIM	UST	ODO	MCL	NAV	SMZ	CTL	
Suivi de chemin optimal DVZ	Autonome	P3D	SIM	UST	ODO	MCL	NAV	GUI	CTL	DVZ
Suivi de chemin SMZ sans MCL	Autonome	P3D	SIM	UST	ODO		NAV	SMZ	CTL	
Suivi de chemin DVZ sans MCL	Autonome	P3D	SIM	UST	ODO		NAV	GUI	CTL	DVZ
Suivi de chemin sans sonar	Autonome	P3D	ODO	NAV	GUI	CTL				
Choix HRI	Téléprogrammé	NAV	UDP	P3D						
Re-localisation robot	Téléprogrammé	NAV	UDP	P3D						
Nouveau chemin	Téléprogrammé	NAV	UDP	P3D						
Diagnostic distant modules fautifs	Téléprogrammé	NAV	UDP	P3D						
Téléopération DVZ	Téléopération	P3D	UST	ODO	NAV	UDP	CTL	DVZ		
Téléopération	Téléopération	P3D	ODO	NAV	UDP	CTL				

Table 8.10 – Classification des sous-objectifs avec leurs modules respectifs

8.3.2 Le Superviseur Global pour la gestion de la mission

Le superviseur global doit à la base gérer l'exécution de la mission. Sur le réseau de Petri de la figure 8.4 qui décrit son comportement, on constate qu'il enchaine les différentes étapes associées : **ATTENTE LIVRAISON**, **DÉPLACEMENT VERS ÉMETTEUR**, **RÉCEPTION COURRIER**, **DÉPLACEMENT VERS DESTINATAIRE** et **LIVRAISON COURRIER**.

A ce comportement s'ajoute celui lié à tolérance aux fautes pour les défaillances de niveau fatal de sévérité. Deux cas peuvent être envisagés en fonction de la nature réversible ou non du module de défaillant :

- Le robot attend un hypothétique rétablissement d'une fonctionnalité. Le Superviseur Contextuel adresse au superviseur global un évènement de type **Arrêt temporaire**, qui le conduit dans un des deux états d'**ATTENTE**. Lors du rétablissement et de la réception d'un évènement **Reprise**, le robot reprendra sa mission, conformément au mode de fonctionnement déterminé par le superviseur contextuel.
- La sévérité est sans espoir de réversibilité, le robot se positionne alors, s'il le peut, dans un état sécuritaire **STOP** avant de mettre fin de façon prématurée à la mission.

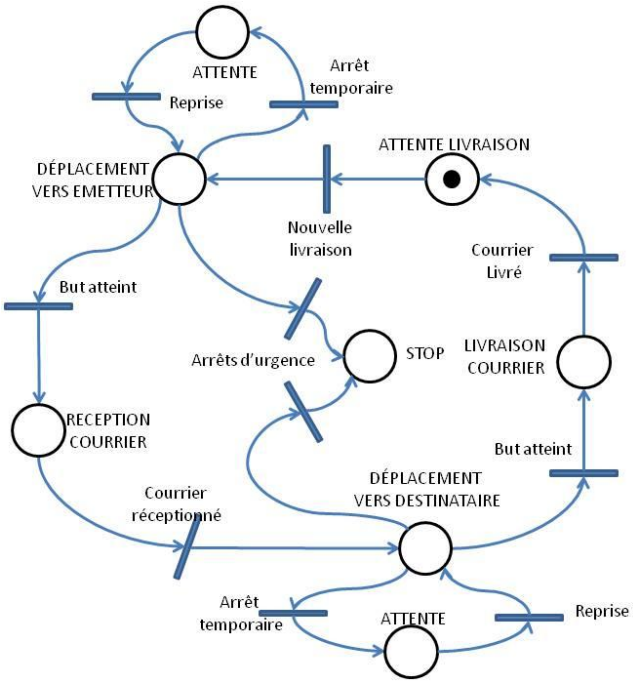


Figure 8.4 – Représentation du superviseur global en réseau de Petri pour la mission livraison de courrier

8.3.3 Le Superviseur Local pour l'ajustement de l'autonomie

Le rôle premier du superviseur local est de décomposer les objectifs de la mission en sous-objectifs. Avec la logique déployée pour l'architecture, il devrait aussi mettre en œuvre l'ensemble des mécanismes de recouvrement de la tolérance aux fautes à savoir : l'ajustement du niveau d'autonomie, du mode de fonctionnement, et la reconfiguration locale de module. Pour mieux structurer le niveau décisionnel et par la même en faciliter son développement, les missions du superviseur local ont été limitées à la gestion des objectifs de la mission et du niveau d'autonomie du robot.

Le comportement du superviseur local est présenté dans le réseau de Petri de la figure 8.5. Le déplacement vers un utilisateur est initialement décomposé en **GÉNÉRATION DE CHEMIN** puis **SUIVI DE CHEMIN**. Sur émission (par le superviseur contextuel) d'un évènement demandant le changement du niveau d'autonomie, le sous-objectif en cours est stoppé, pour lancer un sous-objectif spécifique (**CHOIX HRI**) où l'opérateur définit le niveau d'autonomie qu'il souhaite utiliser par la suite pour continuer la mission. Deux sous-objectifs sont alors utilisables, **TÉLÉOPÉRATION** et **TÉLÉPROGRAMMATION**. L'étape **ARRIVÉE** est symbolique car l'atteinte de l'objectif conduira le superviseur global à commuter automatiquement la mission vers un nouvel objectif.

Les missions, reconfiguration locale des modules et adaptation des modes de fonctionnement, initialement dévolues au superviseur local, ont été confiées dans le cadre de notre architecture tolérante aux fautes au superviseur d'adaptation présenté ci dessous.

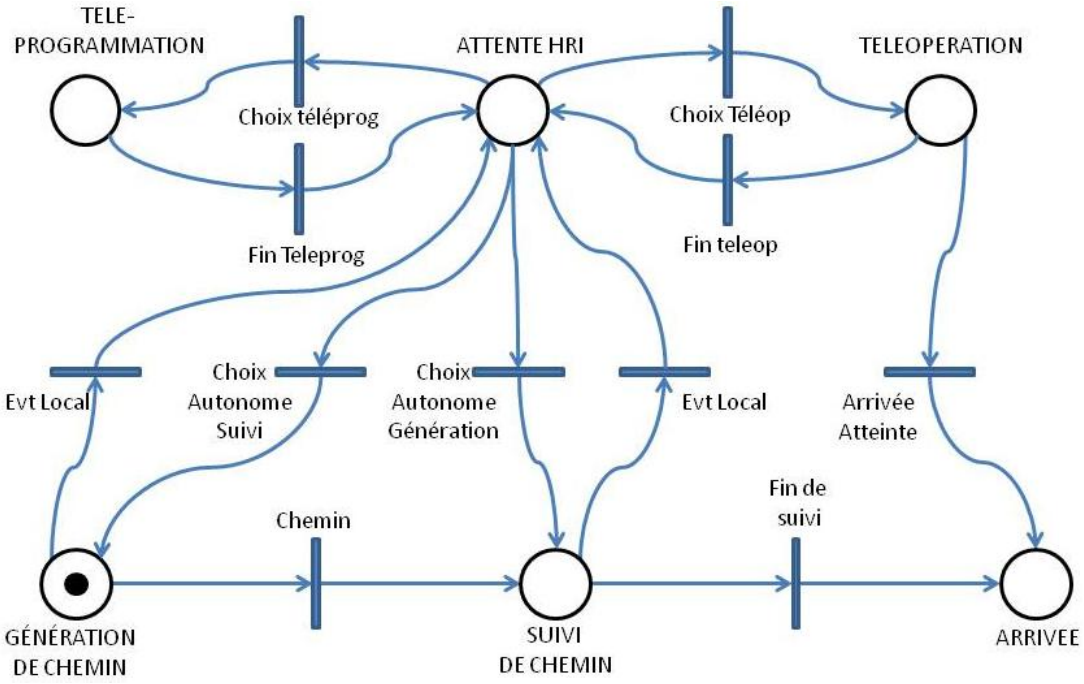


Figure 8.5 – Représentation du superviseur local en réseau de Petri pour l'objectif **déplacement vers utilisateur**

8.3.4 Le Superviseur d'Adaptation pour la reconfiguration et l'adaptation locale

Le but de ce nouveau superviseur d'adaptation est, pour un sous-objectif choisi donné, de mettre en œuvre le mode de fonctionnement le plus adapté choisi, ou de reconfigurer localement un module.

Lors de la réception d'un sous-objectif, le superviseur d'adaptation attend que le superviseur contextuel définisse le meilleur mode de fonctionnement en fonction du contexte actuel du robot.

La figure 8.6 donne à l'aide d'un réseau de Petri une représentation simplifiée du comportement du sous-objectif de suivi de chemin autonome.

Le **SUIVI DE CHEMIN SMZ**, considéré comme optimal, est activé lors du fonctionnement sain du robot. Selon la faute rencontrée différentes solutions sont ensuite proposées :

- Une erreur localisée sur le module MCL (**Erreur temps réel MCL**) peut être corrigée par la **RECONFIGURATION** de celui-ci en diminuant le nombre de particules du filtre particulaire.
- Une défaillance complète du module SMZ, (**SMZ HS**) peut être compensée par l'activation du **SUIVI DE CHEMIN DVZ**.
- Si la performance des capteurs ne leur permet pas de représenter l'environnement et que le résidu **sonar aveugle** est activé, l'utilisation du module MCL n'est plus possible. Le choix d'un mode de fonctionnement dégradé est donc fait. (**SMZ OU DVZ SANS MCL**)
- À l'inverse si la capacité des capteurs est retrouvée, (**sonar OK**),on peut retourner dans un mode de fonctionnement plus performant.

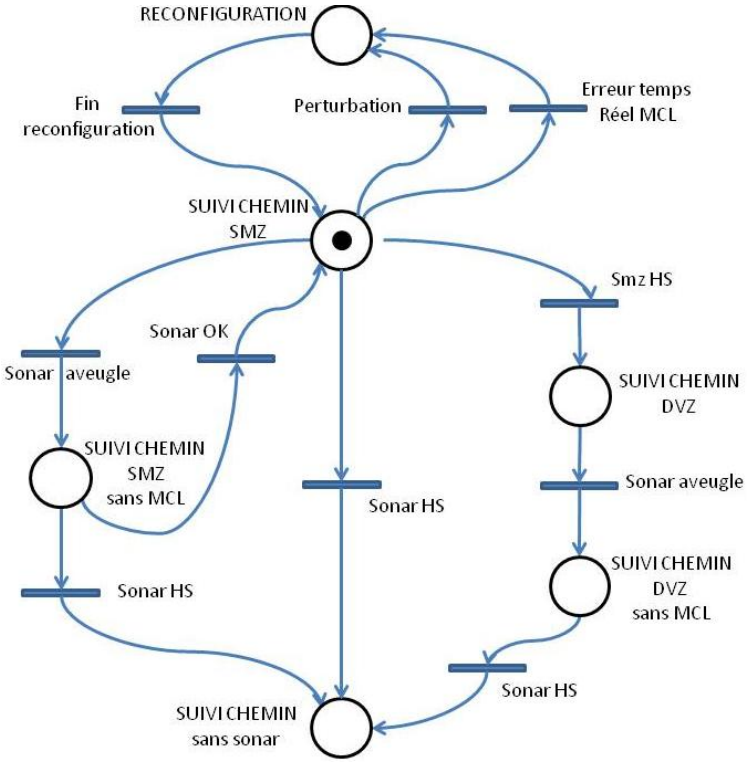


Figure 8.6 – Représentation du superviseur adaptation en réseau de Petri

– Pour finir, si tous les sonars sont considérés comme hors-service (**sonar HS**) un mode de fonctionnement sans sonars est engagé. L'utilisation de ce mode autonome est discutable car il ne propose aucune sécurité mais compte tenu de la faible vitesse du robot, nous le considérons comme acceptable. L'interaction Homme-Robot ne sera donc utilisé que si le robot souffre d'un autre dysfonctionnement, par exemple un choc.

La figure 8.6 n'est qu'un modèle partiel car la représentation de tous les liens potentiels pouvant être créés lors du recouvrement asynchrone de capacités surchargerait inutilement le schéma. Ci-dessous sont listées les informations manquantes :

– La reconfiguration du module MCL est accessible du suivi de chemin avec DVZ.
– Le retour après recouvrement des capacités sonars du suivi de chemin avec DVZ sans MCL vers celui comprenant l'algorithme de localisation.

8.4 Conclusion

Ce chapitre a permis d'expliciter et de mettre en œuvre, dans le cadre d'une mission de livraison de courrier par un robot Pioneer P3-DX, la méthodologie de construction d'une architecture de contrôle tolérante aux fautes. Pour cela nous nous sommes appuyés sur l'analyse AMDEC préalable que nous avons menée. Celle-ci constitue la pierre angulaire du déploiement, au sein de l'architecture, des mécanismes de tolérance aux fautes, au travers des phases de détection,

diagnostic et recouvrement.

La détection a été intégrée au sein des modes de fonctionnement par l'intermédiaire d'un ensemble de Modules d'Observation dédiés permettant de détecter l'occurrence d'un dysfonctionnement.

L'analyse des signatures conduite à partir des résidus ainsi obtenus et des informations rassemblées dans le tableau de synthèse AMDEC permet de diagnostiquer l'origine des dysfonctionnements, les modules fautifs. Le Module de Diagnostic détermine aussi leur état (opérationnel ou non) et le niveau de sévérité des défaillances.

Enfin le niveau décisionnel a été structuré de façon à déployer les mécanismes de recouvrement adaptés au contexte fautif identifié. Pour cela un superviseur contextuel, qui peut être considéré comme l'élément central du processus décisionnel, choisit, en fonction des informations contextuelles fournies par le niveau exécutif, et les règles de priorités, définies les actions à engager ainsi que le superviseur auquel elles doivent être appliquées. Celles-ci sont déployées, en fonction de la sévérité estimée de la défaillance au travers de trois superviseurs. Au superviseur global dédié à la gestion la mission et des évènements à même de la remettre en cause s'ajoutent deux superviseurs pour faciliter la description fonctionnelle des objectifs. Le superviseur local gère chacun des objectifs de la mission et l'adaptation du niveau d'autonomie de leurs sous-objectifs. Le superviseur d'adaptation met en œuvre pour chaque sous-objectif l'un des modes de fonctionnement potentiels.

Il est important de noter que les réseaux de Petri associés aux superviseurs ne sont que des représentations comportementales car ils sont codés sous la forme de machines à états plus génériques dans l'architecture COTAMA. Le développement actuel de l'architecture ne correspond pas exactement à la structuration proposée dans ce chapitre. En effet, pour en faciliter le développement, le module de diagnostic et le superviseur contextuel sont pour l'instant regroupés au sein d'un même module appelé **Module Global d'Observation (MGO)**.

Nous disposons maintenant, pour la mission de livraison de courrier, d'une architecture tolérante aux fautes. Pour clore ce document nous allons détailler et analyser son fonctionnement expérimental.

Cinquième partie

Expérimentation de l'architecture tolérante aux fautes

Chapitre 9

Description et analyse d'une mission

Cette partie va dans un premier temps énoncer les conditions expérimentales mises en œuvre avant de détailler chacun des points significatifs du déroulement de la mission. Ensuite, avant de conclure, une évaluation du comportement de l'architecture tolérante aux fautes est proposée.

9.1 Conditions expérimentales de la mission

Dans le but de réaliser une mission courte permettant de détecter l'occurrence d'un maximum de défaillances et d'analyser les réactions de recouvrement engagées nous avons décidé de déployer une mission encadrée par des conditions d'évolution spécifiques. Les différents évènements qui viendront troubler sont déroulement sont donc scriptés. Ils sont soit introduits volontairement dans le code pour perturber le fonctionnement de l'architecture à des dates précises, soit déclenchés manuellement sur le robot ou son environnement. Pour faciliter l'exécution de cette mission et pallier à des soucis récurrents de portée et de précision des sonars, l'expérimentation est réalisée HIL (Hardware In the Loop). Le robot est donc posé sur cales pour permettre aux roues de tourner dans le vide et de faire fonctionner le système actionneurs/capteurs. Par conséquent, les sonars sont simulés, à l'aide d'une carte de l'environnement et de la position courante estimée du robot, tandis que les autres capteurs (odométriques et bumpers) et les actionneurs sont réels.

L'objectif de la mission est de récupérer, au sein du laboratoire, un objet au niveau d'un bureau A et de le livrer au bureau B. La figure 9.1 présente la trajectoire expérimentale suivie par le robot. Elle singularise sur la carte un ensemble de points, numérotés de 1 à 10 où des évènements pertinents ont été observés et qui seront expliqués en détails par la suite. Lors du déplacement, la vitesse du robot est de 0,3 m/s. La période de la boucle de contrôle de chaque sous-objectif est de 0,1 s.

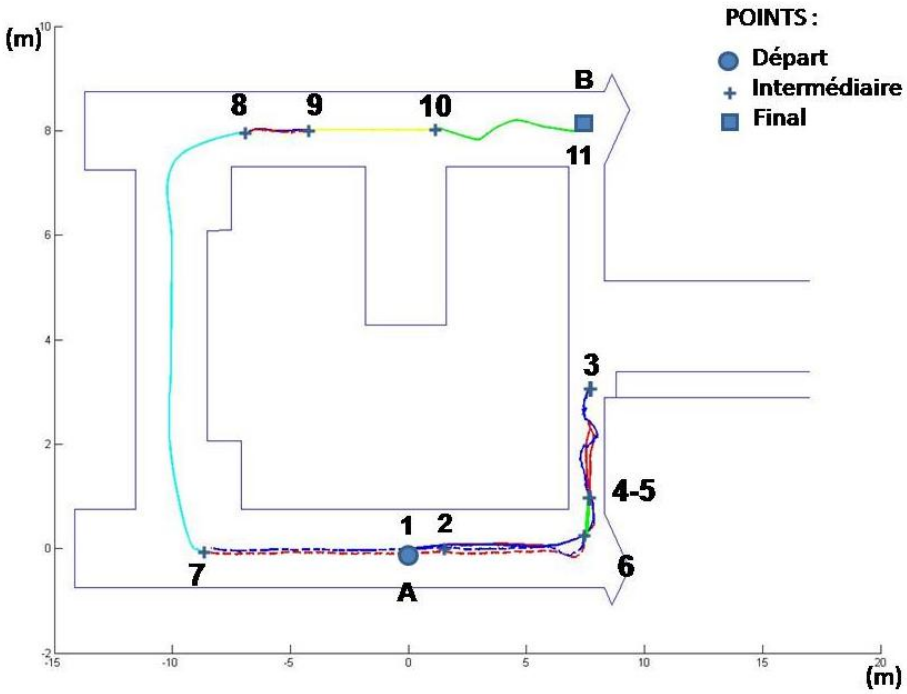


Figure 9.1 – Trajectoire expérimentale du robot

Pour pouvoir suivre le fonctionnement du robot dans son environnement et interagir avec lui, un opérateur suit le fonctionnement du robot à l'aide d'une interface simple développée spécifiquement pour l'expérience HIL. Elle permet deux types d'interactions :

- visualiser l'environnement du robot, ses déplacements, les localisations odométriques et de Monte-Carlo estimées. Par ailleurs, certaines informations utiles lors des phases de test sont aussi disponibles telles que les rayons des sonars, les référentiels utiles à l'évitement d'obstacle et la position du rabbit du suivi de chemin.
- agir sur le robot, c'est-à-dire pouvoir à distance définir sa position actuelle, le chemin qu'il doit suivre, ou le téléopérer à l'aide d'un (cartésien) ou de deux (articulaire) joysticks [1].

La photographie 9.2 présente le poste de travail où sont réalisées les expériences HIL. L'ordinateur portable sous RTAI linux qui accueille l'architecture tolérante aux fautes est embarqué sur le robot pour le contrôler. Il est relié par le biais de sa connexion Wifi et de l'intranet du laboratoire à un ordinateur fixe qui supporte l'interface de visualisation et de téléopération.

1. Réalisé lors d'un stage de fin d'études IUT [Lopa, 2009]

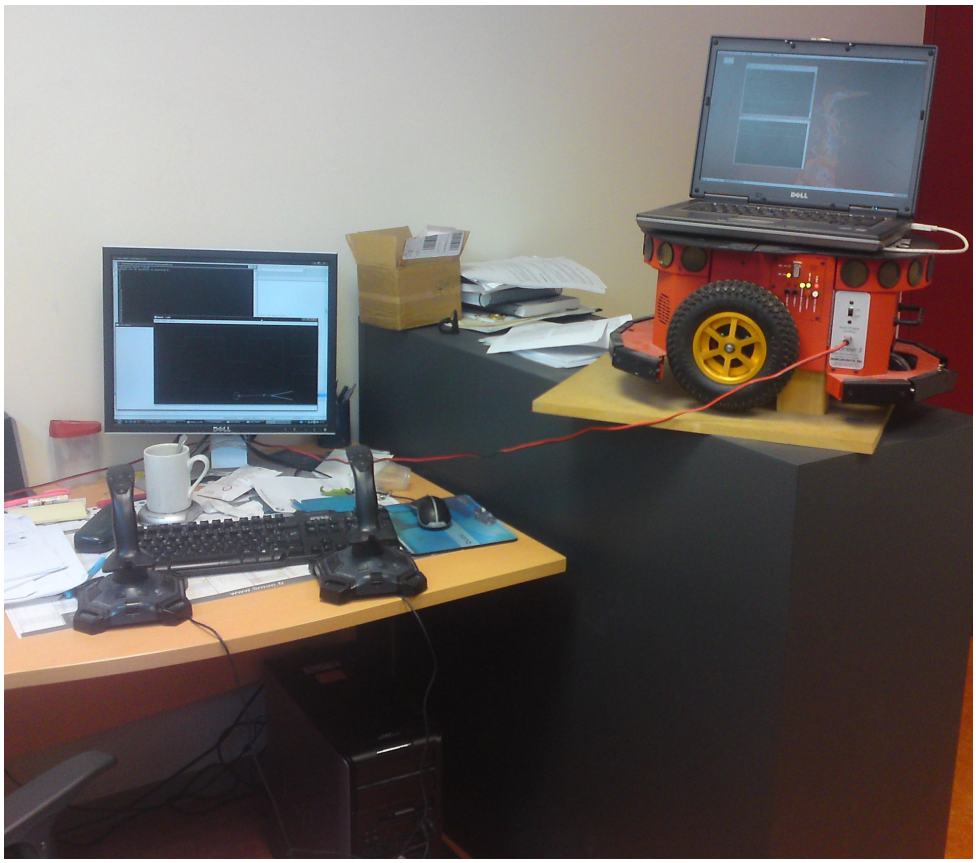


Figure 9.2 – Poste de travail pour l'expérimentation HIL

Après avoir présenté le contexte expérimental, et avant de détailler le déroulement de la mission, nous souhaitons maintenant focaliser brièvement notre discours sur les éléments pertinents de la matrice de résidus qui a été définie, et sur lesquels nous appuyions largement le processus de tolérance aux fautes.

9.2 Retour sur la matrice des résidus et le tableau AMDEC

La matrice des résidus de la figure 9.1 met en évidence (zone non grisée) ceux qui seront utilisés dans le cadre de l'expérimentation et auxquels nous nous réfèrerons. De même, la figure 9.2 représente le tableau récapitulatif de l'analyse AMDEC avec les fautes intéressantes pour la suite de ce chapitre en non grisées.

