Automatisation de la mise au point de lois de contrôle de systèmes cyber-physiques évoluant en environnement incertain

Hamza El Baccouri

THESE DE DOCTORAT DE

L'UNIVERSITE
DE BRETAGNE OCCIDENTALE

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

Par
Hamza EL BACCOURI

Automatisation de la mise au point de lois de contrôle de systèmes cyber-physiques évoluant en environnement incertain

Thèse présentée et soutenue à Brest, le 17 décembre 2020
Unité de recherche : Lab-STICC UMR CNRS 6285

Rapporteurs avant soutenance :

Charles LESIRE-CABANIOLS	Directeur de recherche, HDR ONERA
Emmanuel GROLLEAU	Professeur des Universités, ISAE - ENSMA

Composition du Jury :

Président du Jury :	Xavier BRUN	Professeur des Universités, INSA Lyon
Examinateurs	Karen GODARY-DEJEAN	Maître de conférences, Université de Montpellier
	Charles LESIRE-CABANIOLS	Directeur de recherche, HDR, ONERA
	Emmanuel GROLLEAU	Professeur des Universités, ISAE-ENSMA
Dir. de thèse :	Jean-Philippe BABAU	Professeur des Universités, UBO
Encadrant :	Goulven GUILLOU	Maître de conférences, UBO

Table des Matières

Liste des figures

Liste des Tables

Chapitre 1

Introduction

1.1 Contexte de l'approche

Dans cette thèse, nous nous intéressons au contrôle de système cyber-physiques évoluant en environnement incertain. Les avions, les engins spatiaux, les bateaux ou encore les drones sont des exemples typiques de cette catégorie de système.
Un système cyber-physique doit faire face à un niveau élevé d'incertitude. Son comportement dépend étroitement des évolutions de son environnement. De ce fait, le système de contrôle associé est spécifique au système cyber-physique et a pour objectif de réguler son comportement. Afin de réagir à divers situations rencontrées, le système de contrôle associé doit prévoir un certain nombre de stratégies adaptées. Si actuellement pour faire face à divers situations, les applications s'appuient sur des configurations pré-calculées, le système devrait pouvoir proposer des capacités d'adaptation à des contextes inconnus a priori.
Dans le but d'assurer l'adaptabilité et la sécurité du système contrôlé, l'application de contrôle a besoin de se faire une image de l'environnement. Le système de contrôle utilise des capteurs pour obtenir des informations sur le processus à contrôler ainsi que sur son environnement physique. D'autre part, des actionneurs permettent d'appliquer les commandes déterminées par le système de contrôle sur le système cyber-physique. La mise en oeuvre est souvent spécifique à chaque domaine à cause des contraintes liées à l'environnement ainsi qu'à celles liées aux matériels. En particulier, le réglage de la loi de contrôle est lié aux capteurs et actionneurs, à leur technologie et à leurs performances temporelles (fréquence, retard).
Généralement, les lois de contrôles font intervenir plusieurs modes de fonctionnement dépendant du contexte. Pour chaque mode, la qualité de contrôle de ces systèmes repose sur la qualité du paramétrage des lois de contrôle. Il n'est pas évident de définir a priori quel paramétrage doit être utilisé. De nombreux paramétrages doivent alors être testés et ajustés.
Lorsqu'on ne dispose pas de modèle (une équation différentielle ou une fonction de transfert par exemple) du système, seuls des tests permettent d'évaluer la qualité du contrôle. La grande variété des situations possibles rend caduque l'espoir d'effectuer des tests réels, sans compter d'autres aspects tels que la sécu-

rité (du système lui-même et de son environnement) ou encore le coût de mise en oeuvre. La simulation, bien que plus ou moins proche de la réalité, permet de contourner tous ces problèmes. Pour la simulation, le concepteur du système cyber-physique crée un modèle exécutable du processus contrôlé et de son environnement, de la plate-forme matérielle (capteurs et actionneurs) et du contrôleur. Cette phase de simulation est généralement connue sous le nom de (*Model In the Loop*) (MiL). Les modèles contiennent généralement de nombreux paramètres qui déterminent la dynamique globale du système. En particulier, le contexte du contrôleur est composé de tous les éléments (la plate-forme matérielle et l'environnement) qui ne sont pas contrôlés par le système (c'est-à-dire l'environnement physique dans lequel le système évolue). Le but principal de la simulation est de tester et de régler le contrôleur en tenant compte des différents contextes. Les observations accumulées lors de la simulation approfondissent aussi la compréhension du fonctionnement du système. Mais, compte tenu du nombre requis de simulations et la quantité de données collectées, il est désormais nécessaire de fournir des outils de mise au point adéquats.