Le déroulement de la mission et le comportement de l'architecture de contrôle tolérante aux fautes vont être maintenant présentés dans la section suivante.

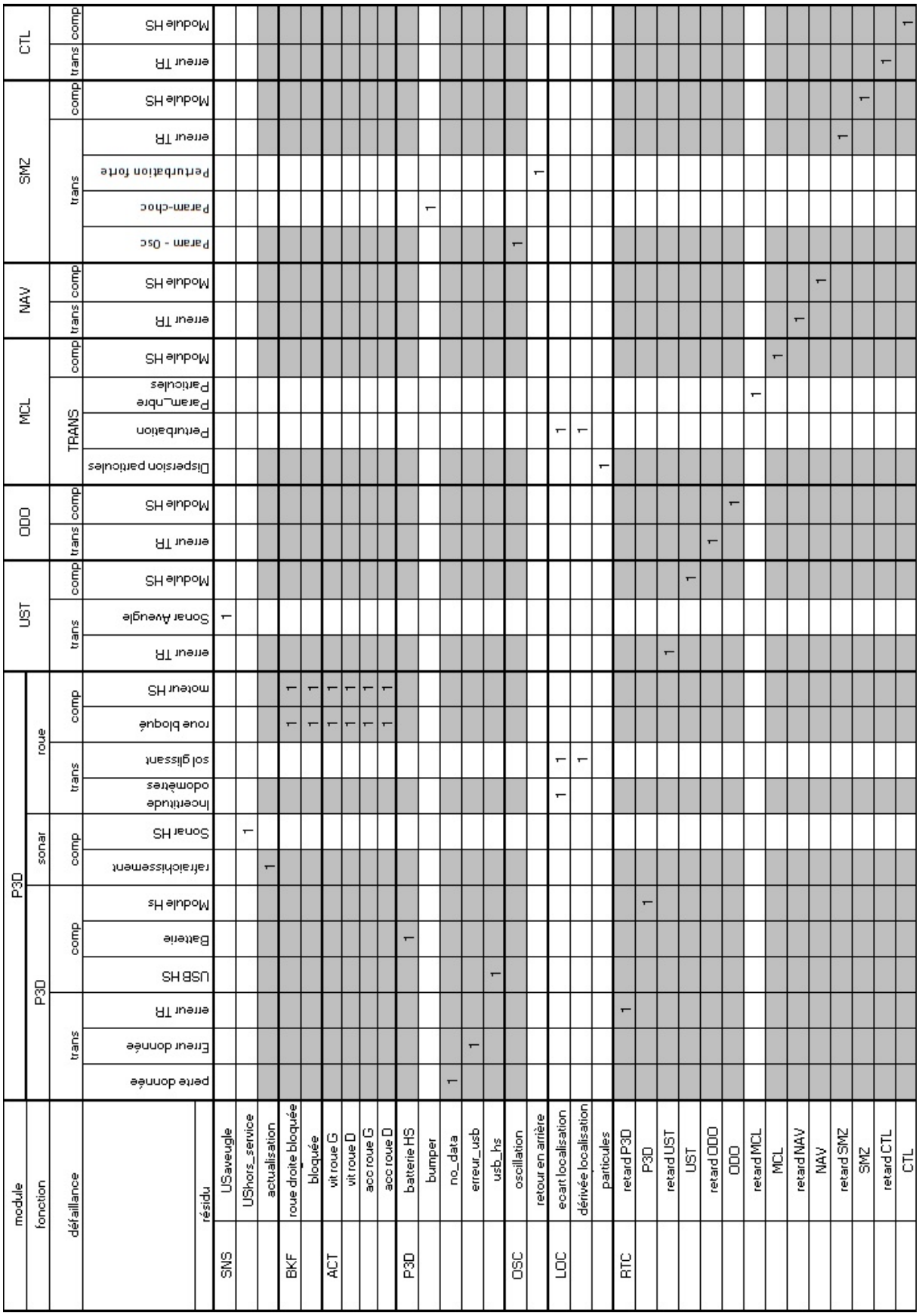


Table 9.1 – Rappel : Matrice des résidus pour le suivi de chemin autonome SMZ

MODULE	MODE DEFAILLANCES		CAUSES DES DEFAILLANCES			SEVERITE	ACTIONS	
	classe	Type	Fautes		Modules liés fonctionnellement		Principe d'observation	Informations Recouvrement
			Identificateur	Information				
P3D(mod)	Partielle	transitoire	P3D-mod1	Tps_Com_USB		ignorée	Watch dog variable	
			P3D-mod2	Perte donnée				
			P3D-mod3	Erreur TR			Watchdog activation	
	Complète	transitoire			CTL(C)	grave		
		permanent	P3D-mod4	Batterie		fatale	Information niveau de batterie robot	Arret
			P3D-mod5	Module HS			Watchdog activation	Arret
			P3D-mod6	USB HS			Watchdog communication	Arret
P3D(roue)	Partielle	transitoire	P3D-roue1	Sol glissant		ignorée	Perturbation ponctuelle non détectée	Utilisation préventive du MCL pour correction passive
		permanent	P3D-roue2	Incertitude odomètres			Perturbation permanente	
	Complète	permanent			P3D(mod)(C)	fatale		Arret
			P3D-roue4	Roue bloqué			Banque filtre de kalman	Arret
			P3D-roue5	Moteur HS			Banque filtre de kalman	Arret
P3D(sonar)	Complète	transitoire	P3D-sonar1	Rafraichissement		moyen	Watchdog activation	MCL non utilisable
		permanent			P3D(mod)(C)			
			P3D-sonar2	Sonar HS			Watchdog - vivacité des valeurs	Aucun sonar
UST	Partielle	transitoire		Erreur TR		ignorée	Watchdog activation	
			UST1	Sonar aveugle		moyen	Limite d'un nombre de sonar actif	MCL non utilisable
	Complète	permanent	UST2		P3D(sonar)(C)			Mode sans US
			UST3	Module HS			Watchdog activation	
ODO	Partielle	transitoire			P3D(roue)(P)	faible		
			ODO1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent			P3D(roue)(C)	fatale		Arret
			ODO2	Module HS			Watchdog activation	Arret
MCL	Partielle	transitoire	MCL1	Perturbation		faible	Différence entre position ODO et MCL	reconfig NAV avec odométrie
			MCL3	Param_nbre Particules			Watchdog activation	reparam nb de particules
	Complète	transitoire	MCL4	Dispersion particules	UST(P)	grave	Analyse Ecart type des particules	
		permanent			UST(C)			Mode sans MCL
			MCL5	Module HS			Watchdog activation	
NAV	Partielle	transitoire	NAV1	Erreur TR		ignorée	Watchdog activation	
	Complète	transitoire			MCL(C)	grave		
		permanent	NAV3	Module HS		fatale	Watchdog activation	Arret
SMZ	Partielle	transitoire	SMZ1	Erreur TR		ignorée	Watchdog activation	
	Complète		SMZ2	Param - Osc		faible	Observateur d'oscillation	
		transitoire	SMZ3	Param-choc		grave	Information bumper robot	Appel Opérateur
			SMZ4	Perturbation forte		grave	Retour arrière	Appel Opérateur
		permanent			(NAV(C) - UST(C))	moyen		Changement d'algorithme de suivit de chemin
			SMZ5	Module HS			Watchdog activation	
CTL	Partielle	transitoire	CTL1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent			SMZ (C)			
			CTL2	Module HS		grave	Watchdog activation	

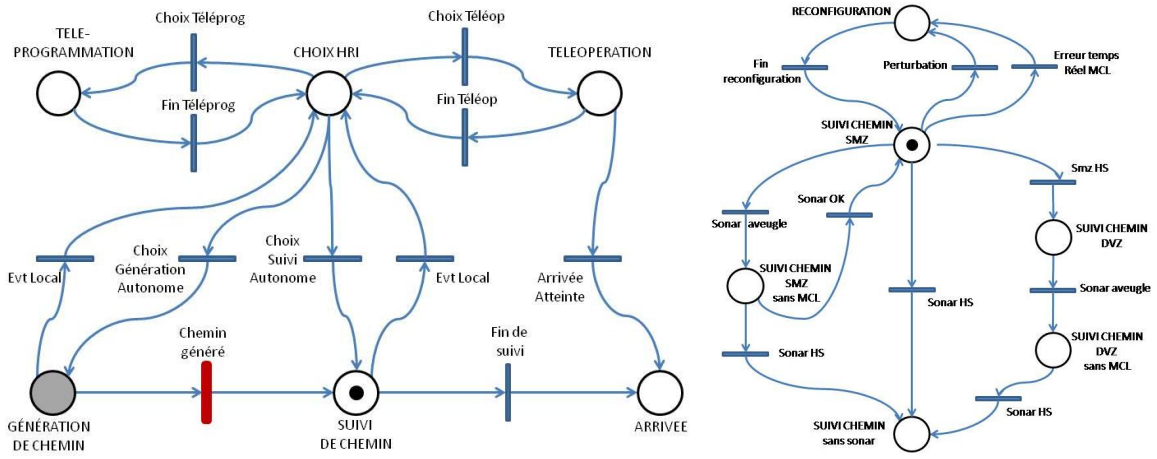
Table 9.2 – Rappel : Tableau AMDEC pour le suivi de chemin autonome SMZ

9.3 Déroulement de la mission

Comme dans le reste du document, pour ne pas complexifier inutilement la description de la mission, nous allons nous polariser sur la partie suivi de chemin entre l'expéditeur A et le destinataire B. La description de l'expérimentation va se focaliser successivement sur les différents points significatifs indiqués figure 9.1 en s'attachant à détailler les processus de détection, diagnostic et recouvrement mis en place pour achever la mission avec succès.

9.3.1 Phase 1 : Lancement de l'expérimentation

Une fois le courrier réceptionné au point A, position initiale du robot, l'objectif **Se déplacer vers (Point B)** est donc lancé. Au niveau du superviseur local (figure 9.3a), le sous-objectif **Génération de chemin** est alors activé (place grisée). Lorsque ce dernier est construit, l'évènement **Chemin généré** permet d'engager un nouveau sous-objectif de **Suivi de chemin** autonome (place avec jeton). Le superviseur d'adaptation du suivi de chemin autonome (figure 9.3b) est alors activé : il lance le mode de fonctionnement **Suivi de chemin SMZ** (place avec jeton). Ce mode de fonctionnement, qui a été considéré comme optimal, est activé puisque initialement l'ensemble des ressources du robot sont opérationnelles.



(a) Superviseur local (b) Superviseur d'adaptation

Figure 9.3 – Phase 1 : Evolution des superviseurs au lancement de l'expérimentation

Les modules exécutés dans ce mode de fonctionnement sont présentés dans la table 9.3. Ils sont classés en fonction de leur rôle au sein de l'architecture, contrôle du robot (Contrôle), lien de supervision avec l'opérateur qui observe l'évolution du robot (Supervision), observation du comportement du robot et de l'architecture (Observation). Dans ce mode qui est le plus performant, 18 modules sont impliqués pour contrôler le robot en utilisant une stratégie de suivi de chemin avec évitement d'obstacle et une localisation de type Monte-Carlo.

Types de module	Sous-objectif : suivi de chemin autonome SMZ							
Contrôle	P3D	SIM	UST	ODO	MCL	NAV	SMZ	CTL
Supervision	UDP							
Observation	OSC	SNS	ACT	OWF	LOC	BKF	RTC	MGO

Table 9.3 – Phase 1 : Modules actifs du sous-objectif suivi de chemin autonome SMZ

9.3.2 Phase 2 : Détection de faute unique de dépassement temporel

Après le démarrage du suivi de chemin, l'observateur temps réel RTC détecte rapidement (au point 2 de la figure 9.4) une défaillance temps réel du module MCL. Le résidu **retard MCL** est le seul actif, l'analyse de signature permet de diagnostiquer que le robot doit faire face à une faute unique **Param_NbreParticules** provoquant, d'après le tableau AMDEC, une défaillance partielle du module MCL de sévérité faible.

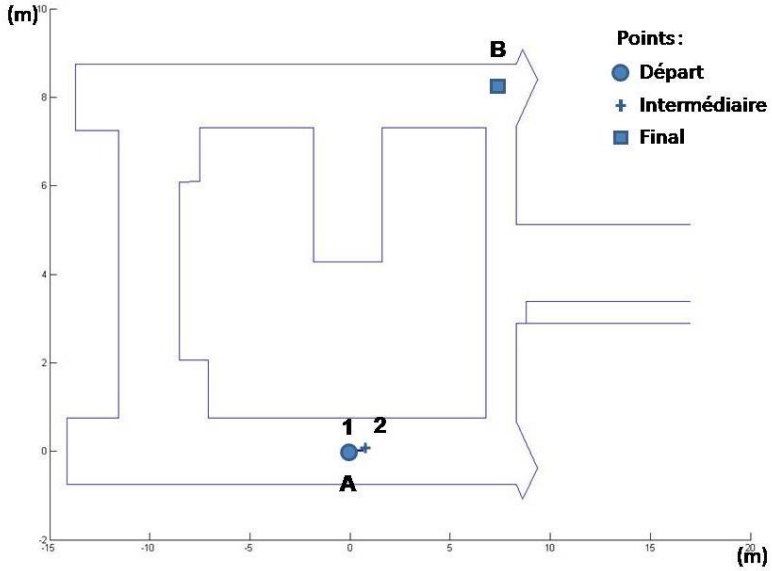


Figure 9.4 – Phase 2 : Trajectoire du robot

Cette faute de sévérité faible permet au superviseur contextuel de conclure qu'elle peut être corrigée par une simple adaptation locale en reparamétrant le module MCL concerné. Il adresse donc un évènement d'adaptation au superviseur d'adaptation du sous-objectif en cours. La solution de recouvrement adoptée conduit ce dernier (figure 9.5) à réduire le nombre de particules de l'algorithme de localisation diminuant ainsi sa charge de calcul, et donc son temps d'exécution. Le module MCL a été reparamétré mais le mode de fonctionnement du robot reste le même : Suivi de chemin SMZ autonome. Les modules impliqués restent donc les mêmes que ceux de la table 9.3.

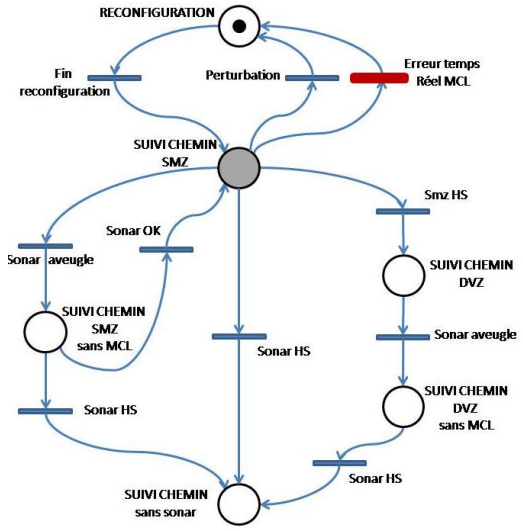


Figure 9.5 – Phase 2 : Evolution du superviseur d'adaptation de Suivi de chemin autonome

La table 9.4 suivante résume le processus de tolérance aux fautes mis en œuvre pour faire face à la défaillance détectée au point 2. On y retrouve toutes les informations pertinentes, à savoir : le mode de fonctionnement courant, les résidus observés, les fautes diagnostiquées, le(s) module(s) concerné(s), la sévérité retenue et le moyen de recouvrement adopté. Dans la suite, nous retrouverons pour chacune des phases de l'expérimentation ce tableau de synthèse.

Processus de tolérance	**Étapes**
Mode de fonctionnement	Suivi de chemin autonome SMZ
Observation	Dépassement temps réel MCL
Résidu	retard MCL
Faute	Param_NbreParticules
Module (Sévérité)	SMZ (faible)
Recouvrement	Reconfiguration : reparamétrage du module MCL

Table 9.4 – Phase 2 : Synthèse du processus de tolérance aux fautes

9.3.3 Phase 3 : Détection de défaillances multiples de localisation

On peut constater qu'à la fin de la phase 3 (figure 9.1, zoomée figure 9.6) l'évolution du robot devient hasardeuse. Cette situation est provoquée par la présence d'un obstacle non indiqué sur la carte. L'algorithme d'évitement d'obstacle SMZ essaie de contourner cet obstacle et entraîne le robot à faire demi-tour, comme l'indique la courbe rouge qui représente les données des capteurs odométriques. Cependant, l'algorithme de localisation de Monte Carlo ne détecte pas cette situation et se perd, comme l'indique la courbe bleue qui représente la localisation MCL. Ainsi, au point 3, le module d'observation LOC détecte une défaillance du mécanisme de localisation et génère les résidus **Ecart localisation** et **Dérivée localisation** correspondants.

Ceux-ci, comme le démontre la figure 9.6 détectent en fait une divergence notable (supérieure à un seuil fixé) entre la localisation par odométrie (**Position Odomètrie** sur la figure) et celle par la méthode de Monte-Carlo (**Position MCL**).

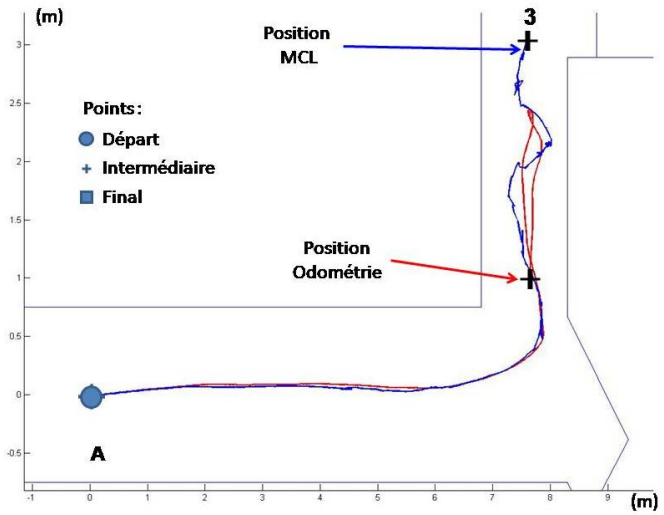


Figure 9.6 – Zoom sur la phase 3 de la mission

Dans ce cas, l'analyse de signature ne permet pas immédiatement de diagnostiquer l'origine de la défaillance détectée car elle peut correspondre à deux fautes distinctes : **sol glissant** (de la fonction P3D-roue) et **Perturbation** (du module MCL). En effet, une divergence des données de localisation peut correspondre à diverses situation comme un obstacle bouchant le chemin, mais également au patinage d'une ou des roues sur un sol glissant. Sans données supplémentaires (telles par exemple des informations sur le courant moteur pour détecter le patinage, ou des informations plus précises sur les tentatives d'évitement d'obstacle), ces fautes ne peuvent être diagnostiquées de façon autonome. Le tableau AMDEC nous révèle que les deux fautes concernées sont indépendantes, avec respectivement une sévérité ignorée et faible. Le processus de diagnostic pour des fautes multiples indépendantes (décrit section 4.4.2.2.b) sélectionne la faute de sévérité la plus importante, ici une **Perturbation** de l'environnement qui provoque une défaillance transitoire de sévérité faible du module MCL. L'action de recouvrement préconisée est alors la reconfiguration du module NAV afin de ne plus prendre en compte la localisation MCL pour la construction de l'état du robot, mais seulement les données odométriques. Comme pour la phase 2, le superviseur contextuel envoit donc un évènement spécifique (**Perturbation**) au superviseur d'adaptation concerné (figure 9.7).

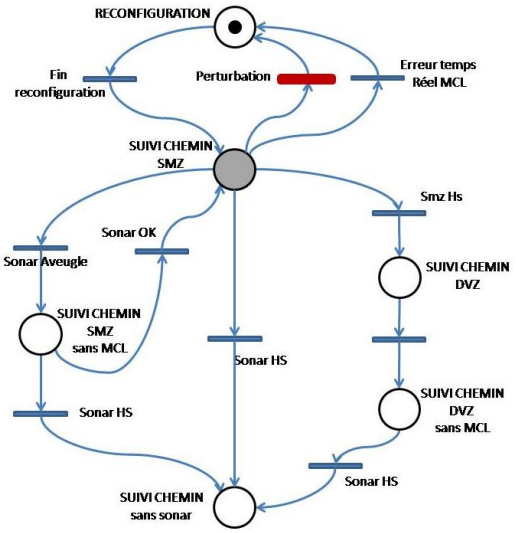


Figure 9.7 – Phase 3 : Evolution du superviseur d'adaptation de Suivi de chemin autonome

La table 9.5 synthétise les différentes informations qui ont entraîné la reconfiguration du module NAV suite à la détection d'une divergence des informations de localisation.

Processus de tolérance	**Étapes**
Mode de fonctionnement	Suivi de chemin autonome SMZ
Résidu	écart localisation et dérivée localisation
Faute	Multiples indépendantes : sol glissant ou Perturbation
Module (Sévérité)	P3D-roue (ignorée) et MCL (faible)
Recouvrement	Reconfiguration : reparamétrage de NAV

Table 9.5 – Phase 3 : Synthèse du processus de tolérance aux fautes

9.3.4 Phase 4 : Détection du retour arrière du robot

Une fois le reparamétrage de NAV effectué, le robot continue dans le même mode de fonctionnement : suivi de chemin SMZ autonome. La différence sera que l'algorithme SMZ, qui se base sur les informations fournies par le module NAV, utilisera donc maintenant les données de localisation du robot issue des capteurs odométriques. Or la localisation fournie par l'odométrie est très différente de la dernière localisation MCL enregistrée. L'algorithme de suivi de chemin SMZ va donc adapter la position du rabbit à la nouvelle position du robot : le rabbit fait un retour en arrière sur le chemin. Cette situation est alors détectée par le module d'observation OSC qui active le résidu **retour en arrière**. OSC n'avait pas détecté le retour arrière du robot pendant la phase 3, car il ne se base que sur les données du module SMZ, qui étaient jusqu'alors cohérentes.

L'analyse de la signature associée à ce résidu permet de diagnostiquer la faute de sévérité

grave **Perturbation forte** de l'environnement, associée au module SMZ. Ce niveau de sévérité conduit le superviseur contextuel, par l'envoi d'un **EvtLocal**, à demander au superviseur local d'engager une action de recouvrement impliquant l'aide d'un opérateur humain (figure 9.8). En effet, bien qu'un diagnostic est été effectué, l'origine exacte de la perturbation de l'environnement n'est pas connue par le robot. Il a besoin des capacités de l'Homme pour résoudre ce problème et finaliser les processus de diagnostic et recouvrement.

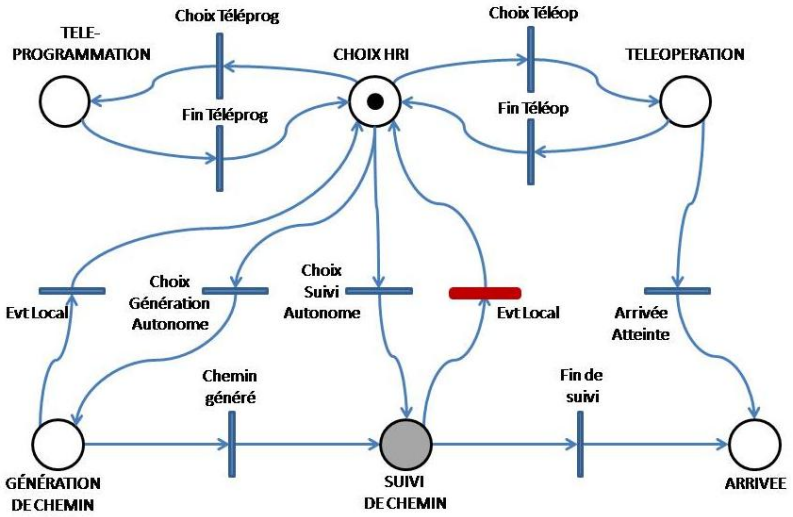


Figure 9.8 – Phase 4 : Modification du superviseur local pour lancer une interaction Homme-Robot

La table 9.6 synthétise les différentes informations qui ont conduit le processus de tolérance aux fautes à ajuster le niveau d'autonomie du robot pour engager l'opérateur dans une interaction distante avec le robot.

Processus de tolérance	**Étapes**
Mode de fonctionnement	Suivi de chemin autonome SMZ
Résidu	retour en arrière
Faute	Perturbation forte
Module (Sévérité)	SMZ (Grave)
Recouvrement	Appel à un opérateur humain : Choix HRI

Table 9.6 – Phase 4 : Synthèse du processus de tolérance aux fautes

9.3.5 Phase 5 : Diagnostic par interaction Homme-Robot - Téléopération

Le mode de fonctionnement **Choix HRI** dont les modules sont rappelés en table 9.7, doit donner les moyens à l'opérateur humain de résoudre un problème de diagnostic que le robot n'a pu solutionner de façon autonome. Dans le cadre de notre mission, l'opérateur peut éva-

luer le contexte d'évolution du robot en analysant l'ensemble des informations disponibles au niveau de l'architecture, ou engager une action de téléopération pour appréhender la situation réelle du robot et de son environnement. On peut remarquer qu'ici le module UDP, précédemment positionné dans l'architecture pour pourvoir aux besoins de la supervision, fait maintenant explicitement partie de la boucle de contrôle en permettant des transferts d'informations bidirectionnels.

Types de module	Sous-objectif : Choix HRI				
Contrôle	P3D	NAV	UDP		
Supervision					
Observation	SNS	ACT	OWF	RTC	MGO

Table 9.7 – Phase 5 : Les modules actifs pour le sous-objectif de Choix HRI

Dans le cas qui nous intéresse, l'opérateur doit répondre à la perte de localisation et à l'incapacité du robot à suivre le chemin fourni, et soit lui permettre de reprendre son fonctionnement d'une façon plus sûre. Dans notre cas, l'opérateur décide de faire appel à la **Téléopération** pour être capable d'explorer l'environnement du robot à l'aide des images provenant de la caméra embarquée. Il espère ainsi pouvoir déterminer la position réelle du robot et diagnostiquer l'origine du dysfonctionnement observé.

Pour commuter en téléopération, l'opérateur sélectionne le mode **Téléopération** (évènement **Choix Téléop**, figure 9.9). Cela se traduit par une demande de changement de sous-objectif directement reçue par le superviseur local. Celui-ci devra donc transférer cette demande vers les superviseurs d'adaptation et contextuel. Ce dernier détermine en fonction de l'état des modules le mode de fonctionnement le plus performant disponible (en mode téléopération) que le superviseur d'adaptation approprié mettra en œuvre. Puisque les sonars sont opérationnels, la téléopération avec évitement d'obstacles DVZ est choisie. La table 9.8 énumère les modules utilisés.

Types de module	Sous-objectif : téléopération					
Contrôle	P3D	ODO	NAV	UDP	DVZ	
Supervision						
Observation	SNS	ACT	OWF	BKF	RTC	MGO

Table 9.8 – Phase 5 : Les modules actifs pour la Téléopération avec DVZ

Durant la téléopération (courbe verte figure 9.10), l'opérateur va tenter de comprendre le problème lié à l'environnement en faisant faire un tour sur lui-même au robot. Il repère ainsi, grâce à la caméra, un obstacle imprévu qui barre complètement la route au robot (cet obstacle a été simulé en modifiant la carte de référence du module SIM). La défaillance observée venait donc de cette perturbation puisque la carte utilisée par le robot ne représente pas exactement l'environnement réel.

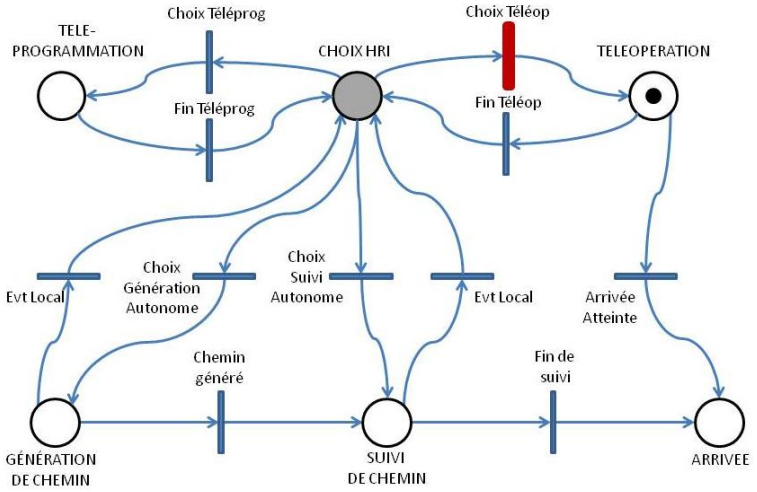


Figure 9.9 – Phase 5 : Modification du superviseur local au lancement de la téléopération

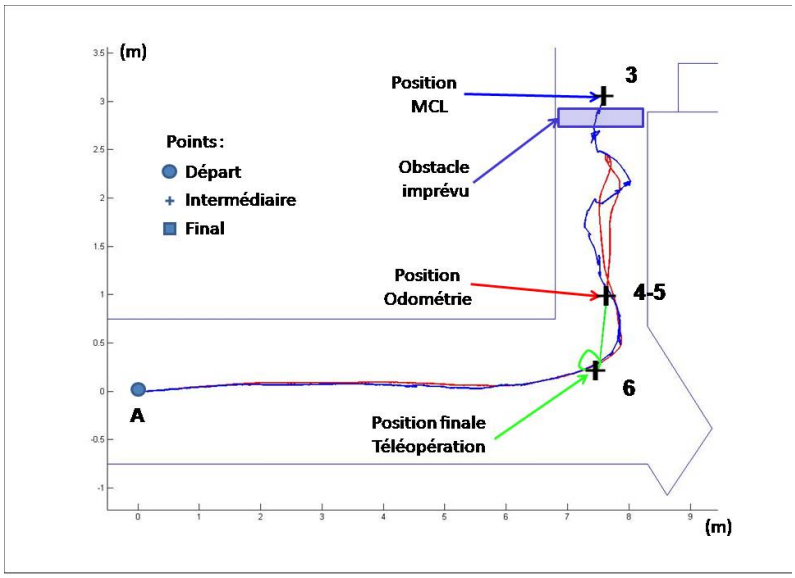


Figure 9.10 – Phase 5 : Zoom sur la faute de localisation

L'origine du dysfonctionnement étant maintenant déterminée une solution de recouvrement peut être engagée à l'aide de la téléprogrammation.

9.3.6 Phase 6 : Recouvrement par interaction Homme-Robot - Téléprogrammation

Après la téléopération (évènement **Fin Téléop**), l'opérateur décide d'utiliser successivement 3 modes de téléprogrammation pour :

– relocaliser le robot, au point 6, à partir de la localisation qui lui semble la plus vraisemblable.

– donner un nouveau chemin à suivre pour éviter le couloir obstrué par l'obstacle à partir du point 6.
– mettre à jour l'état des modules : Les modules et fonctions MCL, P3D-roue et SMZ préalablement déclarés fautifs lors des phases 3 et 4 sont déclarés sains par l'opérateur.

Ces 3 étapes de téléprogrammation se traduisent au sein du superviseur local par 3 exécutions de la boucle **Choix HRI** - **Choix Teleprog** - **Teleprogrammation** - **Fin Teleprog** sur la figure 9.11.

Une fois le robot relocalisé et le nouveau chemin mis à jour dans la mémoire interne, l'opérateur décide alors de terminer l'interaction avec le robot afin de lui permettre de reprendre sa mission en autonomie. Il adresse donc l'évènement **Choix Autonome Suivi**.

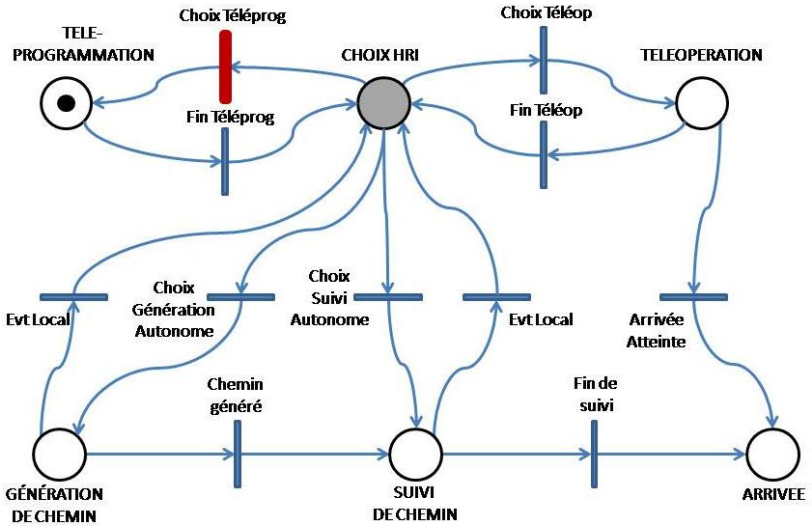


Figure 9.11 – Phase 6 : Modification du superviseur local pour répondre au diagnostic et proposer un recouvrement

Les phases 5 et 6 de la mission ont permis de conduire, avec l'aide de l'opérateur, les processus de diagnostic et de recouvrement. Les informations significatives correspondantes sont rassemblées dans la table 9.9.

Processus de tolérance	**Étapes**
Mode de fonctionnement	Téléopération DVZ et Téléprogrammation
Résidu	
Faute	Diagnostic humain : obstacle non représenté sur la carte
Module (Sévérité)	
Recouvrement	Relocalisation - Génération nouveau chemin - Mise à jour état modules

Table 9.9 – Phase 5 et 6 : Synthèse du processus de tolérance aux fautes de diagnostic et recouvrement par interaction Hommme-Robot.

A l'issue de la phase 6 de recouvrement, le superviseur contextuel devra préciser au superviseur d'adaptation concerné, pour le sous-objectif **Suivi de chemin** autonome, le mode de fonctionnement le plus performant pouvant être employé. Puisque l'ensemble des modules nécessaires sont opérationnels, le mode de fonctionnement optimal **Suivi de chemin autonome SMZ** est donc sélectionné.

9.3.7 Phase 7 : Détection d'une défaillance partielle des sonars

A partir du point 6, le robot est à nouveau capable de se déplacer dans les conditions les plus performantes. Sa progression est présentée dans la figure 9.12 et les modules exécutés dans ce mode de fonctionnement sont ceux mis en œuvre lors de la première phase (voir table 9.3).

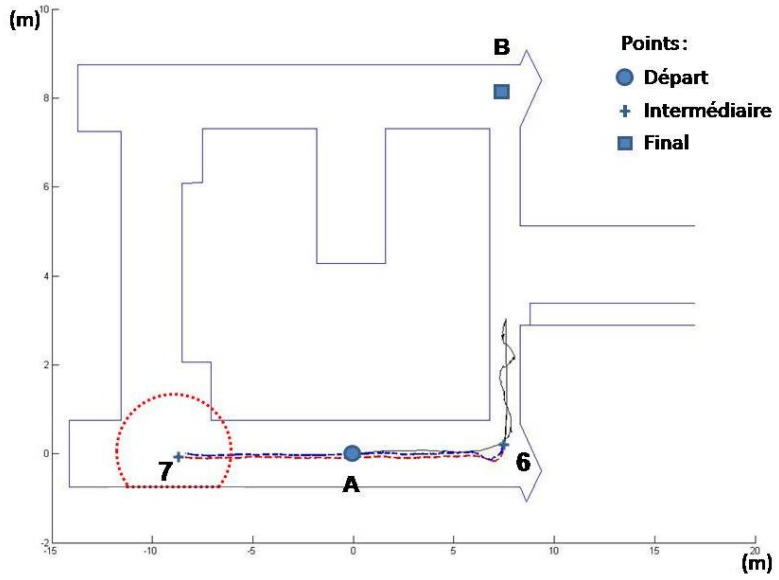


Figure 9.12 – Phase 7 : Trajectoire du robot

Sur la figure 9.12 on peut remarquer qu'au niveau du point 7, la portée maximale des capteurs sonars du robot ne permet pas de reconstruire fidèlement l'environnement du robot. En effet, à la fréquence d'acquisition utilisée lors de cette expérimentation (acquisition des 16 valeurs de sonars en 0,1s), la portée des sonars est réduite à une distance maximale de 1m50 autour du robot. Le résidu **USaveugle** est donc généré. L'analyse de signature permet de conclure à une faute unique **Sonar aveugle** engendrant une défaillance partielle de sévérité moyenne du module UST.

En réaction à la mise à jour de l'état des modules et à ce niveau de sévérité, le superviseur contextuel identifie le mode de fonctionnement suivi de chemin autonome SMZ sans MCL comme étant le plus performant. Le robot ne se basera plus que sur les données odométriques pour se localiser. En effet, le module MCL est inutilisable lorsque le module UST est défaillant. Par contre, les sonars peuvent toujours être utilisés pour la détection d'obstacle : SMZ reste opérationnel. Le superviseur contextuel adresse donc un évènement d'adaptation au superviseur

correspondant (figure 9.11).

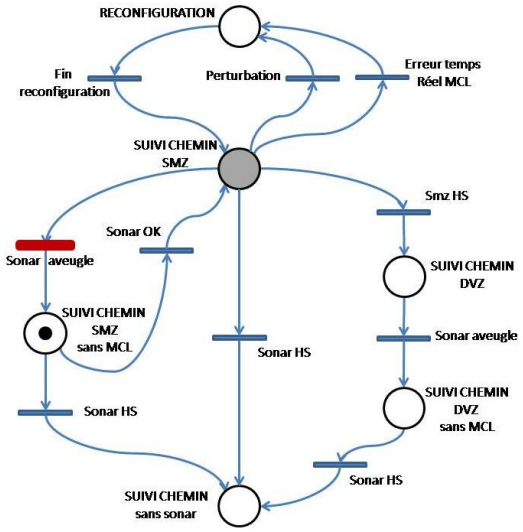


Figure 9.13 – Phase 7 : Evolution du superviseur d'adaptation

Le tableau 9.10 synthétise le processus de tolérance aux fautes réalisé lors de cette phase de la mission.

Processus de tolérance	**Étapes**
Mode de fonctionnement	Suivi de chemin autonome SMZ
Résidu	USaveugle
Faute	Sonar aveugle
Module (Sévérité)	UST (moyenne)
décision	plus d'utilisation de la localisation MCL

Table 9.10 – Phase 7 : Synthèse du processus de tolérance aux fautes

9.3.8 Phase 8 : Recouvrement de la perturbation partielle des sonars

Les modules présents dans le mode de fonctionnement suivi de chemin autonome SMZ sans MCL sont présentés dans la table 9.11. Les modules MCL et LOC (grisés), qui ne sont pas utilisés dans ce mode de fonctionnement, sont à ce stade déclarés non opérationnels. Le module UST est toujours présent car, malgré sa défaillance partielle, il reste assez performant pour permettre d'utiliser le comportement réactif associé à la fonctionnalité d'évitement d'obstacle.

La faute Sonar aveugle étant réversible, le résidu correspondant est toujours mis à jour par le module SNS qui est donc toujours exécuté. Cela pourra éventuellement permettre de rétablir un meilleur mode de fonctionnement si les sonars retrouvent leur pleine fonctionnalité.

Comme l'illustre la figure 9.14, c'est au point 8 de la trajectoire du robot que le résidu **USaveugle** disparait car un nombre assez important de sonars produit des informations significatives permettant de nouveau de représenter l'environnement. Cette situation n'a pas pu se produire

Types de module	suivi de chemin autonome SMZ sans MCL							
Contrôle	P3D	SIM	UST	ODO	MCL	NAV	SMZ	CTL
Supervision	UDP							
Observation	OSC	SNS	ACT	OWF	LOC	BKF	RTC	MGO

Table 9.11 – Phase 8 : Les modules actifs pour suivi de chemin autonome SMZ sans MCL

avant : le couloir entre le point 7 et le point 8 est plus large que celui parcouru au point 8 (attention, les échelles en abscisse et ordonnée ne sont pas les mêmes sur les figures).

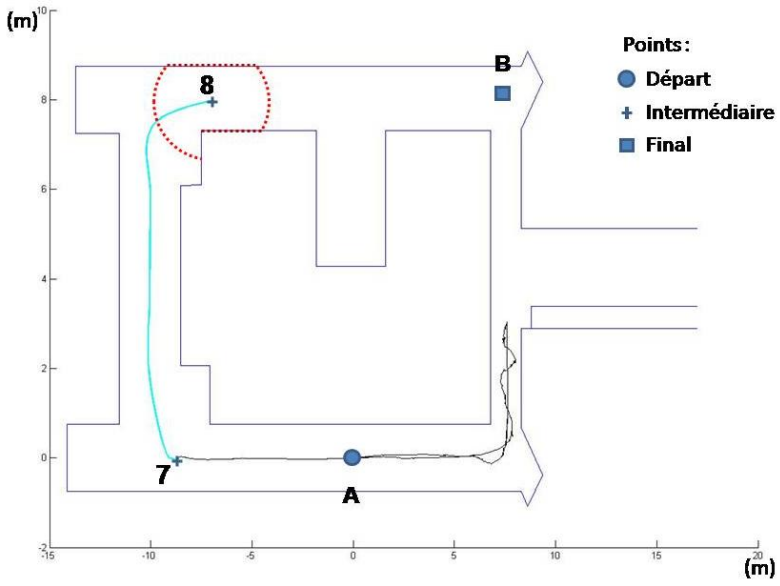


Figure 9.14 – Phase 8 : Déplacement du robot

La disparition du résidu **USaveugle** entraine le module de diagnostic à mettre à jour l'état des modules associés (MCL et LOC), ainsi qu'à signaler cette situation au superviseur contextuel en générant un évènement de sévérité spécifique, notée ∅. Le superviseur contextuel rétablit alors un mode de fonctionnement plus performant (ici le nominal) par le biais du superviseur d'adaptation (figure 9.15) en lui envoyant l'évènement **Sonar OK**.

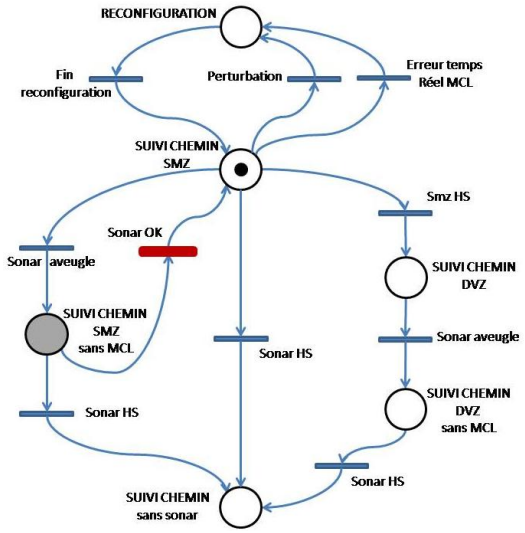


Figure 9.15 – Phase 8 : Evolution du superviseur d'adaptation

La table de synthèse 9.12 résume le processus de tolérance aux fautes mis en œuvre lors de la phase 8.

Processus de tolérance	**Étapes**
Mode de fonctionnement	Suivi de chemin autonome SMZ
Résidu	~~USaveugle~~ (le résidu repasse à 0)
Faute	
Module (Sévérité)	∅
décision	rétablissement du mode de fonctionnement optimal

Table 9.12 – Phase 8 : Synthèse du processus de tolérance aux fautes

Le mode de fonctionnement optimal, c'est-à-dire le suivi de chemin autonome SMZ utilisé lors de la première phase (table 9.3), est donc à nouveau sélectionné pour permettre la poursuite de la mission dans les meilleures condition de sureté et de performance.

9.3.9 Phase 9 : Détection d'une défaillance complète des sonars

Le robot continue à nouveau sa mission à partir du point 8 (figure 9.16).

Afin d'intégrer une défaillance complète d'un élément matériel au cours du déroulement de la mission, nous avons choisi d'en simuler l'occurrence au niveau de certains sonars (module SIM) en leur faisant retourner en permanence la même valeur.

Cette défaillance est détectée au point 9 à l'aide du résidu **UShors_service**. L'analyse de signature conclut à une faute unique **Sonar HS** qui d'après la table AMDEC perturbe de manière permanente la fonction P3D-sonar, et qui est associée à un niveau de sévérité moyen.

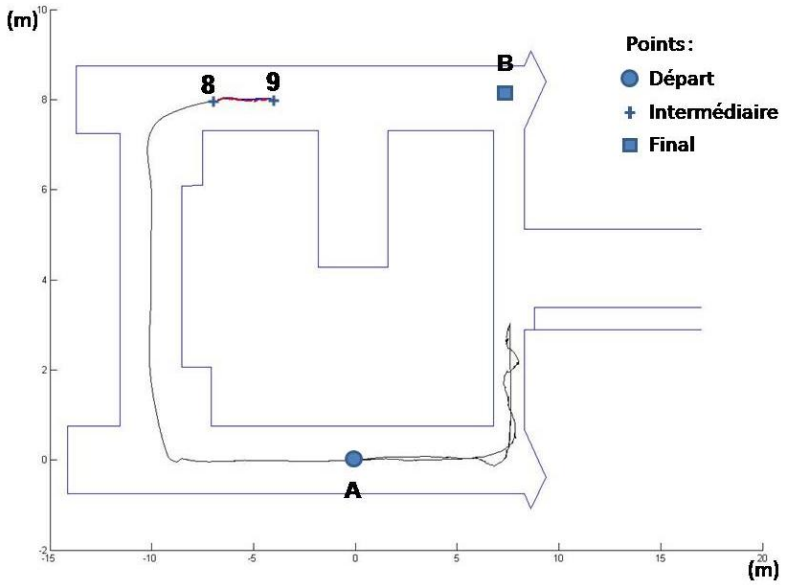


Figure 9.16 – Phase 9 : Déplacement du robot

La mise à jour de l'état des modules engendrée par ce diagnostic conduit à déclarer la fonction P3D-sonar comme non opérationnelle. Ceci, associé à la sévérité moyenne, conduit le superviseur contextuel à dégrader le mode de fonctionnement courant par le biais du superviseur d'adaptation (figure 9.17), qui choisira le **Suivi de chemin sans sonar**.

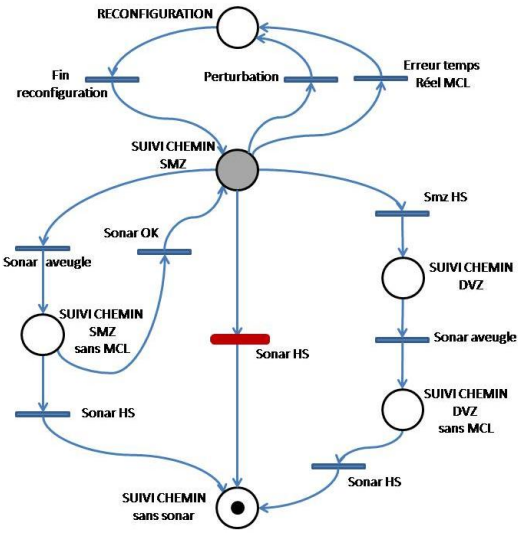


Figure 9.17 – Phase 9 : Evolution du superviseur d'adaptation

La table 9.13 synthétise le processus de tolérance aux fautes pour la phase 9.