1.2 Problématiques

Le principal défi lors de la mise au point des contrôleurs associés aux systèmes cyber-physiques évoluant en environnement incertain vient de la multiplicité des contextes à prendre en compte et du fait que chaque contexte est spécifique. Un réglage de contrôle correct (correct en termes d'exigences de qualité de contrôle) pour un contexte spécifique peut ne pas être adapté à un autre. La modification d'un seul paramètre peut conduire à une modification radicale de la dynamique du système. Il faut donc établir des valeurs adéquates des paramètres de la loi de contrôle pour chaque contexte, ce qui peut s'avérer long et fastidieux. L'espace de recherche est un produit cartésien des valeurs de chaque paramètre. Étant donné que les paramètres sont généralement des réels (de type réel), l'espace de recherche conduit généralement à une explosion combinatoire (grand nombre de situations possibles, l'espace de recherche est large, complexe et multidimensionnel). Dans ce contexte, nous avons besoin d'une stratégie précise, automatique et systématique pour explorer l'espace défini par les contextes et les paramètres du contrôleur.
De cette exploration, pour chaque contexte, il doit être possible de déterminer un réglage adéquat (i.e., un paramétrage correct en termes de qualité de contrôle) du contrôleur. Néanmoins, il n'est pas approprié de changer le réglage de contrôle suite à chaque changement du contexte (changement de l'environnement, dans l'état des capteurs ou des actionneurs). Cela risque d'amener le système vers l'instabilité. En effet, ce dernier a besoin d'un temps de stabilisation suite à chaque changement de réglage. Il est donc impératif de pousser l'exploration dans le but de trouver des réglages qui peuvent convenir pour différents contextes, ces réglages couvrant plusieurs contextes seront appelés modes. Comme il n'est pas possible de simuler toutes les situations, ceci conduit à poser la question suivante : Quel est le réglage à adopter pour les contextes non simulés?
Pour récapituler, dans environnement naturel fortement perturbé, il est primor-

dial de répondre aux trois questions suivantes :

- Comment paramétrer la loi de contrôle en fonction des différents contextes?
- Quels sont les modes de fonctionnement pertinents?
- Quels paramétrages pour les contextes inconnus ?

1.3 Contribution

Pour répondre à ces questions, l'approche proposée s'appuie sur la simulation, accompagnée des techniques de réduction de dimensionnalité, de clustering et de prédiction pour définir des réglages adéquats des paramètres de contrôle. L'idée de base consiste à exposer le contrôleur à différents contextes, c'est-à-dire à des états différents de l'environnement, des capteurs, des actionneurs, pour trouver des réglages du contrôleur qui conviennent à un maximum de contextes. Pour pouvoir faire de la simulation le système est tout d'abord modélisé en respectant un style architectural prédéfini (modélisation à base de composants). Le modèle doit en particulier être capable de prendre en compte les éléments standards d'une boucle de contrôle. De plus, la variabilité du comportement du système cyber-physique et de son environnement doit pouvoir passer par la définition de paramètres variables numériques comme attributs des composants. Ces paramètres sont classés en deux catégories :

- les paramètres contextuels qui sont liés aux capteurs, actionneurs et au système contrôlé et qui décrivent les changements dans l'environnement
- les paramètres de contrôle qui apparaissent dans les lois de contrôle.