Processus de tolérance	**Étapes**
Mode de fonctionnement	Suivi de chemin autonome SMZ
Résidu	UShors_service
Faute	US HS
Module (Sévérité)	P3D-sonar (moyenne)
décision	mode autonome dégradé : Suivi de chemin sans sonar

Table 9.13 – Phase 9 : Synthèse du processus de tolérance aux fautes

Le mode de fonctionnement **Suivi de chemin sans sonar** n'est pas le plus sécuritaire car le robot fonctionne en « aveugle » et risque à tout moment de percuter un obstacle ou un mur si la localisation odométrique dérive. Cependant, compte tenu de la vitesse et de l'inertie du robot, les risques engendrés ont été considérés comme acceptables pour lui et son environnement.

9.3.10 Phase 10 : Détection de collision

Dans la table 9.14 sont présentés les modules réalisant le Suivi de chemin autonome sans sonar. Ce mode de fonctionnement aveugle a été choisi en dernier recours car il n'a aucun moyen d'évitement d'obstacle. En grisé sont indiqués les modules qui ne peuvent être utilisés en raison de la défaillance complète des sonars.

Types de module	Sous-objectif : suivi de chemin dégradé sans sonar								
Contrôle	P3D	ODO	SIM	UST	MCL	NAV	GUI	SMZ	CTL
Supervision	UDP								
Observation	OSC	SNS	ACT	OWF	LOC	BKF	RTC	MGO	

Table 9.14 – Phase 10 : Les modules actifs pour le Suivi de chemin sans sonar

Enfin, nous avons souhaité simuler la collision du robot avec un obstacle. Nous avons donc appuyé, au point 10, sur un des bumpers du robot 9.18.

L'analyse des signatures permet alors, grâce au résidu **bumper**, de diagnostiquer la faute unique **Param-choc** du module GUI. Celle-ci en référence à la table AMDEC du suivi de chemin autonome sans sonar permet de déterminer une défaillance partielle de sévérité grave. En effet, cette faute peut refléter différentes situations auquel le robot ne pourra répondre de façon autonome, comme par exemple un obstacle imprévu, une mauvaise localisation sur la carte. Il est donc nécessaire à ce stade d'interpeler un opérateur humain (par le biais d'un évènement de type local **EvtLocal** pour le passage en état **ChoixHRI**, figure 9.19) pour permettre la poursuite de la mission.

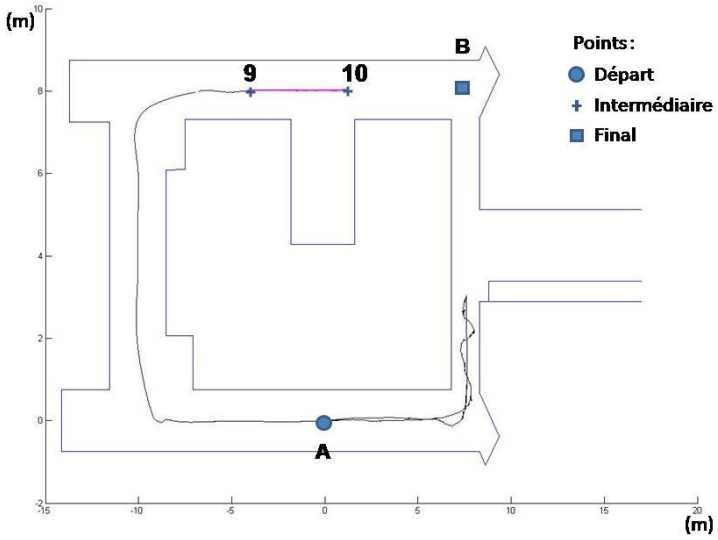


Figure 9.18 – Phase 10 : Déplacement du robot

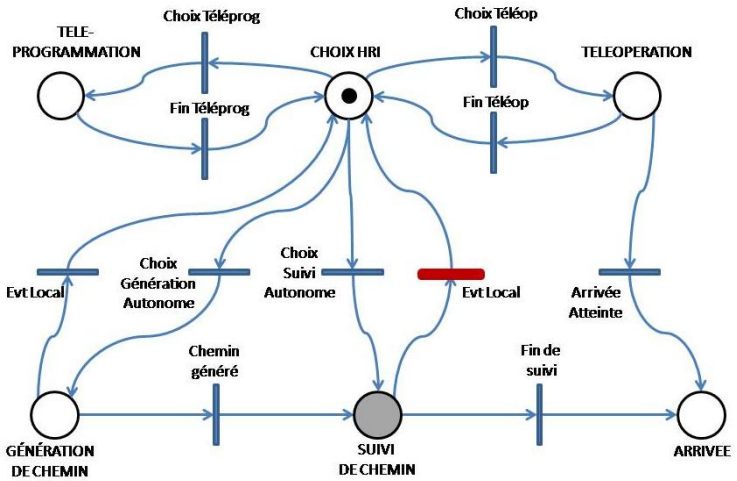


Figure 9.19 – Phase 10 : Evolution du superviseur local pour répondre au besoin d'interaction Homme-robot.

La table 9.15 résume le processus de tolérance aux fautes mis en œuvre lors de la phase 10.

Processus de tolérance	**Étapes**
Mode de fonctionnement	Suivi de chemin autonome sans sonar
Résidu	bumper
Faute	Param-choc
Module (Sévérité)	GUI (grave)
Recouvrement	choix HRI

Table 9.15 – Phase 10 : Synthèse du processus de tolérance aux fautes

L'opérateur est appelé en dernier recours pour tenter d'achever la mission malgré la perte d'une partie des capteurs extéroceptifs du robot.

9.3.11 Phase 11 : Conclusion de la mission par Interaction Homme-Robot : Téléopération

Dans le cadre d'un dysfonctionnement majeur et irréversible du robot, peu de modules restent utilisables, seule la téléopération la plus élémentaire, sans évitement d'obstacle, demeure possible. L'opérateur choisi donc de basculer en téléopération et lors de la sélection de ce sous-objectif le superviseur contextuel sélectionne le mode de fonctionnement Téléopération sans DVZ (figure 9.20).

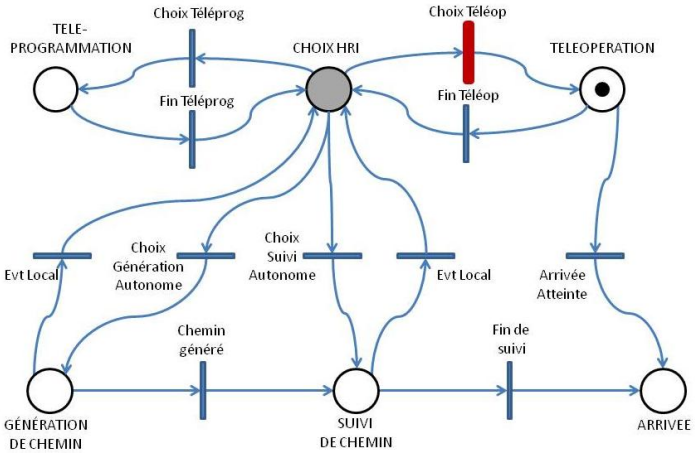


Figure 9.20 – Phase 11 : Modification du superviseur local au lancement de la téléopération

Le téléopérateur conduit le robot du point 10 au 11 (figure 9.21) qui correspond à la position de livraison attendue.

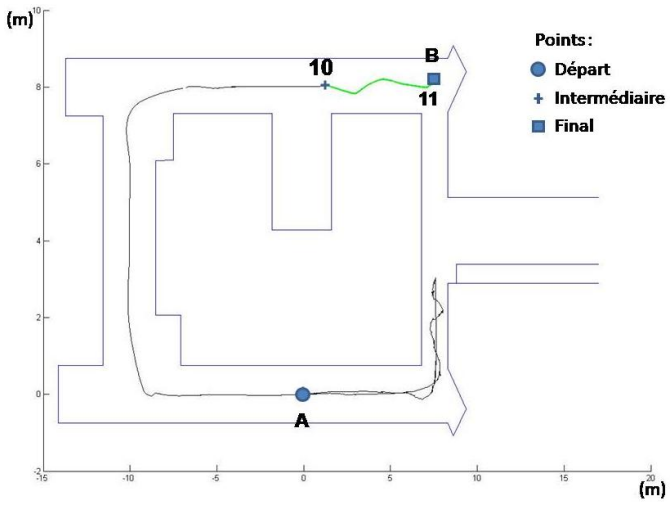


Figure 9.21 – Phase 11 : Déplacement du robot

L'objectif déplacement de la mission de livraison du courrier est donc achevé lors de l'émission de l'évènement **Arrivée Atteinte**.

9.4 Synthèse et analyse

9.4.1 Vue globale du déroulement de la mission

La table 9.16 synthétise le déroulement de la mission de livraison de courrier, et plus particulièrement de l'objectif suivi de chemin, qui vient d'être détaillé. Le déplacement du robot a duré un peu plus de 4 minutes (4mn11s) et environ 55 mètres ont été parcourus ce qui correspond à une vitesse de déplacement moyenne de 0.22 m/s.

Phases		**Tolérance aux fautes**			
n°	**Date**	**Détection**	**Diagnostic**	**Sévérité**	**Recouvrement**
1	0	**Début**			
2	5	Retard MCL	Param-Nbparticules	faible	Adaptation
3	52	Dérivée et Écart localisation	Perturbation ou Sol glissant	faible	Adaptation
4	55	Retour en arrière	Perturbation forte	grave	Local
5	56				*HRI : Téléopération avec DVZ*
6	86				*HRI : Téléprogrammation*
7	151	US aveugle	Sonar Aveugle	moyenne	Adaptation
8	191	*US aveugle*	∅	moyenne	Adaptation
9	201	US hors-service	US HS	moyenne	Adaptation
10	226	Bumper	Param-choc	grave	Local
11	251	**But atteint**			*HRI : Téléopération sans DVZ*

Table 9.16 – Synthèse de l'évolution de la mission

Tout au long de ce périple, les nombreux évènements perturbateurs observés ont permis d'analyser le comportement des mécanismes de tolérance aux fautes au sein de l'architecture construite. L'occurrence de ces évènements a été soit volontairement provoquée logiciellement (obstacle imprévu, perte des sonars) ou physiquement sur le robot (contact bumper), soit induite par les limitations des capteurs (sonars aveugle). Les défaillances détectées ont conduit aux diagnostic de fautes uniques (paramètre nb_de_particules, sonars aveugles et sonars HS, choc bumper), ou multiples (perturbation de la localisation / sol glissant). Lorsque le diagnostic a convergé vers une faute pertinente de sévérité moyenne ou faible, les actions de recouvrement ont été engagées de façon autonome vers le mode de fonctionnement le plus performant possible. Dans le cas contraire, l'opérateur a été interpelé (informations en italique dans la table)

pour palier aux limitations décisionnelles embarquées. Enfin, il faut souligner qu'au cours de la mission, le rétablissement de certaines fonctionnalités suite à des fautes temporaires (sonars, localisation) ont permis de retrouver, au fil de l'eau, des modes de fonctionnement ayant de meilleur niveaux de performance.

Même si cette mission a été conduite à son terme, il faut maintenant souligner l'importance que revêt le mécanisme d'interaction Homme-Robot dans son succès.

9.4.2 Importance de l'Interaction Homme-Robot

La figure 9.22 décrit, sous forme graphique, l'évolution du niveau d'autonomie en fonction des points significatifs de la mission et de leur date d'occurrence.

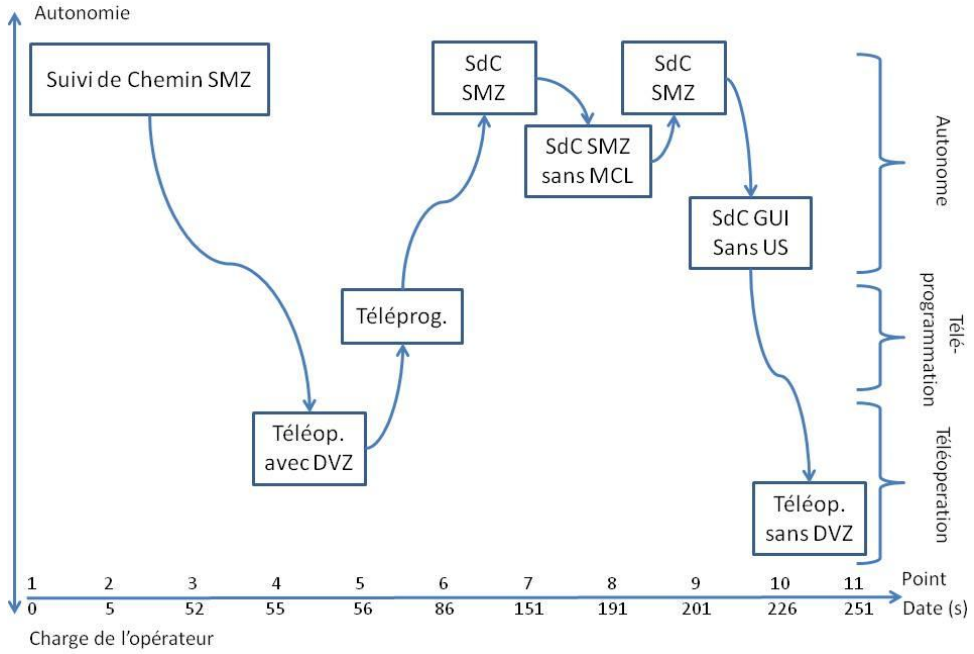


Figure 9.22 – Adaptation et évolution des modes de fonctionnement au cours de la mission

Au cours de cette mission trois niveaux d'autonomie ont été utilisés : les modes autonome, teleprogrammation et téléopération. Au regard de la durée de chacun d'entre eux, on peut constater que le comportement autonome n'est assuré que pendant la moitié de la mission (51.8% - 2mn10s) et que l'interaction avec l'opérateur est nécessaire avec une répartition pratiquement symétrique en téléopération (21.9% - 55s)) et téléprogrammation (26.3% - 1mn6s).

En outre, grâce aux mécanismes de redondance logicielle, le mode autonome est en fait décomposé en plusieurs modes de fonctionnement dont trois ont été utilisés durant cette expérimentation : le plus performant (suivi de chemin SMZ), et deux dégradés (l'un sans localisation de Monte-Carlo, l'autre ajoutant la perte de la capacité d'évitement d'obstacle).

Pour limiter les appels à l'homme et ne pas surcharger l'opérateur ce dernier n'est interpelé qu'en cas de dysfonctionnements graves, si le processus de diagnostic embarqué n'aboutit

pas ou si le mécanisme de recouvrement autonome ne trouve pas de mode de fonctionnement opérationnel. Même si l'on peut déplorer que pendant près de la moitié du temps de la mission l'opérateur a été mobilisé, il faut souligner que sans son aide celle-ci aurait été irrémédiablement abandonnée au point 3, au bout de 55s et d'environ 10 m seulement, avec le robot perdu dans les couloirs. On peut remarquer que plusieurs fautes graves ont été intentionnellement injectées au cours de la mission, provoquant ainsi l'intervention de l'homme. De plus, parmi les nombreux processus de recouvrement exécutés, 66,7% (4 sur 6) ont été effectués de façon autonome. Les capacités cognitives de l'opérateur ont permis d'appréhender et de traiter ces situations d'exception en permettant, le plus souvent localement et de façon limitée dans le temps, de rétablir le robot dans un état cohérent pour que la mission puisse se poursuivre.

Les mécanismes d'interaction Homme-Robot mis en œuvre relèvent, à notre sens, du concept d'autonomie ajustable adaptative puisque c'est le robot qui initie l'ajustement du niveau d'autonomie. Au mode autonome, s'ajoutent d'une part la téléprogrammation qui peut s'apparenter à de l'échange de contrôle où l'opérateur intervient comme une ressource cognitive supplémentaire, et d'autre part la téléopération qui relève de la commande déportée où l'Homme assure à distance un contrôle complet du robot.

9.5 Conclusion

Ce chapitre a détaillé le comportement de l'architecture tolérantes aux fautes construite tout au long de ce document dans le cadre d'une mission de livraison de courrier. L'expérimentation, qui a malheureusement été réalisée « Hardware in the Loop » en raison de limitations capteurs, a permis d'évaluer les capacités de détection, de diagnostic et de recouvrement de défaillances en suivant un scénario savamment orchestré pour illustrer le comportement de l'architecture face à la plupart des types de situations adverses préalablement identifiées.

Malgré tous les dysfonctionnements induits, la mission a été réalisée avec succès en mettant en oeuvre les réponses les plus adaptées et performantes compte tenu des capacités fonctionnelles opérationnelles et des capacités décisionnelles de l'architecture. En l'absence de solutions embarquées, l'opérateur distant a été impliqué dans le processus de recouvrement pour pallier ces limitations.

Certes cette expérimentation ne constitue pas une évaluation rigoureuse des mécanismes de tolérance aux fautes qu'il conviendrait d'analyser en profondeur tout au long de nombreuses missions. Cela permettrait d'évaluer réellement, et dans la durée, la pertinence des dysfonctionnements identifiés et la « robustesse » des mécanismes de recouvrement mis en place. Par ailleurs les conditions expérimentales mises en oeuvre ne nous permettent pas ici d'évaluer objectivement la charge de travail opérateur induite par le processus de tolérance aux fautes, et qui devrait notablement décroitre lors d'utilisations réelles du robot.

Conclusion Générale

Tout système, qu'il soit vivant ou artificiel est, au cours de son cycle de vie, en proie à des défaillances d'origine et de magnitude très variées. Pour y faire face, les organismes déploient un ensemble de réponses adaptées leur permettant, autant que faire ce peut, d'assurer la viabilité et la pérennité de l'entité affectée. Les dispositifs créés par l'Homme ne peuvent déroger à ce principe sous peine d'être très rapidement hors de service. C'est pourquoi, dans certains domaines, lorsque la sécurité est critique (avionique, nucléaire) ou lorsque la continuité de service est cruciale (système de production), un ensemble de mécanismes dédiés, automatisés ou impliquant l'analyse humaine, ont été mis en place pour contrôler le comportement du système y compris en présence de défaillances. Cependant, force est de constater que la robotique mobile ne propose pas actuellement de dispositif suffisamment fiable pour pouvoir envisager, à court terme, une évolution réellement autonome pour des applications critiques dans un environnement dynamique [Dariot, 2010, ISTAG, 2009].

Le développement de systèmes tolérants aux fautes pour la robotique mobile autonome a donc constitué la problématique centrale de notre étude que nous avons abordée sous sa dimension architecturale logicielle. Plus globalement, cette thématique a été identifiée, de par ses enjeux tant technologiques que socio-économiques, comme étant un des verrous scientifiques majeurs nécessaire au déploiement de la sûreté de fonctionnement au sein de systèmes matériels et logiciels [Basseville et al., 2007, ISTAG, 2009].

La première partie de ce document a permis d'effectuer un large tour d'horizon des approches développées pour améliorer la sûreté de fonctionnement des systèmes contrôlés en portant une attention particulière aux techniques de tolérance aux fautes. Celles-ci nécessitent le déploiement d'une démarche classiquement décomposée en trois phases : la détection et le diagnostic de la faute, puis le recouvrement pour y faire face. Dans un premier temps, l'étude bibliographique réalisée a démontré le foisonnement de travaux relevant de cette thématique en particulier pour les techniques de détection et de diagnostic de faute. Les techniques de recouvrement quand à elles cherchent à synthétiser un contrôleur robuste vis-à-vis de certaines pannes ou pouvant s'adapter à leur occurrence. A ces démarches classiques relevant de l'automatique nous avons ajouté des techniques impliquant l'entité humaine dans la boucle de contrôle et modifiant donc le niveau d'autonomie du système. Dans un second temps nous avons focalisé notre étude sur le domaine de la robotique du point de vue des architectures logicielles [Durand et al., 2010b]. Bien qu'il existe de nombreux travaux relevant des thématiques de dé-

tection et diagnostic de fautes ou de recouvrement, l'analyse conduite a mis en évidence un fort cloisonnement de ces travaux et l'absence d'une approche complète de mise en œuvre au sein des architectures de contrôle. Cette analyse a donc permis de constater qu'il n'existe pas actuellement de démarche globale, transversale, structurée et ouverte permettant de concevoir et de déployer une architecture de contrôle tolérante aux fautes.

La seconde partie présente notre réponse aux manques identifiés en détaillant la démarche élaborée [Durand et al., 2010a]. Celle-ci s'ancre en premier lieu sur l'adaptation de la démarche AMDEC au contexte des architectures de contrôle robotique. Pour chaque mode de fonctionnement, cette étape cruciale permet d'identifier et de caractériser tant temporellement qu'au niveau de leur sévérité, les fautes pertinentes qui doivent être considérées ainsi que les moyens d'observation à adopter. Ce chapitre détaille dans un second temps comment les mécanismes de détection, de diagnostic et de recouvrement sont pris en compte dans la démarche proposée. La notion de module d'observation est introduite pour permettre, au sein de l'architecture déployée, de détecter l'occurrence d'une défaillance. La phase de diagnostic s'appuie largement sur le concept de résidus et sur la matrice des signatures des fautes associée. Le tableau de synthèse AMDEC est encore utilisé pour déterminer la sévérité des fautes diagnostiquées. Enfin la phase de recouvrement s'appuie sur le déploiement d'une réponse adaptée en ajustant dynamiquement le mode de fonctionnement du robot et potentiellement son niveau d'autonomie en fonction de la sévérité de la faute et des fonctionnalités opérationnelles du robot.

La troisième partie a présenté le contexte applicatif et expérimental qui a supporté l'évaluation de la démarche proposée. Il détaille dans un premier temps les mécanismes du middleware COTAMA utilisé pour déployer l'architecture temps réel tolérante aux fautes. Nous retiendrons ici l'importance de la notion de modularité, de la définition de règles nécessaires à la décomposition de la mission en objectifs et sous-objectifs, de superviseurs pour supporter le niveau décisionnel et enfin d'ordonnanceur qui permet une maitrise fine et précise du niveau exécutif de l'architecture. Dans un second temps le contexte expérimental a été précisé. Une mission de livraison de courrier robotisée entre des bureaux du laboratoire a été choisie. L'organisation architecturale globale comprend un ordinateur de supervision distant coordonnant les échanges d'informations entre les différents acteurs humains et un ou plusieurs robots Pioneer 3-DX utilisés. Enfin dans une dernière partie, la décomposition de la mission en objectifs et sous-objectifs ainsi que les principaux modes de fonctionnement sont présentés. En raison de son rôle central dans l'exécution de la mission, le déploiement du suivi de chemin est détaillé en effectuant une présentation approfondie des différents modules et algorithmes mis en œuvre.

La quatrième partie expose comment les différentes phases de notre démarche de conception d'une architecture tolérante aux fautes ont été implémentées pour la mission retenue. Pour cela, faute de place, il se focalise encore sur la mise en œuvre du suivi de chemin. On y trouve le tableau de synthèse AMDEC relatif aux différentes fautes pertinentes considérées. Il détaille l'ensemble des modules et algorithmes d'observation utilisés pour la détection [Durand et al., 2009] ainsi que la matrice des signatures construite. Enfin et surtout il montre comment les méca-

nismes nécessaires à la tolérance aux fautes ont été adossés à l'architecture de contrôle préexistante détaillée dans le chapitre précédent [Durand et al., 2010a]. Pour ce faire un module global d'observation a été introduit pour centraliser l'ensemble des données issues des modules d'observation impliquées dans le mode de fonctionnement courant, et mettre à jour l'état des modules de l'architecture. Ces informations permettent alors à un superviseur contextuel de préparer la phase de recouvrement en identifiant la réaction la plus adaptée à mettre en œuvre compte tenu de la sévérité de la faute et des modules opérationnels. En fonction du diagnostic réalisé il produit des évènements typés qui permettent au superviseur correspondant d'engager l'action de recouvrement choisie.

Enfin la dernière partie dissèque le déroulement d'une des missions effectuées « Hardware In the Loop » par notre robot Pioneer 3-DX [Durand et al., 2010b]. Tout au long de celle-ci des évènements perturbateurs engendrant des fautes de sévérité plus ou moins élevée ont été volontairement introduits ou simulés de façon à étudier le comportement de l'architecture tolérante aux fautes construite. Pour chacune de leurs occurrences une analyse détaillée des mécanismes mis en œuvre et du comportement de l'architecture a été réalisée. Enfin, plus globalement le déroulement de l'ensemble de la mission a aussi été commenté de façon à démontrer la capacité d'adaptation et de réactivité de l'architecture de contrôle ainsi que l'évolution des interactions Homme-Robot mises en œuvre pour permettre d'achever avec succès la mission entamée [Durand et al., 2010a].

Compte tenu des multiples domaines d'études abordés pour développer notre démarche de conception d'architecture de contrôle tolérante aux fautes de nombreuses perspectives scientifiques et expérimentales peuvent être envisagées. Celles ci relèvent autant de l'amélioration ou de l'enrichissement de l'approche proposée que de l'évaluation en situation de son efficacité.

Tout d'abord commençons par évoquer comment pourrait être amélioré le processus de détection et de diagnostic de fautes actuellement mis en place et dont dépend largement la pertinence de la phase de recouvrement. Le système d'insertion de modules d'observation proposé nous semble satisfaisant pour traiter correctement l'occurrence d'une défaillance unique. Il pourrait être complété, dans certaines situations, par des actions spécifiques du robot permettant de s'assurer que le capteur (actionneur) est réellement fautif. Par exemple, dans le cas de notre robot Pioneer 3-DX, un ultrason renvoie systématiquement la même valeur lorsqu'aucun obstacle n'est détecté à moins de 1,5 mètres. Ainsi, dans cette situation, courante lorsque le robot se déplace dans un long couloir rectiligne, il est impossible de savoir s'il est défaillant ou non. Afin de lever cette incertitude, il est envisageable de lui faire effectuer un tour sur lui même, de façon à vérifier que les sonars incriminés sont toujours opérationnels. Évidement, l'introduction de tels mécanismes autonomes de diagnostic actif, impacterait le nombre de modes de fonctionnement à considérer au sein de l'architecture. Ce type d'approche a déjà été abordé en robotique autonome embarquée dans le cadre de la planification de mission [Chantery and Pendolé, 2009].

L'une des limitations de notre proposition est le traitement du diagnostic en présence de défaillances multiples qui reste une problématique difficile dans les systèmes complexes

[Daigle et al., 2006]. Lorsque celles-ci sont corrélées le nombre de modules perturbés par l'incidence de la faute originelle peut venir brouiller le diagnostic réalisé. Actuellement cette situation est résolue par une analyse préalable de la chaine des défaillances. Nous pourrions envisager d'utiliser une approche plus structurée afin de construire un ensemble de résidus permettant de caractériser sans ambigüité l'origine de la défaillance observée [Coquenpot, 2004].

Même si nous sommes persuadés que la technique d'analyse de signature préconisée est adaptée à la complexité d'un système robotique autonome mobile, et aux contraintes temps réel imposées, il n'est évidemment pas exclu de comparer son efficacité avec des approches qualitatives. Par exemple en robotique, dans le cas de faute unique, Hamilton [Hamilton et al., 2001] a recours à des réseaux sémantiques pour modéliser l'ensemble des connaissances hétérogènes, relevant de l'espace des fautes, de diagnostic, et d'observation, nécessaires à la localisation de l'origine d'un dysfonctionnement. L'isolation qualitative de fautes multiples est abordée dans [Daigle et al., 2006] en utilisant une technique faisant appel à des graphes causaux temporels et à la génération de signature de faute.

La phase de recouvrement reste aussi largement perfectible et cela à différents niveaux.

D'une part, il faut remarquer que le mécanisme de recouvrement mis en place dans notre approche intervient exclusivement sur le mode de fonctionnement du robot associé à la tâche courante qui lui est assignée. Il ne faut pas ignorer que certains travaux relevant de la tolérance aux fautes en robotique mobile engagent aussi des actions de recouvrement induisant une replanification de la mission pour s'affranchir des fautes détectées [Goldman et al., 1997, Lussier, 2007, Chantery and Pendolé, 2009]. En ce sens, il nous semble que le travail que nous avons réalisé est complémentaire de ces approches de plus haut niveau et qu'il serait souhaitable de les faire coexister au sein d'une même architecture. Évidemment, cela complexifierait de fait le processus décisionnel puisqu'il faudrait arbitrer pour choisir le niveau du mécanisme de recouvrement à mettre en œuvre.

D'autre part il semble indispensable de nous attarder aussi sur l'interaction Homme-Robot qui occupe une place importante dans la démarche proposée puisqu'elle permet d'assurer, lorsque cela est possible, une redondance fonctionnelle indispensable au succès de la mission. Elle autorise également un diagnostic cognitif distant hors d'atteinte des capacités décisionnelles embarquées. De toute évidence, bien qu'utile dans le cadre d'une mission autonome, l'interaction entre l'Homme et le robot doit être à terme limitée.

L'utilisation de la redondance fonctionnelle humaine peut évidemment être lourde à gérer lorsque le manager de mission se retrouve en première ligne pour faire face à une succession de défauts qui frappent une flottille de robots. La charge de travail induite par la gestion de situation de crise peut être parfois trop importante pour l'Homme. Mais il nous semble cependant possible de tirer avantage de systèmes multi-robots en réseaux (networked robots) coopératifs, constitués d'un ensemble hétérogène de robots aux aptitudes partiellement redondantes. Dans ce cadre opérationnel il est sans doute possible de mettre en place des mécanismes adaptatifs au niveau de la flottille de robots permettant de restituer certaines des fonctionnalités disparues

par un robot, en utilisant celles encore opérationnelles d'un ou de plusieurs de ses compagnons. Une telle démarche développerait des comportements de groupe relevant de l'aphorisme « de l'aveugle et du paralytique ». Il serait ainsi possible d'introduire entre le robot et l'Homme, une couche comportementale tampon, permettant à la mission de se poursuivre, certes en mode dégradé, mais en ayant pris collectivement en charge la défaillance partielle d'un ou plusieurs robots, limitant par la même la charge de travail de l'Homme. Le rétablissement de certaines des fonctionnalités d'un ou plusieurs robots sera donc réalisé en partageant les moyens de perception extéroceptifs, les capacités d'action, les moyens de calcul d'autres robots partenaires. La notion de partage impose évidemment de disposer de capacités de communication entre les entités de la flottille et de pouvoir constituer des formations permettant de palier les dysfonctionnements rencontrés. Cette approche de la gestion de la tolérance aux fautes des systèmes multi-robots semble adopter un point de vue original s'écartant des démarches classiques [Parker, 1996, Chand and Carnegie, 2007] qui cherchent plutôt à trouver, en présence de robots fautifs, une ré-allocation des tâches compatible avec l'objectif de la mission.

Pour limiter dans le temps l'intervention humaine dans le cadre d'un diagnostic cognitif distant il est possible d'envisager mettre en place des mécanismes d'apprentissage embarqués. L'objectif ici est de construire et d'enrichir dynamiquement, lorsque cela est possible, une base de données permettant de mémoriser les solutions apportées par l'opérateur pour faire face à un dysfonctionnement qui n'a pu être traité initialement de façon autonome. On peut imaginer un mécanisme évolutif et adaptatif tirant avantage des heures de fonctionnement du robot et des compétences des opérateurs impliqués lors de la gestion de dysfonctionnements. Le processus d'apprentissage pourrait se décomposer en plusieurs étapes. Tout d'abord évaluer de la criticité de la configuration fautive observée et de la pertinence de la (des) solution(s) proposées. Puis en fonctionnement autonome, ne retenir que celle ayant permis d'obtenir au taux de réussite suffisamment élevé. Évidemment une telle démarche suppose être capable d'inférer et de créer dynamiquement les nouveaux sous-objectifs et modes de fonctionnement que l'opérateur aura mis en œuvre.

Enfin, toujours dans le cadre des interactions Homme-Robot, bien que les travaux développés dans ce manuscrit mettent en place des principes d'autonomie ajustable et permettent au Manager de Mission de prendre à tout moment le contrôle d'un robot, il est évident que le sens d'interaction privilégié est plutôt du Robot vers l'Homme que le contraire. Il est donc souhaitable d'enrichir le comportement de l'architecture de façon à ce que le robot puisse à nouveau adapter son niveau d'autonomie, non pas en fonction des difficultés qu'il rencontre, mais en fonction des situations adverses auxquelles l'Homme doit aussi faire face. En effet les travaux de Carlson [Carlson and Murphy, 2005] démontrent clairement qu'une des raisons des défaillances des robots mobiles provient d'erreurs réalisées par l'opérateur lui-même. Ainsi, tout comme notre approche préconise d'intégrer des observateurs dans l'architecture pour surveiller l'occurrence de défaillances, il faudrait aussi développer des mécanismes de scrutation permettant, lorsque l'opérateur est intégré dans la boucle de contrôle, d'analyser la cohérence du niveau d'autono-

mie ou de l'action déployée avec le contexte environnemental. En effet, il est parfois souhaitable que le robot décide unilatéralement de changer de mode de fonctionnement afin de préserver son intégrité ou celle de son environnement. Comme dans [Baker and Yanco, 2004], on peut imaginer des réactions de sauvegarde ou de suggestions.

Abordons maintenant la dimension expérimentale que doit adresser tout projet relevant de la robotique mobile. En premier lieu il faut pallier aux nombreuses difficultés de mise en œuvre qui ont été rencontrées en utilisant les concepts architecturaux de COTAMA. D'une part cette architecture, issue d'un travail de doctorat, n'est pas encore totalement stable. D'autre part elle souffre encore d'un ensemble de limitations qui ont fortement pénalisé la pleine évaluation de notre démarche. En effet, dans COTAMA, le partage de données entre sous objectifs est difficile, le lancement d'un sous objectif ne peut relever d'un évènement issu d'un module mais exclusivement d'évènements liés à la durée ou à la fin de sous objectifs, enfin il n'est pas possible d'enchainer les sous objectifs dans l'ordre que l'on veut. A ces limitations conceptuelles s'ajoute l'absence d'un environnement de programmation ergonomique. C'est pourquoi nous travaillons actuellement sur le portage de notre travail [Chauve, 2010] en nous appuyant sur CONTRACT [Passama, 2010], méthodologie de conception et développement d'architecture de contrôle temps réel robotique. Elle fait suite aux travaux réalisés dans le cadre de COTAMA et apporte des réponses aux différentes limitations que nous venons d'identifier.

Il est aussi indispensable d'achever le développement de l'ensemble de la structure architecturale devant supporter la supervision de la mission robotisée ainsi que les interactions des différentes classes d'opérateurs (manager, téléopérateur, utilisateur, etc.). Actuellement, au niveau de l'ordinateur de supervision, les différentes boites aux lettres et serveurs multi-clients ont été développés. Il reste cependant, et c'est une tâche importante, à définir la nature et la composition de l'ensemble des messages devant être échangés ainsi que toutes les interfaces Homme-machine qui permettront aux opérateurs, en fonction de leur rôle, d'interagir avec les robots. Évidemment, le développement d'environnements de travail sophistiqués facilitant, pour l'opérateur, la perception de l'environnement du robot, son interprétation et sa projection dans un état futur proche, dépasse le cadre de notre étude. On peut cependant remarquer que l'architecture déployée dans le cadre de notre travail devrait à terme tout de même permettre de visualiser l'environnement du robot, de repérer sa position sur une carte, de connaitre son état interne, son niveau d'autonomie et la tâche exécutée, ainsi que d'identifier les erreurs rencontrées.

Une fois le développement de cette plate-forme robotique d'expérimentation finalisé, une évaluation approfondie et en conditions réelles des architectures de contrôle tolérantes aux fautes construites devra être envisagée. Elle pourra exploiter pleinement la flexibilité de l'approche architecturale modulaire retenue en disposant d'un ensemble de modules sur étagère mettant à disposition des fonctionnalités éprouvées et pouvant être facilement enrichi (modules de vision, de commande de bras, etc..). Evidemment comme dans de nombreux articles dont ceux de Carlson [Carlson and Murphy, 2005] nous devrons effectuer des mesures de disponibi-

lité du robot, de MTBF ou de MTTR. Mais nous pourrons aussi analyser les fautes observées, leurs fréquences et la charge de travail des différents opérateurs. Comme dans [Berman et al., 2009] nous pourrons aussi ajouter l'identification des situations de blocage et une analyse de la performance des modules robotiques impliqués (navigation, guidage, etc.). Des travaux tels que ceux d'Hamilton [Hamilton et al., 1996] ont proposé une métrique visant à estimer la tolérance aux fautes des robots. Par ailleurs, notre démarche impliquant potentiellement l'intervention humaine, il est aussi possible, comme dans [Freedy et al., 2007] d'envisager mesurer l'efficacité de l'interaction Homme-Robot.

Ce travail, au confluent de la robotique et de l'informatique industrielle, a permis une collaboration au sein de l'équipe NERO (Networked Robot) du département Robotique du LIRMM en associant à la fois des compétences relevant du domaine de l'automatique proprement dit, de celui des architectures de contrôle et de celui de la sûreté de fonctionnement. Malgré ses limitations actuelles, il devrait constituer un excellent point de départ pour l'élaboration, à court terme, d'un démonstrateur réellement ouvert et opérationnel. Il permettra aussi d'engager une réflexion encore plus approfondie sur le développement des architectures de contrôle tolérantes aux fautes qui constitue, à plus long terme, l'un des défis majeurs que les scientifiques ont à relever pour développer des systèmes autonomes évoluant de façon performante et en toute sécurité dans des environnements dynamiques inconnus.

Bibliographie

[Adams et al., 2004] Adams, J., Rani, P., and Sarkar, N. (2004). Mixed initiative interaction and robotic systems. In *Workshop on Supervisory Control of Learning and Adaptive Systems, 9^{th} National Conference on Artificial Intelligence*, San Jose, CA.

[Aguiar, 2007] Aguiar, A. (2007). Multiple-model adaptive estimators : Open problems and future directions. In *proceedings of the European Control Conference*, Greece.

[Alami et al., 1998] Alami, r., Chatila, R., Fleury, S., Ghallab, M., and Ingrand, F. (1998). An architecture for autonomy. *International Journal of Robotics Research*, 17 :315–337.

[Albus et al., 2002] Albus, J. S., Huang, H., Messina, E., Murphy, K., Juberts, M., Lacaze, A., Balakirsky, S. B., Shneier, M. O., Hong, T. H., Scott, H. A., Proctor, F. M., Shackleford, W. P., Michaloski, J. L., Wavering, A. J., Kramer, T., Dagalakis, N. G., Rippey, W. G., Stouffer, K. A., and Legowik, S. (2002). 4D/RCS version 2.0 : A reference model architecture for unmanned vehicle systems. Internal Report (NISTIR) 6910, NIST Interagency.

[Albus et al., 1989] Albus, J. S., Lumia, R., Fiala, J., and Wavering, A. J. (1989). NASREM - the NASA/NBS standard reference model for telerobot control system architecture. In *proceedings of the 20th International Symposium on Industrial Robots*, Tokyo, Japan.

[Allen, 1999] Allen, J. F. (1999). Mixed-initiative interaction. *IEEE Intelligent Systems*, 14(5) :14–16.

[Alur, 1998] Alur, R. (1998). Timed automata. In *proceedings of the 11^{th} International Conference on Computer-Aided Verification, LNCS 1633*, pages 8–22.

[Arbib, 1981] Arbib, M. (1981). *Handbook of Physiology - The Nervous System II*, chapter Perceptual structures and distributed motor control, pages 1449–1480. American Physiological Society - V.B. Brooks editors.

[Arkin, 1987] Arkin, C. (1987). Motor schema based navigation for a mobile robot : An approach to programming by behavior. In *proceedings of IEEE International Conference on Robotics and Automation*, pages 264–271, Raleigh, North Carolina.

[Avižienis, 1978] Avižienis, A. (1978). Fault-tolerance, the survival attribute of digital systems. In *IEEE Transactions on dependable and secure computing*, volume 66-10, pages 1109–1125.

[Avižienis et al., 2004] Avižienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on dependable and secure computing*, 1(1) :11–33.

[Baker and Yanco, 2004] Baker, M. and Yanco, H. A. (2004). Autonomy mode suggestions for improving Human-Robot Interaction. In *proceedings of the International Conference on Systems, Man and Cybernetics*, Hague, Netherlands.

[Barbier et al., 2006] Barbier, M., Gabard, J. F., Vizcaino, D., and Bonnet-Torres, O. (2006). PROCOSA : a software package for autonomous system supervision. In *proceedings of the 1^{th} National Conference on Control Architectures of Robots*, Montpellier, France.

[Bartley, 2001] Bartley, G. F. (2001). *The Avionics handbook : Boeing B-777 : Fly-by-wire flight controls*. CRC Press - C. R. Spitzer.

[Basila et al., 1990] Basila, M. J., Stefanek, G., and Cinar, A. (1990). A model-object based supervisory expert system for fault tolerant chemical reactor control. *Computers and Chemical Engineering*, 14(4-5) :551–560.

[Basseville, 1999] Basseville, M. (1999). On fault detectability and isolability. In *proceedings of the European Control Conference*, Karlsruhe, Germany.

[Basseville et al., 2007] Basseville, M., Cocquempot, V., Dague, P., Denoeux, T., Gentil, S., Jard, C., Martin, N., Nikiforov, I., Powell, D., Simonot-Lion, F., and Zolghadri, A. (2007). Synthèse du comité d'experts diagnostic et sûreté de fonctionnement. Technical report.

[Basu et al., 2006] Basu, A., Bozga, M., and Sifakis, J. (2006). Modeling heterogeneous realtime components in BIP. In *International Conference on Software Engineering and Formal Methods (SEFM)*, Pune, India.

[Basu et al., 2008] Basu, A., Gallien, M., Lesire, C., Nguyen, T. H., Bensalem, S., Ingrand, F., and Sifakis, J. (2008). Incremental component-based construction and verification of a robotic system. In *proceedings of the* 18^{th} *European Conference on Artificial Intelligence*, Patras, Greece.

[Becraft and Lee, 1993] Becraft, W. and Lee, P. (1993). An integrated neural network/expert system approach for fault diagnosis. *Computers and Chemical Engineering*, 17(10) :1001–1014.

[Ben-Haim, 1980] Ben-Haim, Y. (1980). An algorithm for failure location in a complex network. *Nuclear Science and Engineering*, 75 :191–199.

[Berman et al., 2009] Berman, S., Schechtman, E., and Edan, Y. (2009). Evaluation of automatic guided vehicle systems. *Journal of Robotics and Computer-Integrated Manufacturing*, 25(3) :522–528.

[Bernard et al., 2000] Bernard, D. E., Gamble, E. B., Rouquette, N. F., Smith, B., Tung, Y. W., Muscettola, N., Dorias, G. A., Kanefsky, B., Kurien, J., Millar, W., Nayal, P., Rajan, K., and Taylor, W. (2000). Remote agent experiment DS1 technology validation report. Technical report, Ames Research Center and JPL.

[Blanke et al., 2003] Blanke, M., Kinnaert, M., Lunze, J., Staroswiecki, M., and Schröder, J. (2003). *Diagnosis and Fault-Tolerant Control*. springer.

[Borrelly et al., 1998] Borrelly, J., Coste-Manière, E., Espiau, B., Kapellos, K., Pissard-Gibollet, R., Simon, D., and Turro, N. (1998). The ORCCAD architecture. *International Journal of Robotics Research, special issue on Integrated Architectures for Robot Control and Programming*, 18(4) :338–359.

[Boskovic and Menta, 2002] Boskovic, J. D. and Menta, R. K. (2002). Multiple-model adaptive flight control scheme for accommodation of actuator failures. *Journal of Guidance, Control, and Dynamics*, 25(4) :712–724.

[Bouti and Ait Kadi, 1994] Bouti, A. and Ait Kadi, D. (1994). A state-of-the-art review of FMEA/FMECA. *International Journal of Reliability, Quality and Safety Engineering (IJRQSE)*, 1(4) :515–543.

[Brandstötter et al., 2007] Brandstötter, M., Hofbaur, M. W., Steinbauer, G., and Wotawa, F. (2007). Model-based fault diagnosis and reconfiguration of robot drives. In *proceedings of the IEEE/RSJ International Conference on Intelligent RObots and Systems*, pages 1203–1209, San Diego, USA.

[Braynov and Hexmoor, 2001] Braynov, S. and Hexmoor, H. (2001). Quantifying relative autonomy in multi-agent interaction. In *proceedings of the Workshop on Autonomy, Delegation and Control, IJCAI' 01*.

[Brissaud, 2011] Brissaud, F. (2011). *Contributions à la modélisation et à l'évaluation de la sureté de fonctionnement de systèmes de sécurité à fonctionnalités numériques*. PhD thesis, Université de Technologie de Troyes.

[Brooks, 1986] Brooks, R. A. (1986). A robust layered control system for a mobile robot. *Robotics and Automation*, 2 :14–23.

[Bruemmer et al., 2002] Bruemmer, D. J., Dudenhoeffer, D. D., and Marble, J. L. (2002). Dynamic-autonomy for urban search and rescue. In *proceedings of the AAAI Mobile Robot Workshop*.

[BSIGroup, 1991] BSIGroup (1991). British Standards Institution Group - BS 5760-5 - Guide to failure modes, effects and criticality analysis (FMEA and FMECA).

[Burch et al., 1992] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, J. (1992). Symbolic model checking : 10^{20} states and beyond. *Information and Computation*, 98(2) :142–170.

[Calefato et al., 2008] Calefato, C., Montanari, R., and Tesauri, F. (2008). *Human Computer Interaction : New Developments*, chapter The Adaptive Automation Design, pages 141–154. Kikuo Asai (Ed.) - InTech.

[Carbou et al., 2002] Carbou, J., Andreu, D., and Fraisse, P. (2002). Events as a key of an autonomous robot controller. In 15^{th} *Triennial World Congress of the International Federation of Automatic Control*, pages 91–97, Barcelona, Spain.

[Carlson, 2004] Carlson, J. (2004). Analysis of how mobile robots fail in the field. Master's thesis, College of Engineering, University of South Florida.

[Carlson and Murphy, 2005] Carlson, J. and Murphy, R. (2005). How UGVs physically fail in the field ? *IEEE Transactions on Robotics*, 21(3) :423–437.