À partir de ce modèle, des politiques de balayage sont proposées pour conduire la simulation en explorant différents contextes. Des stratégies de *binning* sont mises en oeuvre dans le but de répartir les différentes configurations testées en diverses classes de qualité de contrôle *QoC*.
Les principales contributions de cette thèse sont :

- Pour faire face au problème de l'explosion combinatoire, la réduction de dimensionnalité, et plus précisément la sélection des caractéristiques est utilisée afin de réduire le nombre de situations à explorer en sélectionnant les paramètres contextuels les plus pertinents pour le problème considéré, i.e., qui contribuent le plus à l'attribution d'une configuration (un contexte et un réglage de contrôle à une classe de *QoC*.
- En se concentrant sur les paramètres sélectionnés, nous effectuons un regroupement (clustering) des contextes possédant les mêmes réglages de contrôle.
- Enfin, nous proposons de mettre en place un module de prédiction pour attribuer des réglages de contrôle adéquats pour des contextes non-simulés en se référant aux données de simulation collectées.

L'approche proposée peut s'appliquer d'une manière itérative pour définir les limites d'adaptabilité du contrôleur.

1.4 Plan du mémoire

Ce manuscrit de thèse est organisé en cinq chapitres qui après l'introduction couvrent l'état de l'art, la contribution, l'évaluation de la proposition et une conclusion générale.
Dans le deuxième chapitre, nous présentons un état de l'art sur les différentes thématiques abordées tout au long de cette thèse. Ce chapitre se divise en cinq sous-chapitres. Nous commençons par présenter des généralités sur la modélisation à base de composants des systèmes cyber-physiques avec un intérêt particulier pour l'utilisation des composants à des fins de s imulation. Ensuite, nous discutons des techniques de synthèse des régulateurs dédiés aux systèmes cyber-physiques, en mettant l'accent sur la régulation *PID* tout en présentant deux techniques qui facilitent la mise au point de ces régulateurs, à savoir la méthode *Ziegler-Nichols* et la méthode *Root Locus*. Par la suite, nous abordons la thématique de la réduction de dimensionnalité et plus spécifiquement de la sélection de caractéristiques. Ensuite, nous exposons les concepts clefs des techniques de clustering comme présentés dans la littérature. Enfin nous concluons le chapitre par la présentation des principes et des intérêts de l'utilisation des réseaux de neurones pour la prédiction.
Le troisième chapitre est consacré à notre contribution pour faciliter la mise au point des contrôleurs associés aux systèmes cyber-physiques évoluant en environnement perturbé. Ce chapitre se compose de trois sous-chapitres. Nous commençons par la présentation des principes généraux de notre approche. Nous présentons d'abord les principes utilisés pour le réglage des paramètres calculé via des simulations et l'utilisation d'algorithmes de sélection de caractéristiques et de clustering. Dans un deuxième temps, un paramétrage en ligne est proposé par un module de prédiction à base de réseau de neurones. Enfin, l'ensemble des outils qui permettent de systématiser et d'automatiser la mise au point sont présentés.
Dans le quatrième chapitre nous abordons la validation et l'évaluation de la proposition. Nous commençons par la comparaison des performances de l'approche proposée avec des méthodes classiques en théorie du contrôle. Ensuite, nous présentons les résultats de l'application de l'approche sur un problème de type *Leader-Follower*.
Enfin le dernier chapitre conclue sur un rappel de la problématique et des contributions, et expose les perspectives futures à cette étude.

Chapitre 2

État de l'art

2.1 Introduction

Dans ce chapitre, nous commençons par définir ce que nous considérons comme un système cyber-physique. Puis, nous discutons de l'importance de modéliser ces systèmes et d'explorer diverses situations via la simulation (avant de passer aux tests dans un environnement réel).

2.2 Systèmes cyber-physiques

Le terme *Système Cyber-Physique* est apparu aux alentours de l'année 2006 (au milieu des années 2000). Il est généralement attribué à *Helen Gill* du National Science Foundation [CAR16]. La racine de ce terme provient de « cybernétique », dont l'origine est attribuée au mathématicien américain *Norbert Wiener*. Selon *Wiener*, la cybernétique est la combinaison de la commande et de la communication. Sa vision de la commande est basée sur le contrôle en boucle fermée, où des capteurs surveillent le processus physique piloté par la commande à travers les actionneurs.
La première définition que l'on peut trouver des systèmes cyber-physiques a été donnée par [Lee06]. Cette définition peut être retranscrite ainsi : «**Les CPS intègrent des processus physiques et computationnels. Des ordinateurs et réseaux embarqués surveillent et contrôlent les processus physiques, généralement avec des boucles de rétroaction où les processus physiques affectent les calculs et vice versa. En d'autres mots, les CPS utilisent des calculs et de la communication profondément intégrées et interagissant avec les processus physiques afin de produire de nouvelles capacités du système**».
D'autres définitions ont été depuis proposées [Raj+10; BG11; Cen13], mais la définition de [Mon14] permet de les synthétiser : « **Les CPS sont des systèmes formés d'entités collaboratives, dotées de capacité de calcul, qui sont en connexion intensive avec le monde physique environnant et les phénomènes s'y déroulant** ».