[Chalandon, 2003] Chalandon, X. (2003). Situation awareness en conception système. In *ÉPIQUE'2003 - Deuxièmes Journées d'Étude en Psychologie Ergonomique*, pages 55–62.

[Chand and Carnegie, 2007] Chand, P. and Carnegie, A. (2007). Task allocation and coordination for limited capability mobile robots. In *proceedings of the australian Conference on Robotics and Automation*, Brisbane, Australia.

[Chantery and Pendolé, 2009] Chantery, E. and Pendolé, Y. (2009). Modélisation et intégration du diagnostic actif dans une architecture embarquée. *Journal Européen des Systèmes Automatisés*, 43(7-9) :789–803.

[Chauve, 2010] Chauve, C. (2010). Portage d'une architecture temps réel sur un robot mobile terrestre. Rapport de stage ERII4, Polytech'Montpellier, Université Montpellier 2.

[Chen and Jiang, 2005] Chen, W. and Jiang, J. (2005). Fault-tolerant control against stuck actuator faults. In *proceedings of the Control Theory and Applications*, volume 152(2), pages 138–146.

[Cheung and Stephanopoulos, 1990] Cheung, J.-Y. and Stephanopoulos, G. (1990). Representation of process trends-part I. a formal representation framework. *Computers and Chemical Engineering*, 14(4-5) :495–510.

[Chevance, 2010] Chevance, R. J. (2010). Systèmes à haute disponibilité. In *technologies logicielles Architectures des systèmes*. Collection Techniques de l'ingénieur.

[Chopinaud, 2007] Chopinaud, C. (2007). *Contrôle de l'émergence de comportements dans les systèmes d'agents cognitifs autonomes*. PhD thesis, Université Pierre et Marie Curie - Spécialité : Informatique.

[Clough, 2002] Clough, B. T. (2002). Metrics, schmetrics ! how the heck do you determine a UAV's autonomy anyway ? In *proceedings of the* 1^{st} *AIAA Conference on Unmanned Air Systems*.

[Coquenpot, 2004] Coquenpot, V. (2004). *Contribution à la surveillance des systèmes industriels complexes*. Habilitation à diriger la recherche, Université des Sciences et Technologies de Lille.

[Daigle et al., 2006] Daigle, M., Koutsoukos, X., and Biswas, G. (2006). Multiple fault diagnosis in complex physical systems. In *proceedings of the* 17^{th} *International Workshop on Principles of Diagnosis*.

[Dalgalarrondo, 2003] Dalgalarrondo, A. (2003). A propos de l'autonomie des robots. Technical Report CTA/02 350 108/RIEX/807, DGA - DCEE, Centre Technique Arceuil.

[Dariot, 2010] Dariot, R. (2010). Robot companions for citizens a new generation of soft, compliant, sentient, interactive, embodied machines. In *worshop on Science & Policy Forum on FET Flagships*, Brussels, Belgium.

[Diaz et al., 2003] Diaz, B., Lemai, S., and N.Muscettola (2003). A real-time rover executive based on model-based reactive planning. In *proceedings of International Conference on Advanced Robotics*, pages 1239–1246, Coimbra, Portugal.

[Duan et al., 2005] Duan, Z., Cai, Z., and Yu, J. (2005). Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments : A survey. In *proceedings of the IEEE International Conference on Robotics and Automation*, pages 3439–3444, Barcelona, Spain.

[Duchemin et al., 2004] Duchemin, G., Poignet, P., Dombre, E., and Pierrot, F. (2004). Medically safe and sound. *IEEE Magazine on Robotics & Automation*, 11(2) :46–55.

[Dumont, 2006] Dumont, P. E. (2006). *Tolérance active aux fautes des systèmes d'instrumentation*. PhD thesis, Université des Sciences et Technologies de Lille.

[Dunia et al., 1996] Dunia, R., Joe Qin, S., Edgar, T. F., and Mcavoy, T. J. (1996). Identification of faulty sensors using principal component analysis. *AIChE Journal - American Institute of Chemical Engineers*, 42(10) :2797–2812.

[Durand et al., 2009] Durand, B., Godary, K., Lapierre, L., and Crestani, D. (2009). Inconsistencies evaluation mechanisms for an hybrid control architecture with adaptive autonomy. In *proceedings of the* 4^{th} *National Conference on Control Architectures of Robots*, Toulouse.

[Durand et al., 2010a] Durand, B., Godary-Dejean, K., Lapierre, L., Passama, R., and Crestani, D. (2010a). Fault tolerance enhancement using autonomy adaptation for autonomous mobile robots. In *proceedings of the International Conference on Control and Fault Tolerant Systems (SysTol'10)*, pages 24–29, Nice, France.

[Durand et al., 2010b] Durand, B., Godary-Dejean, K., Lapierre, L., Passama, R., and Crestani, D. (2010b). Using adaptive control architecture to enhance autonomous mobile robot reliability. In *proceedings of the* 11^{th} *Conference Towards Autonomous Robotic Systems*, pages 54–61, Plymouth, UK.

[El Jalaoui, 2007] El Jalaoui, A. (2007). *Gestion Contextuelle de Tâches pour le Contrôle d'un Véhicule Sous-Marin Autonome*. PhD thesis, Université Montpellier II Science et Techniques du Languedoc.

[El Jalaoui et al., 2005] El Jalaoui, A., Andreu, D., and Jouvencel, B. (2005). A Control Architecture for Contextual Tasks Management : Application to the AUV Taipan. In *proceeding of the RSJ/IEEE international conference OCEANS'05 Europe*, Brest, France.

[El Jalaoui et al., 2006] El Jalaoui, A., Andreu, D., and Jouvencel, B. (2006). Contextual management of tasks and instrumentation within an auv control software architecture. In *proceedings of the IEEE/RSJ International Conference on Intelligent RObots and Systems*, Beijing, China.

[Endsley and Kaber, 1999] Endsley, M. and Kaber, D. (1999). Level of automation effects on performance, situation awareness and workload in a dynamic control task. *Ergonomics*, 42(3) :462–92.

[Espiau et al., 1995] Espiau, B., Kapellos, K., and Jourdan, M. (1995). Formal verification in robotics : Why and how ? In *proceedings of the* 7^{th} *International Symposium of Robotics Research*, pages 201–213, Munich, Germany. Cambridge Press.

[Essamé et al., 2000] Essamé, D., Arlat, J., and Powell, D. (2000). Tolérance aux fautes dans les systèmes critiques. LAAS Reports 00151, LAAS.

[Estlin et al., 2001] Estlin, T., Volpe, R., Nesnas, I., Mutz, D., Fisher, F., Engelhardt, B., and Chien, S. (2001). Decision-making in a robotic architecture for autonomy. In *proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space i-SAIRAS*, Canadian Space Agency, St-Hubert, Quebec, Canada.

[Eterno et al., 1985] Eterno, J. S., Weiss, J. L., Looze, D. P., and Willsky, A. (1985). Design issues for fault tolerant-restructurable aircraft control. In *proceedings of the* 24^{th} *Conference on Decision and Control*, pages 900–905, Fort Lauderdale, Florida.

[Feather and Smith, 1999] Feather, M. S. and Smith, B. (1999). Automatic generation of test oracles - from pilot studies to application. In *proceedings of the* 14^{th} *Automated Software Engineering Conference*, Cocoa Beach, Florida.

[Finch and Kramer, 1988] Finch, F. E. and Kramer, M. A. (1988). Narrowing diagnostic focus using functional decomposition. *AIChE Journal - American Institute of Chemical Engineers*, 34(1) :25–36.

[Fleury et al., 1997] Fleury, S., Herrb, M., and Chatila, R. (1997). G^{en}oM : a tool for the specification and the implementation of operating modules in a distributed robot architecture. In *proceedings of the IEEE/RSJ International Conference on Intelligent RObots and Systems*, volume 2, pages 842–849, Grenoble, France.

[Fong et al., 1999] Fong, T., Thorpe, C., and Baur, C. (1999). Collaborative control : A robot-centered model for vehicle teleoperation. In *proceedings of the AAAI Spring Symposium on Agents with Adjustable Autonomy - AAAI Technical Report SS-99-06*, Stanford, CA.

[Frank, 1990] Frank, P. (1990). Fault diagnosis in dynamic system using analytical and knowledge-based redundancy - a survey and some news results. *Automatica*, 26(3) :459–474.

[Frank, 1998] Frank, P. (1998). The application of fuzzy logic to fault diagnosis and supervision. In *LFA*, pages 59–87, Rennes, France.

[Frank and Wünnenberg, 1989] Frank, P. M. and Wünnenberg, J. (1989). *Robust diagnosis using unknown input observer scheme*. Prentice Hall, Englewood, Cliffs.

[Freedy et al., 2007] Freedy, A., De Visser, E. J., Weltman, G., and Coeyman, N. (2007). Measurement of trust in human-robot collaboration. In *International Symposium on Collaborative Technologies and Systems*, pages 106–114, Orlando, FL.

[Galster et al., 2007] Galster, S., Barnes, M., Cosenzo, K., Galster, S., Hollnagel, E., Miller, C., Parasuraman, R., Reising, J., Taylor, R., and Breda, L. V. (2007). *Uninhabited Military Vehicles (UMVs) : Human Factors Issues in Augmenting the Force*, chapter Human Automation Integration. Number RTO-TR-HFM-078. NATO Research and Technology Organisation.

[Gat, 1998] Gat, E. (1998). On three-layer architecture. In *proceedings of the Artificial intelligence and mobile robots : case studies of successful robot systems*, 0-262-61137-6, pages 195–210. MIT press.

[Gertler, 1991] Gertler, J. (1991). Analytical redundancy methods in fault detection and isolation : survey and synthesis. In *proceedings of the IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, volume 1, pages 9–12, Baden Baden, Germany.

[Gertler, 1995] Gertler, J. (1995). Diagnosis parametric faults : from parameters estimation to parity relations. In *proceedings of the American control conference*, Seattle, Washington.

[Gertler and Monajemy, 1995] Gertler, J. and Monajemy, R. (1995). Generating directional residuals with dynamic parity relations. *Automatica*, 31 :627–635.

[Gertler and Singer, 1990] Gertler, J. and Singer, D. (1990). A new structural framework for parity equation-based failure detection and isolation. *Automatica*, 26 :381–388.

[Godary, 2004] Godary, K. (2004). *Validation temporelle de réseaux embarqués critiques et fiables pour l'automobile*. PhD thesis, Institut National des Sciences Appliquées de Lyon.

[Goldman et al., 1997] Goldman, R. P., Musliner, D. J., Boddy, M. S., and Krebsbach, K. D. (1997). The CIRCA model of planning and execution. In *Working Notes of the AAAI Workshop on Robots, Softbots, Immobots : Theories of Action, Planning and Control*, Providence, Rhode Island.

[Goldman et al., 2000] Goldman, R. P., Musliner, D. J., and Pelican, M. J. (2000). Using model-checking to plan hard real-time controllers. In *AIPS Workshop on Model-Theoretic Approaches to Planning*.

[Goodrich et al., 2001] Goodrich, M. A., Olsen Jr., D. R., Crandall, J., and Palmer, T. J. (2001). Experiments in adjustable autonomy. In *proceedings of the IJCAI-01 Workshop on Autonomy, Delegation, and Control : Interaction with Autonomous Agents*, Seattle, Washington.

[Goodrich and Schultz, 2007] Goodrich, M. A. and Schultz, A. C. (2007). Human-robot interaction : A survey. *Foundations and Trends ® In Human-Computer Interaction*, 3 :203–275.

[Gould et al., 2000] Gould, J., Glossop, M., and Ioannides, A. (2000). Review of hazard identification techniques. Technical Report HSL/2005/58, Health and Safety Laboratory.

[Graf, 2005] Graf, B. (2005). Dependability of mobile robots in direct interaction with humans. In *Advances in Human-Robot Interaction*, volume 14 of *Springer Tracts in Advanced Robotics*, pages 488–490. Springer Berlin / Heidelberg.

[Grosclaude, 2004] Grosclaude, I. (2004). Model-based monitoring of component-based software systems. In *proceedings of the* 15^{th} *International Workshop on Principles of Diagnosis*, pages 155–160, Carcassonne, France.

[Guiochet, 2003] Guiochet, J. (2003). *Maîtrise de la sécurité des systèmes de la robotique de service. Approche* UML *basée sur une analyse du risque système*. Laboratoire d'étude des systèmes informatique et automatiques, Institut National des Sciences Appliquées de Toulouse.

[Guiochet and Baron, 2004] Guiochet, J. and Baron, C. (2004). UML based risk analysis - application to a medical robot. In *proceedings of the Quality Reliability and Maintenance 5th International Conference*, pages 213–216, Oxford, UK. Professional Engineering Publishing, I Mech E.

[Hamilton et al., 1996] Hamilton, D., Walker, I. D., and Bennett, J. K. (1996). Fault tolerance versus performance metrics for robot systems. In *proceedings of the IEEE International Conference on Robotics and Automation*, pages 3073–3080.

[Hamilton et al., 2001] Hamilton, K., Lane, D., Taylor, N., and Brown, K. (2001). Fault diagnosis on autonomous robotic vehicles with RECOVERY : An integrated heterogeneous-knowledge approach. In *proceedings of the IEEE International Conference on Robotics and Automation*, pages 3232–3237, Seoul, Korea.

[Hamming, 1959] Hamming, R. W. (1959). Symposium on error detection and correction. In *proceedings of the IFIP Congress*, pages 487–491, Paris, France.

[Hasslacher and Tilden, 1995] Hasslacher, B. and Tilden, M. (1995). Living machine. *Robotics and Autonomous Systems*, 15(1-2) :143–169.

[Henley, 1984] Henley, E. (1984). Application of expert systems to fault diagnosis. In *proceedings of the AIChE annual meeting*, San Francisco, CA.

[Horowitz et al., 1985] Horowitz, I., Arnold, P. B., and Houpis, C. H. (1985). YF-16-CCV flight control system reconfiguration design using quantitative feedback theory. In *proceedings of the National Aerospace Electronics Conference*, pages 578–585, Dayton, OH.

[Hotelling, 1947] Hotelling, H. (1947). *Techniques of Statistical Analysis*, chapter Multivariate quality control - Illustrated by the air testing of sample bombsights, pages 111–184. MacGraw-Hill - Eisenhart, C., Hastay, M.W. and Wallis W.A.Editors.

[Howitt and Luus, 1991] Howitt, G. and Luus, T. (1991). Simultaneaous stabilization of linear single-input system by linear state feedback control. *International Journal of Control*, 54 :1015–1030.

[Huang and Stengel, 1990] Huang, C. Y. and Stengel, R. F. (1990). Restructurable control using proportional-integral implicit model following. *Journal of Guidance, Control, and Dynamics*, 13(2) :303–309.

[Huang et al., 2004] Huang, H.-M., Messina, E., R.Wade, English, R., Novak, B., and Albus, J. (2004). Autonomy measures for robots. In *proceedings of IMECE : International Mechanical Engineering Congress*, Anaheim, CA.

[Ingrand, 2006] Ingrand, F. (2006). LAAS architecture : Open robot. In *proceedings of the* 1^{th} *National Conference on Control Architectures of Robots*, Montpellier, France.

[Iri et al., 1979] Iri, M., Aoki, K., O'Shima, E., and Matsuyama, H. (1979). An algorithm for diagnosis of system failures in the chemical process. *Computers and Chemical Engineering*, 3(1-4) :489–493.

[Isermann, 1984] Isermann, R. (1984). Process fauls detection based on modelling and estimation methods : a survey. *Automatica*, 20(4) :387–404.

[Isermann, 2006] Isermann, R. (2006). *Fault-Diagnosis Systems : An Introduction from Fault Detection to Fault Tolerance*. Springer.

[Isermann and Ballé, 1997] Isermann, R. and Ballé, P. (1997). Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Engineering Practice*, 5(5) :709–719.

[Ishikawa, 1985] Ishikawa, K. (1985). *What is Total Quality Control ? The japanese Way*. Prentice-Hall.

[ISTAG, 2009] ISTAG (2009). Les technologies émergentes et du futur (Future and Emerging Technologies / FET), 7ème programme cadre. Technical report, Commission Européenne - groupe de travail "Information Society Technologies Advisory Group".

[Iwasaki and Simon, 1986] Iwasaki, Y. and Simon, H. A. (1986). Causality in device behavior. *Artificial Intelligence*, 29(1) :3–32.

[Janusz and Venkatasubramanian, 1991] Janusz, M. E. and Venkatasubramanian, V. (1991). Automatic generation of qualitative descriptions of process trends for fault detection and diagnosis. *Engineering Applications of Artificial Intelligence*, 4(5) :329–339.

[Ji et al., 2003] Ji, M., Zhang, Z., Biswas, G., and Sarkar, N. (2003). Hybrid fault adaptive control of a wheeled mobile robot. *IEEE / ASME Transactions on Mechatronics*, 8 :226–233.

[Jiang and Chowdhury, 2005] Jiang, B. and Chowdhury, F. N. (2005). Fault estimation and accommodation for linear MIMO discrete-time systems. *IEEE Transactions on Control Systems Technology*, 13 :493–499.

[Kaber and Endsley, 2004] Kaber, D. B. and Endsley, M. R. (2004). The effects of level of automation and adaptive automation on human performance, situation awareness and workload in a dynamic control task. *Theoretical Issues in Ergonomics Science,*, 5(2) :113–153.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D) :35–45.

[Kanakaraju, 2000] Kanakaraju, S. (2000). Online fault diagnosis system for an autonomous guided vehicle. Master's thesis, University of Cincinnati.

[Koppen-Seliger et al., 1995] Koppen-Seliger, B., Frank, P., and A., W. (1995). Residual evaluation for fault detection and isolation with RCE neural networks. In *proceedings of the American control conference*, pages 3264–3268.

[Kortenkamp et al., 1997] Kortenkamp, D., Bonasso, R., Ryan, D., and Schereckenghost, D. (1997). Traded control with autonomous robot as mixed inititiative interaction. In *proceeding* 14^{th} *national conference on artificial intelligence*, Rhode Island.

[Kortenkamp and Simmons, 2008] Kortenkamp, D. and Simmons, R. (2008). *Springer Handbook of Robotics - Siciliano & Katib Editors*, chapter Robotic Systems Architectures and Programming, pages 187–206. Springer.

[Kramer and Palowitch, 1987] Kramer, M. A. and Palowitch, B. L. (1987). A rule based approach to fault diagnosis using the signed directed graph. *AIChE Journal - American Institute of Chemical Engineers*, 33 :1067–1078.

[Kuipers, 1986] Kuipers, B. (1986). Qualitative simulation. *Artificial Intelligence*, 29(3) :289–338.

[Landau et al., 1998] Landau, I. D., Lozano, R., and M'Saad, M. (1998). *Adaptive Control*. Springer.

[Lapierre and Indiveri, 2007] Lapierre, L. and Indiveri, G. (2007). Non-singular path-following, control of wheeled robots with velocity actuator saturations. In 6^{th} *IFAC Symposium on Intelligent Autonomous Vehicles*, Toulouse, France.

[Lapierre et al., 2010] Lapierre, L., Zapata, R., and Bibuli, M. (2010). Guidance of a flotilla of wheeled robots : a practical solution. In *proceedings of the* 7^{th} *IFAC Symposium on Intelligent Autonomous Vehicles conference*, Lecce, Italy.

[Lapierre et al., 2007] Lapierre, L., Zapata, R., and Lepinay, P. (2007). Simultaneous path following and obstacle avoidance control of a unicycle-type robot. In *IEEE International Conference on Robotics and Automation*, pages 2617–2622, Roma, Italy.

[Laprie et al., 1996] Laprie, J.-C., Arlat, J., Blanquart, J. P., Costes, A., Crouzet, Y., Deswarte, Y., Fabre, J. C., Guillermain, H., Kaâniche, M., Kanoun, K., Mazet, C., Powell, D., Rabéjac, C., and P.Thévenod (1996). *Guide de la Sûreté de Fonctionnement*. CÉPADUÈS.

[Le Guernic et al., 1991] Le Guernic, P., Le Borgne, M., Gautier, T., and Le Maire, C. (1991). Programming real time application with SIGNAL. *proceeding of the IEEE*, 79(9) :1321–1336.

[Legras, 2007] Legras, F. (2007). Etude de l'art du partage d'autorité humain-multi-robot. Technical report, ENST bretagne.

[Legras, 2008] Legras, F. (2008). Experiments in human operation of a swarm of UAVs. In *proceedings of the first conference on Humans Operating Unmanned Systems (HUMOUS'08)*, Brest, France.

[Lemai and Ingrand, 2004] Lemai, S. and Ingrand, F. (2004). Interleaving temporal planning and execution in robotics domains. In *proceedings of the* 19^{th} *National Conference on Artifical Intelligence*, pages 617–622, San Jose, California.

[Lemai-Chenevier, 2004] Lemai-Chenevier, S. (2004). IXTET-EXEC *: planning, plan repair and execution control with time and resource management*. PhD thesis, LAAS-CNRS.

[Leung and Romagnoli, 2000] Leung, D. and Romagnoli, J. (2000). Dynamic probabilistic model-based expert system for fault diagnosis. *Computers and Chemical Engineering*, 24(11) :2473–2492.

[Liao, 2005] Liao, S.-H. (2005). Expert system methodologies and applications – a decade review from 1995 to 2004. *Expert Systems with Applications*, 28(1) :93–103.

[Looze et al., 1985] Looze, D., Weiss, J., Eterno, J., and Barrett, N. (1985). An automatic redesign approach for restructurable control systems. *IEEE Control Systems Magazine*, 5 :16–22.

[Lopa, 2009] Lopa, S. (2009). Mise en place de la téléopération pour un robot mobile. Rapport fin d'étude IUT GEII Montpellier.

[Luenberger, 1971] Luenberger, D. (1971). An introduction to observers. *IEEE Transactions on Automatic Control*, 16(6) :596–602.

[Lussier, 2007] Lussier, B. (2007). *Tolérance aux fautes dans les sytèmes autonomes*. PhD thesis, Institut National Polytechnique De Toulouse.

[Lussier et al., 2005] Lussier, B., Lampe, A., Chatila, R., Guiochet, J., Ingrand, F., Killijian, M.-O., and Powell, D. (2005). Fault tolerance in autonomous systems : How and how much ? In *proceedings* 4^{th} *IARP EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*, Japan. IEEE/RAS.

[Lussier, 2004] Lussier, G. (2004). *Test Guide par la Preuve - Application a la verification d'algorithmes de tolerance aux fautes*. PhD thesis, INSA Toulouse, Toulouse, France.

[MacGregor and Kourti, 1995] MacGregor, J. F. and Kourti, T. (1995). Statistical process control of multivariate processes. *Control Engineering Practice*, 3(3) :403–414.

[Marchand et al., 1998] Marchand, E., Rutten, E., Marchand, H., and Chaumette., F. (1998). Specifying and verifying active vision-based robotic systems with the SIGNAL environment. *International Journal of Robotics Research*, 17(4) :418–432.

[Mariton, 1989] Mariton, M. (1989). Detection delays, false alarm rates and the reconfiguration of control systems. *International Journal of Control*, 49(3) :981–992.

[Massoumnia, 1986] Massoumnia, M.-A. (1986). A geometric approach to the synthesis of failure detection filters. *IEEE Transactions on Automatic Control*, 31(9) :839–846.

[Mataric and Michaud, 2008] Mataric, M. J. and Michaud, F. (2008). *Springer Handbook of Robotics - Siciliano & Katib Editors*, chapter Behavior-Based Systems, pages 891–909. Springer.

[Mechraoui, 2010] Mechraoui, A. (2010). *Co-conception d'un système commandé en réseau sans fil à l'aide de réseaux bayésiens distribués*. PhD thesis, Université de Grenoble.

[Mercier et al., 2009] Mercier, S., Dehais, F., and Tessier, C. (2009). Adjustable autonomy without levels. In *NATO SCI-202 Symposium on "Intelligent uninhabited vehicle guidance systems"*, Neubiberg, Germany.

[Merz, 2001] Merz, S. (2001). Model checking : A tutorial overview. In Cassez, F., Jard, C., Rozoy, B., and Ryan, M., editors, *Modeling and Verification of Parallel Processes*, volume 2067 of *Lecture Notes in Computer Science*, pages 3–38. Springer-Verlag, Berlin.

[Meystel et al., 2000] Meystel, A., Albus, J., Messina, E., Evans, J., Fogel, D., and Hargrove, W. (2000). Measuring performance and intelligence of systems with autonomy : Metrics of intelligence of constructed systems. In *Measuring Performance and Intelligence of Systems : proceedings from the 2000 PerMIS Workshop*, Gaithersburg, MD.

[Miller et al., 2005a] Miller, C., Funk, H., P.Wu, Goldman, R., Meisner, J., and Chapman, M. (2005a). The playbook ^{TM} approach to adaptive automation. In *proceedings of the Human Factors and Ergonomics Society's* 49^{th} *Annual Meeting*, Orlando, FL.

[Miller et al., 2005b] Miller, C. A., Funk, H. B., Goldman, R. P., Meisner, J., and Wu, P. (2005b). Implications of adaptive vs. adaptable UIs on decision making : Why "automated adaptiveness" is not always the right answer. In *proceedings of the* 1^{st} *International Conference on Augmented Cognition*, Las Vegas, NV.

[Milne, 1987] Milne, R. (1987). Strategies for diagnosis. *IEEE Transactions on Systems, Man and Cybernetics*, 17(3) :333–339.

[Moerder et al., 1989] Moerder, D. D., Halyo, N., Broussard, J. R., and Caglayan, A. K. (1989). Application of precomputed control laws in a reconfigurable aircraft flight control system. *Journal of Guidance, Control, and Dynamics*, 12(3) :325–333.

[Moreno et al., 2008] Moreno, A., Etxeberria, A., and Umerez, J. (2008). The autonomy of biological individuals and artificial models. *BioSystems*, 91 :309–319.

[Morisset et al., 2004] Morisset, B., Infantes, G., Ghallab, M., and Ingrand, F. (2004). ROBEL : Synthesizing and conrolling complex robust robot behaviors. In *proceedings of the 16^{th} European Conference on Artificial Intelligence (ECAI)*, pages 1067–1068, Valencia - Spain.

[Mortureux, 2010] Mortureux, Y. (2010). Arbres de défaillance, des causes et d'événement. In *Sécurité et Gestion des risques*. Collection Techniques de l'ingénieur.

[Murad et al., 1996] Murad, G. A., Postlethwaite, I., and Gu, D. (1996). A robust design approach to integrated controls and diagnostics. In *proceedings of the 13^{th} IFAC World Congress*, San Francisco, CA.

[Murata, 1989] Murata, T. (1989). Petri nets : Properties, analysis and applications. *Proceedings of the IEEE*, 77(4) :541–580.

[Murphy and Hershberger, 1999] Murphy, R. R. and Hershberger, D. (1999). Handling sensing failures in autonomous mobile robots. *The International Journal of Robotics Research*, 18(4) :382–400.

[Muscettola, 1998] Muscettola, N. (1998). Remote agent : To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2) :5–48.

[Muscettola et al., 2002] Muscettola, N., Dorais, G. A., Fry, C., Levinson, R., and Plaunt, C. (2002). IDEA : Planning at the core of autonomous reactive agents. In *proceedings of the 3^{rd} International NASA Workshop on Planning and Scheduling for Space*, Houston, TX.

[Nana, 2007] Nana, L. T. (2007). Investigating software dependability mechanisms for robotics applications. *The IPSI BgD Transactions on Internet Research*, 3(1) :50–55.

[Nana et al., 2005] Nana, L. T., Marcé, L., Opderbecke, J., Perrier, M., and Rigaud, V. (2005). Investigation of safety mechanisms for oceanographic AUV missions programming. In *proceedings of the IEEE OCEAN 05 Europe Conference*, Brest, France.

[Nikam and Hall, 1999] Nikam, U. and Hall, E. (1999). A fault diagnosis system for mobile robot. Technical report, Departement of Mechanical, Industrial and Nuclear Engineering, University Of Cincinatti.

[Nikam and Hall, 1997] Nikam, U. and Hall, E. L. (1997). Fault diagnostic system for a mobile robot. In *proceedings of SPIE Volume : 3208 - Intelligent Robots and Computer Vision XVI : Algorithms, Techniques, Active Vision, and Materials Handling*.

[Noura et al., 1993] Noura, H., Fonte, C., and Robert, M. (1993). Fault tolerant control using simultaneous stabilization. In *proceedings of the International Conference on Systems, Man and Cybernetics - Systems Engineering in the Service of Humans*, pages 605–610, Le Touquet , France.

[Ochi, 1993] Ochi, Y. (1993). Application of feedback linearization method in a digital restructurable flight control system. *Journal of Guidance, Control, and Dynamics*, 16(1) :111–117.

[Olive, 2010] Olive, X. (2010). FDI(R) for satellite at thalès alenia space how to deal with high availability and robustness in space domain ? In *proceedings of the International Conference on Control and Fault Tolerant Systems*, pages 837–842, Nice, France.

[Opperman, 1994] Opperman, R. (1994). *Adaptive user support*. Erlbaum.

[Orebäck and Christensen, 2003] Orebäck, A. and Christensen, H. I. (2003). Evaluation of architectures for mobile robotics. *Autonomous Robots*, 14 :33–49.

[Oustaloup and Melchior, 1993] Oustaloup, A. and Melchior, P. (1993). The great principles of the CRONE control. In *proceedings of the International Conference on Systems, Man and Cybernetics - Systems Engineering in the Service of Humans*, pages 118–129, Le Touquet , France.

[Parasuraman et al., 2007] Parasuraman, R., Barnes, M., and Cosenzoc, K. (2007). Adaptive automation for human-robot teaming in future command and control systems. *The International C2 Journal*, 1 :43–68.

[Parasuraman et al., 2000] Parasuraman, R., Sheridan, T., and C.Wickens (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A : Systems and Humans*, 30(3) :286–297.

[Parker, 1996] Parker, L. E. (1996). *Distributed Autonomous Robotic Systems 2*, chapter Multi-Robot Team Design for Real-World Applications, pages 91–102. Springer-Verlag, Tokyo.

[Passama, 2010] Passama, R. (2010). ContrACT : une méthodologie de conception et de développement d'architectures de contrôle de robots. Technical Report lirmm-00505309, LIRMM.

[Patton, 1997] Patton, R. J. (1997). Fault-tolerant control : the 1997 situation. In *proceedings of the IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, volume 2, pages 1033–1055, Kingston Upon Hull, UK.

[Pearson, 1901] Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6) :559–572.

[Peng and Reggia, 1987a] Peng, Y. and Reggia, J. A. (1987a). A probabilistic causal model for diagnostic problem solving part I : Integrating symbolic causal inference with numeric probabilistic inference. *IEEE Transactions on Systems, Man and Cybernetics*, 17(:2) :146–162.

[Peng and Reggia, 1987b] Peng, Y. and Reggia, J. A. (1987b). A probabilistic causal model for diagnostic problem solving part ii : Diagnostic strategy. *IEEE Transactions on Systems, Man and Cybernetics*, 17(3) :395–406.

[Prasad et al., 1998] Prasad, P. R., Davis, J. F., Jirapinyo, Y., osephson, J. R., and Bhalodia, M. (1998). Structuring diagnostic knowledge for large-scale processsystems. *Computers and Chemical Engineering*, 22(12) :1897–1905.

[Py and Ingrand, 2004] Py, F. and Ingrand, F. (2004). Real-time execution control for autonomous systems. In *proceedings of the* 2^{nd} *European Congress ERTS, Embedded Real Time Software*, Toulouse, France.

[Qin and Li, 1999] Qin, S. J. and Li, W. (1999). Detection, identification, and reconstruction of faulty sensors with maximized sensitivity. *AIChE Journal - American Institute of Chemical Engineers*, 45(9) :1963–1976.

[Ramesh et al., 1989] Ramesh, T., Davis, J., and Schwenzer, G. (1989). Catcracker : an expert system for process and malfunction diagnosis in fluid catalytic cracking units. In *proceedings of the AIChE annual meeting*, San Francisco, CA.

[Ramesh et al., 1988] Ramesh, T. S., Shum, S. K., and Davis, J. F. (1988). A structured framework for efficient problem solving in diagnostic expert systems. *Computers and Chemical Engineering*, 12(9-10) :891 – 902.

[Ranganathan and Koenig., 2003] Ranganathan, A. and Koenig., S. (2003). A reactive robot architecture with planning on demand. In *proceedings of the IEEE/RSJ International Conference on Intelligent RObots and Systems*, pages 1462–1468, Las Vegas, Nevada, USA.

[Rasmussen, 1986] Rasmussen, J. (1986). *Information Processing and Human-Machine Interaction : An Approach to Cognitive Engineering*. Elsevier Science, New York, NY, USA.

[Röfer and Lankenau, 1999] Röfer, T. and Lankenau, A. (1999). Ensuring safe obstacle avoidance in a shared-control system". In *proceedings of the* 7^{th} *International conference on emergent technologies ans factory automation*, Barcelone, Spain.

[Rich and Venkatasubramanian, 1987] Rich, S. H. and Venkatasubramanian, V. (1987). Model-based reasoning in diagnostic expert systems for chemical process plants. *Computers and Chemical Engineering*, 11(2) :111 – 122.

[Rosenblatt, 1997] Rosenblatt, J. (1997). DAMN *: A Distributed Architecture for Mobile Navigation*. PhD thesis, Carnegie Melon University.

[Ross, 1977] Ross, D. (1977). Structured analysis SA : A language for communicating ideas. *IEEE Transactions on Software Engineering*, 3(1) :16–34.

[Roumeliotis et al., 1998] Roumeliotis, S. I., Sukhatme, G. S., and Bekey, G. A. (1998). Fault detection and identification in a mobile robot using multiple model estimation. In *proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2223–2228.

[Rutten, 2001] Rutten, E. (2001). A framework for using discrete control synthesis in safe robotic programming and teleoperation. In *proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 4104–4109.

[Sacks, 1988] Sacks, E. (1988). Qualitative analysis by piecewise linear approximation. *Artificial Intelligence in Engineering*, 3(3) :151 – 155.

[Saeks and Murray, 1982] Saeks, R. and Murray, J. (1982). Fractional representation, algebraic geometry, and the simultaneous stabilization problem. *IEEE Transactions on Automatic Control*, 27 :895 – 903.

[Sánchez et al., 2006] Sánchez, A., Cuautle, R., Zapata, R., and Osorio, M. (2006). *Advances in Artificial Intelligence*, volume 4140/2006, chapter A Reactive Lazy PRM Approach for Nonholonomic Motion Planning, pages 542–551. Springer Berlin - Heidelberg.

[Saraiva and Crocquefer, 2010] Saraiva, P. and Crocquefer, S. (2010). Réalisation d'un programme en langage c de scrutation de la qualité du réseau wifi. Projet Tuteuré IUT GEII Montpellier.

[Schneider et al., 1998] Schneider, S. A., Chen, V. W., Pardo-Castellote, G., and Wang, H. H. (1998). CONTROLSHELL : A software architecture for complex electromechanical systems. *The International Journal of Robotics Research*, 17 :360–380.

[Scholtz, 2003] Scholtz, J. (2003). Theory and evaluation of human robot interactions. In *proceedings of the* 36^{th} *Annual Hawaii International Conference on System Sciences*.

[Sheridan and Verplank, 1978] Sheridan, T. B. and Verplank, W. L. (1978). Human and computer control of undersea teleoperators. Technical report, Man-Machine Systems Lab Report - Cambridge : Massachusetts Institute of Technology.

[Siewiorek and Swarz, 1992] Siewiorek, D. P. and Swarz, R. S. (1992). *Reliable computer systems (2nd ed.) : design and evaluation*. Digital Press, Newton, MA, USA.

[Simani et al., 2003] Simani, S., Patton, R., and Fantuzzi, C. (2003). *Model-Based Fault Diagnosis in Dynamic Systems Using Identification Techniques*. Springer-Verlag, Secaucus, NJ, USA.

[Simmons et al., 2000] Simmons, R., Pecheur, C., and Srinivasan, G. (2000). Towards automatic verification of autonomous systems. In *proceedings of the IEEE/RSJ International Conference on Intelligent RObots and Systems*.

[Simon et al., 1997] Simon, D., Espiau, B., Kapellos, K., and Pissard-Gibollet, R. (1997). ORCCAD : Software engineering for- real-time robotics, a technical insight",. *Robotica, Special issues on Languages and Software in Robotics*, 15(1) :111–116.

[Sommerville, 1988] Sommerville, I. (1988). *Le Génie logiciel et ses applications*. 2-7296-0180-5. InterÉdition, Paris, France.

[Soylemez and Seider, 1973] Soylemez, S. and Seider, W. D. (1973). A new technique for precedence-ordering chemical process equation sets. *AIChE Journal - American Institute of Chemical Engineers*, 19 :934–942.

[Steels, 1995] Steels, L. (1995). When are robots intelligent autonomous systems agents ? *Robotics and Autonomous Systems*, 15 :3–9.

[Steinbauer et al., 2005] Steinbauer, G., Mörth, M., and Wotawa, F. (2005). Real-time diagnosis and repair of faults of robot control software robocup 2005. In *the ROBOCUP*, pages 13–23, Osaka , Japan.

[Steinbauer and Wotawa, 2005] Steinbauer, G. and Wotawa, F. (2005). Detecting and locating faults in the control software of autonomous mobile robots. In *proceedings of the* 16^{th} *International Workshop on Principles of Diagnosis*, pages 13–18, Monterey, USA.

[Tarifa and Scenna, 1997] Tarifa, E. E. and Scenna, N. J. (1997). Fault diagnosis, direct graphs, and fuzzy logic. *Computers and Chemical Engineering*, 21(Supplement 1) :649–654. Supplement to Computers and Chemical

Engineering, 6^{th} International Symposium on Process Systems Engineering and 30^{th} European Symposium on Computer Aided Process Engineering.

[Taylor, 2001] Taylor, R. (2001). Cognitive cockpit systems engineering : Pilot authorisation and control of tasks. In *proceedings of the 8^{th} Conferences on Cognitive Sciences Approaches to process Control (CSAPC-01) - R. Onken (Ed)*, Neubiberg, Germany.

[Taylor, 2006] Taylor, R. M. (2006). Human automation integration for supervisory control of uavs. In *Virtual Media for Military Applications - Meeting Proceedings RTO-MP-HFM-136, Paper 12.*, pages 12–1 – 12–10.

[Thelliol et al., 2003] Thelliol, D., Sauter, D., and Ponsart, J. C. (2003). A multiple model based approach for fault tolerant control in non linear systems. In *proceedings of the IFAC fault detection supervision and safety of technical Processes*, pages 149–154, Washington, D.C., USA.

[Tomatis et al., 2003] Tomatis, N., Terrien, G., Piguet, R., Burnier, D., Bouabdallah, S., Arras, K., and Siegwart, R. (2003). Designing a secure and robust mobile interacting robot for the long term. In *International Conference on Robotics and Automation*, Taipei, Taiwan.

[Tomatis et al., 2002] Tomatis, N., Terrien, G., Piguet, R., Burnier, D., Bouabdallah, S., and Siegwart, R. (2002). Design and system integration for the expo.02 robot. In *Workshop on Robots in Exhibitions, International Conference on Intelligent RObots and Systems*, Lausanne, Switzerland.

[Touaf, 2005] Touaf, S. (2005). *Diagnostic logique des systèmes complexes dynamiques dans un contexte multi-agent*. PhD thesis, université Joseph Fourier.

[Vedam and Venkatasubramanian, 1997] Vedam, H. and Venkatasubramanian, V. (1997). A wavelet theory based adaptive trend analysis system for process monitoring and fault diagnosis. In *proceedings of the American Control Conference*, Albuquerque, New Mexico.

[Venkatasubramanian, 1985] Venkatasubramanian, V. (1985). Inexact reasoning in expert systems : A stochastic parallel network approach. In *proceedings of the 2^{nd} Conference on Artificial Intelligence Applications*, pages 191–195.

[Venkatasubramanian, 1989] Venkatasubramanian, V. (1989). CATDEX, knowledge-based systems in process engineering : case studies in heuristic classification. In *the CACHE Corporation*.

[Venkatasubramanian et al., 2003a] Venkatasubramanian, V., Rengaswamy, R., and Kavuri, S. N. (2003a). A review of process fault detection and diagnosis : Part II : Qualitative models and search strategies. *Computers and Chemical Engineering*, 27(3) :313–326.

[Venkatasubramanian et al., 2003b] Venkatasubramanian, V., Rengaswamy, R., Kavuri, S. N., and Yin, K. (2003b). A review of process fault detection and diagnosis : Part III : Process history based methods. *Computers and Chemical Engineering*, 27(3) :327–346.

[Venkatasubramanian et al., 2003c] Venkatasubramanian, V., Rengaswamy, R., Yin, K., and Kavuri, S. N. (2003c). A review of process fault detection and diagnosis : Part I : Quantitative model-based methods. *Computers and Chemical Engineering*, 27(3) :293–311.

[Verfaillie, 2006] Verfaillie, G. (2006). Une architecture générique modulaire pour le contrôle d'un engin spatial autonome. In *journées Francophones Planification, Décision, Apprentissage*, Toulouse, France.

[Vishnuvardhanaraj, 2000] Vishnuvardhanaraj, S. (2000). Failure mode analysis of autonomous ground robot using java database connectivity jdbc. Master's thesis, University of Cincinnati.

[Volpe et al., 2000] Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., and Das, H. (2000). CLARAty : Coupler layer architecture for robotic autonomy. Technical report, Jet Propulsion Laboratory.

[Volpe et al., 2001] Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., and Das, H. (2001). The CLARAty architecture for robotic autonomy. In *proceedings of the 2001 IEEE Aerospace Conference*, volume 1, pages 121–132, Big Sky Montana.

[Weber and Wotawa, 2008] Weber, J. and Wotawa, F. (2008). Diagnosis and repair of dependent failures in the control system of a mobile autonomous robot. *Applied Intelligence*, 29 :1–18.

[Whiteley and Davis, 1994] Whiteley, J. and Davis, J. (1994). A similarity-based approach to interpretation of sensor data using adaptive resonance theory. *Computers and Chemical Engineering*, 18(7) :637–661. An International Journal of Computer Applications in Chemical Engineering.

[Wickens et al., 1998] Wickens, C. D., Lee, J. D., Liu, Y., and Gordon-Becker, S. (1998). *An Introduction to Human Factors Engineering*. Longman, New York.

[Wilcox and Himmelblau, 1994] Wilcox, N. and Himmelblau, D. (1994). The possible cause and effect graphs (PCEG) model for fault diagnosis-II. applications. *Computers and Chemical Engineering*, 18(2) :117–127.

[Williams et al., 2003a] Williams, B. C., Ingham, M. D., Chung, S. H., and Elliott, P. H. (2003a). Model-based programming of intelligent embedded systems and robotic space explorers. *proceedings of the IEEE*, 91(1) :212–237.

[Williams et al., 2003b] Williams, B. C., Ingham, M. D., Chung, S. H., Elliott, P. H., Hofbaur, M., and Sullivan, T. (2003b). Model-based programming of fault-aware systems. *AI Magazine*, 24(4) :61–75.

[Williams and Nayak., 1996] Williams, B. C. and Nayak., P. P. (1996). A model-based approach to reactive self-configuring systems. In Press, A., editor, *proceedings of AAAI-96*, pages 971–978, Cambridge, Mass.

[Willsky and Jones, 1976] Willsky, A. and Jones, H. (1976). A generalized likelihood ratio approach to the detection and estimation of jumps in linear systems. *IEEE Transactions on Automatic control*, 21(1) :108–112.

[Willsky, 1976] Willsky, A. S. (1976). A survey of design methods for failure detection in dynamics systems. *Automatica*, 12 :601–611.

[Yen, 2003] Yen, G. G.and Ho, L. W. (2003). On line model-based fault diagnosis and accommodation. *IEEE Transaction on Industrial Electronics*, 50(2) :296–312.

[Youla et al., 1974] Youla, D., Jr., J. B., and Lu, C. (1974). Single-loop feedback-stabilization of linear multivariable dynamical plants. *Automatica*, 10(2) :159–173.

[Young, 1981] Young, P. (1981). Parameter estimation for continuous-time models–a survey. *Automatica*, 17(1) :23–39.

[Zhang, 2010] Zhang, L. (2010). *Self-Adaptive Markov Localization for Single-Robot and Multi-Robot Systems*. PhD thesis, Université Montpellier 2.

[Zhang and Zapata, 2009] Zhang, L. and Zapata, R. (2009). A three-step localization method for mobile robots. In *International Conference on Automation, Robotics and Control Systems*, pages 50–56, Orlando, Florida.

[Zhang and Jiang, 2003] Zhang, Y. and Jiang, J. (2003). Fault tolerant control system design with explicit consideration of performance degradation. *IEEE Transactions on Aerospace and Electronic Systems*, 39 :838–848.

[Zhang and Jiang, 2008] Zhang, Y. and Jiang, J. (2008). Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2) :229–252.

[Zhang and Jiang, 2002] Zhang, Y. M. and Jiang, J. (2002). Active fault-tolerant control system against partial actuator failures. *Control Theory and Applications*, 149(1) :95–104.

[Zhao et al., 1997] Zhao, J., Chen, B., and Shen, J. (1997). A hybrid ANN-ES system for dynamic fault diagnosis of hydrocracking process. *Computers & Chemical Engineering*, 21(Supplement 1) :S929–S933. Supplement to Computers and Chemical Engineering, 6th International Symposium on Process Systems Engineering and 30th European Symposium on Computer Aided Process Engineering.

[Zimmel et al., 2004] Zimmel, B. C., Long, M. T., Carlson, J., and Murphy, R. R. (2004). Distributed error handling and HRI. In *proceedings of the IEEE International Conference on Robotics and Automation*, pages 1874–1881, New Orleans, LA.

[Zimmermann et al., 2008] Zimmermann, M., Volden, T., Kirstein, K.-U., Hafizovic, S., Lichtenberg, J., Brand, O., and Hierlemann, A. (2008). A CMOS-based integrated-system architecture for a static cantilever array. *Sensors and Actuators B : Chemical*, 131(1) :254–264. Special Issue : Selected Papers from the 12th International Symposium on Olfaction and Electronic Noses.

Bibliographie de l'auteur

Conférences internationales avec comité de lecture

2010 – **B. Durand**, K. Godary-Dejean, L. Lapierre,D. Crestani, *Global methodology in control architecture to improve mobile robot reliability*, in proceedings of the 2010 - in proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 10), pp 1018-1023, Taipei, Taiwan.

– **B. Durand**, K. Godary-Dejean, L. Lapierre,D. Crestani, *Using Adaptive Control Architecture to enhance Autonomous Mobile Robot Reliability*, in proceedings of the 2010 - 11th Conference Towards Autonomous Robotic Systems (TAROS 10), pp 54-61, Plymouth, UK.