2.2.1 Modélisation des systèmes cyber-physiques

La mise au point et le réglage des contrôleurs associés aux systèmes cyber-physiques évoluant en environnement incertain est un processus complexe. Ceci s'explique par le fait que le comportement de ces systèmes dépend étroitement de leur environnement (dégradation, incertitude, changement ...).
Il apparaît nécessaire de modéliser le système cyber-physique dans sa globalité en prenant en considération sa structure, son dispositif de commande (contrôleur) et son environnement [Mar97]. Ces modèles sont utilisés pour remplacer le matériel réel dans les phases de test et de réglage. En partant de ces modèles, il est possible d'explorer diverses situations et donc d'anticiper le comportement du système cyber-physique avant de passer aux tests en environnement réel, sur une plateforme réelle.
Pour la modélisation, plusieurs approches sont envisageables. Parmi celles-ci on peut citer la modélisation adoptée par les automaticiens qui consiste à écrire les équations différentielles qui régissent le système et à tenter de les résoudre analytiquement à l'aide d'un outil numérique [BS05]. Dans cette thèse, on s'intéresse plutôt aux modèles informatiques de systèmes cyber-physiques à base de composants [Szt07] tels que *SysML* [FMS08], *AADL* [FG12] et *Capella* [BR18]. Dans ce chapitre, nous étudions les caractéristiques des composants et leurs capacités à modéliser un système cyber-physique dans un but de simulation.

2.2.1.1 Définition

Nous commençons par définir le terme *composant* dans le contexte de l'ingénierie logicielle. Une des définitions les plus communément acceptées du composant est celle proposée par *Clemens Szyperski* dans [Szy02] : « ***A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition*** ».
Par la suite, nous nous appuyons sur la synthèse présentée dans [Jea07] pour définir les concepts clés des composants logiciels.

2.2.1.2 Apports des composants

La décomposition d'un système cyber-physique en composants est très utilisée pour la modélisation des systèmes. Cette décomposition est bénéfique pour de nombreuses raisons dont on peut citer :

- **La séparation des préoccupations** : la décomposition du système permet aux différents intervenants sur un système cyber-physique (informaticiens, électroniciens, mécaniciens...) de se focaliser sur leur coeur de métier, qui est modélisé par un ou plusieurs composants [HL95; Ern03].

- **La réutilisation** : la modélisation du système est simplifiée par la réutilisation des bibliothèques de composants spécialisées existantes (modéliser un système revient à assembler des éléments existants), ce qui réduit l'effort nécessaire à la création de nouveaux systèmes.

- **La facilité d'évolution** : dans le cas ou les exigences ou les objectifs du système évoluent, seuls les composants impactés par les évolutions doivent être révisés, affinés et actualisés en conséquence.

- **La gestion de l'hétérogénéité** : les systèmes cyber-physiques se composent d'éléments hétérogènes par leur nature (capteurs, actionneurs, environnements physiques, logiciels, modèles...), par la nature des données qu'ils échangent, par leur caractéristique temporelle (synchrone, asynchrone, temps réel ou simulé) [BBS06] , mais également par leur granularité. L'approche par composants peut permettre une standardisation des interfaces et la définition d'architectures pour faciliter l'intégration de différents éléments et pour la construction de systèmes hétérogènes complexes [AS11].

Dans ce qui suit, on retient qu'un composant désigne une partie du modèle du système cyber-physique et de son environnement et représente une brique de base du modèle.