– **B. Durand**, K. Godary-Dejean, L. Lapierre, R. Passama, D. Crestani, *Fault tolerance enhancement using autonomy adaptation for autonomous mobile robots* , in proceedings of the 2010 - Conference on Control and Fault-Tolerant Systems (SysTol'10), pp 24-29, Nice, France.

Conférences nationales avec comité de lecture

2009 – **B. Durand**, K. Godary-Dejean, L. Lapierre, D. Crestani, *Inconsistencies Evaluation Mechanisms for a Hybrid Control Architecture with adjustable autonomy*, 4th National Conference on Control Architectures of Robots (CAR'09 - Toulouse).

2010 – **B. Durand**, K. Godary-Dejean, L. Lapierre, R. Passama, D. Crestani, *Reliability improvement in control architecture for mobile robots : implementation using* COTAMA, 5th National Conference on Control Architectures of Robots (CAR'10 - Douai).

Annexes

Annexe A

Les données de l'architecture

Les données de contrôle de l'architecture

Dans cette annexe sont regroupées, pour indication, les types des variables définies au sein de l'architecture pour lesquelles des ports de communication de module de contrôle sont définis.

TArchiVitesseRobot		
type	champ	commentaire
float	u	vitesse en radian par sec
float	r	angle en rad

Table A.1 – TArchiVitesseRobot

TArchiDonneeUltraSon		
type	champ	commentaire
float	distance[16]	Distance sonar-obstacle
long long	date[16]	Dernière date de mise à jour

Table A.2 – TArchiDonneeUltraSon

TArchiUS		
type	champ	commentaire
float	distance[16]	Distance rayon robot-obstacle
float	angle[16]	Angle du rayon dans le repère du robot
long long	date[16]	Dernière date de mise à jour

Table A.3 – TArchiUS

TArchiEtatRobot		
type	champ	commentaire
double	roueG	vitesse roue Gauche
double	roueD	vitesse roue Droite
double	posX	position x
double	posY	position y
double	posTH	orientation du robot θ
double	u	vitesse d'avance du robot
double	r	vitesse de rotation du robot

Table A.4 – TArchiEtatRobot

TArchiMCL		
type	champ	commentaire
double	x	position x
double	y	position y
double	theta	orientation du robot θ
double	u	vitesse d'avance du robot
double	r	vitesse de rotation du robot
double	Ecart_type	

Table A.5 – TArchiMCL

TPositionRobot		
type	champ	commentaire
double	x	position x
double	y	position y
double	theta	orientation du robot θ

Table A.6 – TPositionRobot

TArchiRabbit		
type	champ	commentaire
double	rob_x	position x robot
double	rob_y	position y robot
double	rob_theta	theta robot
double	rab_x	position x rabbit
double	rab_y	position y rabbit
double	rab_theta	theta rabbit = delta
double	TmoinsD	différence entre theta et delta
double	s	position rabbit sur chemin
double	c	
double	s1	position robot dans repère SF du rabbit
double	y1	position robot dans repère SF du rabbit

Table A.7 – TArchiRabbit

Annexe B

Tableaux récapitulatifs AMDEC des modes de fonctionnement de suivi de chemin autonome dégradés

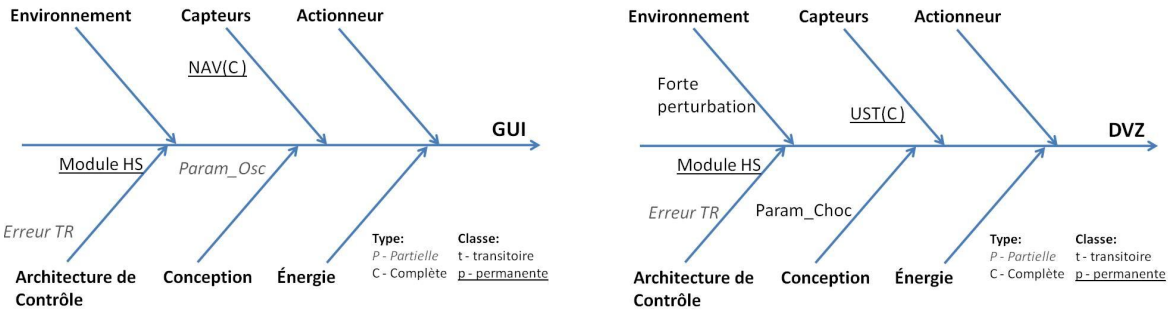
Nous allons présenter ici les différents tableaux récapitulatifs AMDEC des modes de fonctionnement de suivi de chemin autonome dégradés ainsi que les diagrammes d'Ishikawa des modules redondants.

B.1 Diagrammes D'Ishikawa des modules redondants GUI et DVZ

Les deux modules GUI et DVZ permettent de mettre en œuvre des modes de fonctionnement dégradés du suivi de chemin avec évitement d'obstacle.

Le module GUI (figure B.1a) Le module possède les mêmes défaillances Module HS et Erreur TR que les autres modules de l'architetcure. Par contre, une paramétrisation inadaptée de l'algorithme de suivi de chemin peut être à l'origine d'oscillations **Param_Osc** incontrôlées de la trajectoire du robot et engendre donc une défaillance partielle du module SMZ.

Le module DVZ (figure B.1b) est défaillant si les capteurs proximétriques **UST(C)** le sont. Une défaillance complète du module est détectée si une mauvaise paramétrisation de l'algorithme ne permet pas d'éviter un choc avec un obstacle **Param_choc**. Une **forte perturbation** de l'environnement, par exemple un couloir bloqué ou un « cul de sac », ne peut être compenser par cet algorithme d'évitement d'obstacle, Le module est donc complètement défaillant.



(a) Module GUI

(b) Module DVZ

Figure B.1 – Diagrammes d'Ishikawa des modules redondants GUI et DVZ

B.2 Mode de fonctionnement de suivi de chemin DVZ

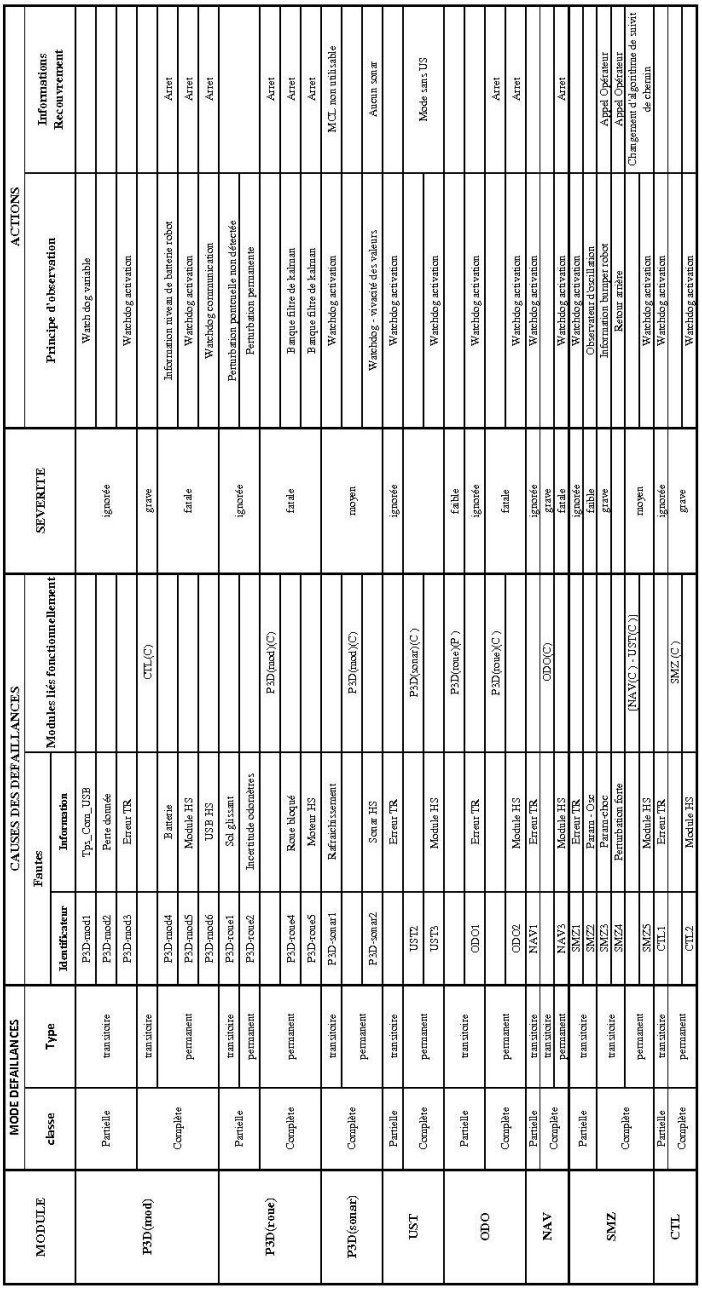
Le mode de fonctionnement de suivi de chemin autonome avec évitement d'obstacle DVZ utilise le schéma de commande proposé figure 7.6. La table B.1 est le résultat de la méthodologie AMDEC appliqué sur ce mode de fonctionnement.

MODULE	MODE DEFAILLANCES		CAUSES DES DEFAILLANCES			SEVERITE	ACTIONS	
	classe	Type	Fautes: Identificateur	Fautes: Information	Modules liés fonctionnellement		Principe d'observation	Informations Recouvrement
P3D(mod)	Partielle	transitoire	P3D-mod1	Tpc_Com_USB		ignorée	Watch dog variable	
			P3D-mod2	Perte donnée				
			P3D-mod3	Erreur TR			Watchdog activation	
	Complète	transitoire			CTL(C)	grave		
		permanent	P3D-mod4	Batterie		fatale	Information niveau de batterie robot	Arret
			P3D-mod5	Module HS			Watchdog activation	Arret
			P3D-mod6	USB HS			Watchdog communication	Arret
P3D(roue)	Partielle	transitoire	P3D-roue1	Sol glissant		ignorée	Permutation ponctuelle non détectée	Utilisation préventive du MCL pour correction passive
		permanent	P3D-roue2	Incertitude odomètres			Permutation permanente	Arret
	Complète	permanent			P3D(mod)(C)	fatale		
			P3D-roue4	Roue bloquée			Banque filtre de kalman	Arret
			P3D-roue5	Moteur HS			Banque filtre de kalman	Arret
P3D(sonar)	Complète	transitoire	P3D-sonar1	Refraction/sonnet		moyen	Watchdog activation	MCL non utilisable
		permanent	P3D-sonar2	Sonar HS	P3D(mod)(C)		Watchdog : vivacité des valeurs	Aucun sonar
UST	Partielle	transitoire		Erreur TR		ignorée	Watchdog activation	
			UST1	Sonar aveugle			Limite d'un nombre de sonar actif	MCL non utilisable
	Complète	permanent	UST2		P3D(sonar)(C)	moyen		
			UST3	Module HS			Watchdog activation	Mode sans US
ODO	Partielle	transitoire			P3D(roue)(P)	faible		
			CDO1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent			P3D(roue)(C)	fatale		
			CDO2	Module HS			Watchdog activation	Arret
MCL	Partielle	transitoire	MCL1	Permutation		faible	Différence entre position ODO et MCL	
			MCL3	Param_nbre Particules			Watchdog activation	reparam nbre particules
	Complète	transitoire	MCL4	Dispersion particules	UST(P)	grave	Analyse Ecart type des particules	
		permanent	MCL5	Module HS			Watchdog activation	Mode sans MCL
NAV	Partielle	transitoire	NAV1	Erreur TR		ignorée	Watchdog activation	
	Complète	transitoire			MCL (C)	grave		
		permanent	NAV3	Module HS		fatale	Watchdog activation	Arret
GUI	Partielle	transitoire	GUI1	Erreur TR		ignorée	Watchdog activation	
		transitoire	GUI2	Param - Cnge		faible	Observateur d'oscillation	
		transitoire	GUI3	Permutation fixe			Retour arrière	Appel Opérateur
	Complète	permanent	GUI4	Module HS	[NAV(C) - UST(C)]	moyen		
CTL	Partielle	transitoire	CTL1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent	CTL2		GUI (C)	grave	Watchdog activation	
DVZ	Partielle	transitoire	DVZ1	Module HS		ignorée	Watchdog activation	
		transitoire	DVZ21	Erreur TR		grave	Information bumper robot	Appel Opérateur
	Complète	transitoire	DVZ22	Param cinc	CTL (C)	grave		
		permanent	DVZ23	Module HS		grave	Watchdog activation	

Table B.1 – Tableau AMDEC du suivi de chemin autonome DVZ

B.3 Mode de fonctionnement de suivi de chemin SMZ sans MCL

Nous retrouvons dans la table B.2 le résultat de l'analyse AMDEC du mode de fonctionnement de suivi de chemin SMZ avec évitement d'obstacle sans localisation Monte Carlo.



MODULE	MODE DEFAILLANCES		CAUSES DES DEFAILLANCES			SEVERITE	ACTIONS	
	classe	Type	Fautes		Modules liés fonctionnellement		Principe d'observation	Informations Recouvrement
			Identificateur	Information				
P3D(mod)	Partielle	transitoire	P3D-mod1	Typ_Conn_USB		ignorée	Wath dog variable	
			P3D-mod2	Perte donnée				
			P3D-mod3	Erreur TR			Watchdog activaton	
	Complète	transitoire			CTL(C)	grave		
		permanent	P3D-mod4	Batterie		farale	Information niveau de batterie robot	Arret
			P3D-mod5	Module HS			Watchdog activaton	Arret
			P3D-mod6	USB HS			Watchdog communication	Arret
P3D(roue)	Partielle	transitoire	P3D-roue1	Sol glissant		ignorée	Perturbation ponctuelle non détectée	
		permanent	P3D-roue2	Incertitude coloriètres			Perturbation permanente	Arret
	Complète	permanent			P3D(mod)(C)	farale		
			P3D-roue4	Roue bloqué			Banque filtre de kalman	Arret
			P3D-roue5	Moteur HS			Banque filtre de kalman	Arret
P3D(somar)	Complète	transitoire	P3D-somar1	Raffraichissement		moyen	Watchdog activaton	MCL non utilisable
		permanent			P3D(mod)(C)			
			P3D-somar2	Senar HS			Watchdog - vivacité des valeurs	Aucun senar
UST	Partielle	transitoire		Erreur TR		ignorée	Watchdog activaton	
	Complète	permanent	UST2		P3D(somar)(C)			Mode sans US
			UST3	Module HS			Watchdog activaton	
ODO	Partielle	transitoire	ODO1	Erreur TR		faible		
					P3D(roue)(P)	ignorée	Watchdog activaton	
	Complète	permanent	ODO2	Module HS		farale		Arret
					P3D(roue)(C)		Watchdog activaton	Arret
NAV	Partielle	transitoire	NAV1	Erreur TR		ignorée	Watchdog activaton	
	Complète	transitoire			ODO(C)	grave		
		permanent	NAV3	Module HS		farale	Watchdog activaton	Arret
SMZ	Partielle	transitoire	SMZ1	Erreur TR		ignorée	Watchdog activaton	
			SMZ2	Param - Osc		faible	Observateur d'oscillation	
		transitoire	SMZ3	Param-choc		grave	Information bumper robot	Appel Opérateur
	Complète		SMZ4	Perturbation forte			Retour arrière	Appel Opérateur
		permanent	SMZ5	Module HS	[NAV(C) ; UST(C)]	moyen	Watchdog activaton	Changement d'algorithme de suivi de chemin
CTL	Partielle	transitoire	CTL1	Erreur TR		ignorée	Watchdog activaton	
	Complète	permanent			SMZ(C)	grave		
			CTL2	Module HS			Watchdog activaton	

Table B.2 – Tableau AMDEC du suivi de chemin autonome SMZsansMCL

B.4 Mode de fonctionnement de suivi de chemin DVZ sans MCL

La table B.3 donne le résultat de l'application de la méthodologie proposé sur le mode de fonctionnement correspondant.

MODULE	MODE DEFAILLANCES		CAUSES DES DEFAILLANCES			SEVERITE	ACTIONS	
	classe	Type	Fautes		Modules liés fonctionnellement		Principe d'observation	Informations Recouvrement
			Identificateur	Information				
P3D(mod)	Partielle	transitoire	P3D-mod1	Tps_Com_USB		ignorée	Watch dog variable	
			P3D-mod2	Perte donnée				
			P3D-mod3	Erreur TR			Watchdog activation	
	Complète	transitoire			CTL(C)	grave		
		permanent	P3D-mod4	Batterie		farale	Information niveau de batterie robot	Arret
			P3D-mod5	Module HS			Watchdog activation	Arret
			P3D-mod6	USB HS			Watchdog communication	Arret
P3D(roue)	Partielle	transitoire	P3D-roue1	3ol glissant		ignorée	Perturbation ponctuelle non détectée	
		permanent	P3D-roue2	Incertitude odométres			Perturbation permanente	
	Complète	permanent			P3D(mod)(C)	farale		Arret
			P3D-roue4	Roue bloqué			Banque filtre de kalman	Arret
			P3D-roue5	Moteur HS			Banque filtre de kalman	Arret
P3D(sonar)	Complète	transitoire	P3D-sonar1	Rafraichissement		moyen	Watchdog activation	MCL non utilisable
		permanent			P3D(mod)(C)			
			P3D-sonar2	Sonar HS			Watchdog - vivacité des valeurs	Aucun sonar
UST	Partielle	transitoire		Erreur TR		ignorée	Watchdog activation	
	Complète	permanent	UST2		P3D(sonar)(C)			Mode sans US
			UST3	Module HS			Watchdog activation	
ODO	Partielle	transitoire			P3D(roue)(P)	faible		
	Complète	permanent	ODO1	Erreur TR		ignorée	Watchdog activation	
			ODO2	Module HS	P3D(roue)(C)	farale		Arret
NAV	Partielle	transitoire	NAV1	Erreur TR		ignorée	Watchdog activation	Arret
	Complète	transitoire			ODO (C)	grave	Watchdog activation	
		permanent	NAV3	Module HS		farale		
GUI	Partielle	transitoire	GUI1	Erreur TR		ignorée	Watchdog activation	Arret
			GUI2	Param - Orc		faible	Observateur d'oeuillation	
	Complète	transitoire	GUI3	Perturbation zone			Retour arrière	Appel Opérateur
		permanent	GUI4	Module HS	INAV(C) - UST(C)	moyen	Watchdog activation	
CTL	Partielle	transitoire	CTL1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent			GUI (C)	grave		
			CTL2	Module HS			Watchdog activation	
DVZ	Partielle	transitoire	DVZ1	Erreur TR		ignorée	Watchdog activation	
		transitoire	DVZ2	Param-choc		grave	Information bumper robot	Appel Opérateur
	Complète	permanent			CTL (C)			
			DVZ3	Module HS		grave	Watchdog activation	

Table B.3 – Tableau AMDEC du suivi de chemin autonome DVZsansMCL

B.5 Mode de fonctionnement de suivi de chemin sans sonar

Dans la table B.4 nous retrouvons le résultat de l'analyse AMDEC du mode de fonctionnement de suivi de chemin sans capteur ultrasons.

MODULE	MODE DEFAILLANCES		CAUSES DES DEFAILLANCES			SEVERITE	ACTIONS	
	classe	Type	Fautes		Modules liés fonctionnellement		Principe d'observation	Informations Recouvrement
			Identificateur	Information				
P3D(mod)	Partielle	transitoire	P3D-mod1	Tps_Com_USB		ignorée	Watch dog variable	
			P3D-mod2	Perte donnée				
			P3D-mod3	Erreur TR			Watchdog activation	
	Complète	transitoire			CTL(C)	grave		
		permanent	P3D-mod4	Batterie		fatale	Information niveau de batterie robot	Arret
			P3D-mod5	Module HS			Watchdog activation	Arret
			P3D-mod6	USB HS			Watchdog communication	Arret
P3D(roue)	Partielle	transitoire	P3D-roue1	Sol glissant		ignorée	Perturbation ponctuelle non détectée	
		permanent	P3D-roue2	Incertitude odométres			Perturbation permanente	
	Complète	permanent			P3D(mod)(C)	fatale		Arret
			P3D-roue4	Roue bloqué			Banque filtre de kalman	Arret
			P3D-roue5	Moteur HS			Banque filtre de kalman	Arret
ODO	Partielle	transitoire			P3D(roue)(P)	faible		
			ODO1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent			P3D(roue)(C)	fatale		Arret
			ODO2	Module HS			Watchdog activation	Arret
NAV	Partielle	transitoire	NAV1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent			ODO (C)	grave		
			NAV3	Module HS		fatale	Watchdog activation	Arret
GUI	Partielle	transitoire	GUI1	Erreur TR		ignorée	Watchdog activation	
			GUI2	Param - Onc		faible	Observateur d'oscillation	
	Complète	transitoire	GUI3	Perturbation forte			Retour arrière	Appel Opérateur
		permanent			(NAV(C) ; UST(C))	moyen		
			GUI4	Module HS			Watchdog activation	
CTL	Partielle	transitoire	CTL1	Erreur TR		ignorée	Watchdog activation	
	Complète	permanent			GUI (C)	grave		
			CTL2	Module HS			Watchdog activation	

Table B.4 – Tableau AMDEC du suivi de chemin autonome GUI

RÉSUMÉ

Titre : Proposition d'une architecture de contrôle adaptative pour la tolérance aux fautes
Résumé : Les architectures logicielles de contrôles sont le centre névralgique des robots. Malheureusement les robots et leurs architectures souffrent de nombreuses imperfections qui perturbent et/ou compromettent la réalisation des missions qui leurs sont affectés. Nous proposons donc une méthodologie de conception d'architecture de contrôle adaptative pour la mise en œuvre de la tolérance aux fautes. La première partie de ce manuscrit propose un état de l'art de la sureté de fonctionnement, d'abord générique avant d'être spécifié au contexte des architectures de contrôle. La seconde partie nous permet de détailler la méthodologie proposée permettant d'identifier les fautes potentielles d'un robot et d'y répondre à l'aide des moyens de tolérance aux fautes. La troisième partie présente le contexte expérimental et applicatif dans lequel la méthodologie proposée sera mise en œuvre et qui constitue la quatrième partie de ce manuscrit. Une expérimentation spécifique mettant en lumière les aspects de la méthodologie est détaillée dans la dernière partie.

Mots-clés : Architecture de contrôle, tolérance aux fautes, sureté de fonctionnement, interaction Homme-robot, AMDEC.

ABSTRACT

Title : Proposition of an adaptive control architecture for fault tolerance
Abstract : The software control architectures are the decisional center of robots. Unfortunately, the robots and their architectures suffer from numerous flaws that disrupt and / or compromise the achievement of missions they are assigned. We therefore propose a methodology for designing adaptive control architecture for the implementation of fault tolerance. The first part of this thesis proposes a state of the art of dependability, at first in a generic way before being specified in the context of control architectures. The second part allows us to detail the proposed methodology to identify potential errors of a robot and respond using the means of fault tolerance. The third part presents the experimental context and application in which the proposed methodology will be implemented and described in the fourth part of this manuscript. An experiment highlighting specific aspects of the methodology is detailed in the last part.

Keywords : control architecture, fault tolerance, dependability, Human robot interaction, FMCEA

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier « LIRMM »
UMR CNRS / Université Montpellier II
161, rue Ada - 34095 Montpellier Cedex 05 - France.