2.2.1.3 Caractéristiques des composants

2.2.1.3.1 Structure et comportement

A chaque composant est associé un comportement. Le comportement peut résulter de l'activité du composant (s'il en dispose), d'une réaction à un événement (nouvelle valeur sur une entrée, synchronisation externe...), ou bien d'une combinaison de ces deux éléments. On distingue deux manières de décrire le comportement d'un composant [DCL08; Rom+10] :

- Les composants dit *primaires* possèdent un comportement décrit dans un langage spécifique (langage de programmation, langage machine). Dans la littérature, la spécification du comportement de ces composants est réparti en quatre catégories non exclusives [Jea07] :
  - spécification du comportement par un (ou plusieurs) exécutable(s) binaire(s) ou sous forme de byte code
  - spécification du comportement à l'aide d'un (ou plusieurs) code(s) sources exprimés dans un langage de programmation classique
  - spécification à l'aide d'un langage formel
  - spécification de paramètres et ou de propriétés permettant de contrôler et/ou caractériser le comportement du code. On retrouve ici des paramètres de configuration du code ainsi que le cycle de vie du composant.

- les composants dit *composites* qui englobent des composants. Les composants *composites* peuvent soit disposer d'un comportement propre soit encapsuler le comportement de leurs constituants. Leur utilisation permet de créer une hiérarchie de composants. Un composant composite est donc d'un niveau d'abstraction supérieur aux composants qui le constituent.

La modélisation du système cyber-physique et de son environnement se fait en assemblant plusieurs composants entre eux pour former ce qu'on appelle une *configuration* (un système cohérent). Il s'agit principalement d'organiser les composants en groupes au sein d'un ensemble cohérent.

2.2.1.3.2 Communication des composants

Les différents composants qui constituent le système cyber-physique et son environnement peuvent communiquer entre eux dans le but d'échanger des informations nécessaires à leur réalisation [Szy02]. Dans le domaine visé, le paradigme de flux de données est généralement adopté car il est proche de la vision système des automaticiens. Chaque composant dispose d'entrées et de sorties. Les entrées correspondent aux données nécessaires à l'exécution du composant. En contrepartie, les sorties correspondent aux données produites par le composant. La communication entre les différents composants est effectuée en reliant les entrées aux sorties à l'aide de connecteurs [SVK97]. Ce type de communication suit soit le paradigme *publish/subscribe* [And+05] où le producteur de la donnée publie une valeur sur sa sortie et tous les abonnés reçoivent cette valeur, soit le paradigme *request/reply* [San+17] où le consommateur demande une donnée auprès du producteur puis la reçoit.

2.2.1.3.3 Exécution des composants

Les modèles des systèmes cyber-physiques sont composés d'éléments qui évoluent au cours du temps. Une modélisation du temps doit être incluse dans les composants du système pour pouvoir les exécuter [AGS02]. Généralement, le temps est représenté de manière discrète. Pour s'assurer que les composants puissent échanger des informations, il faut définir une stratégie d'exécution (activation des composants et synchronisation).

2.2.1.3.4 Paramétrage des composants

On associe aux composants un ensemble d'attributs. Les attributs permettent de configurer le comportement du composant [NBP13], de contrôler le cycle de vie du composant et de connaître son état. On distingue également les attributs qui ont pour but de définir les conditions initiales du composant et les périodes d'activation de ce dernier. Dans cette thèse on s'intéresse particulièrement aux attributs qui permettent de configurer l'évolution de l'état du composant. On parle alors de paramètres. En faisant varier les valeurs des paramètres du composant *environnement*, il est possible d'explorer diverses situations que le système cyber-physique peut rencontrer.

2.2.1.4 Discussion sur les modèles à composants

De nombreux frameworks de modèles à composants offrent des outils pour faciliter la modélisation des systèmes cyber-physiques tout en profitant des dif-

férents avantages mentionnés auparavant [BS10; Bro+05]. Parmi les frameworks les plus utilisés, outillés et documentés, on peut citer *BRICS* [Bru+13], *Orocos* [BSK03; Bru01], *SmartSoft* [SW99] et *OpenRTM* [ASK08]. Ces frameworks mettent à disposition des outils pour créer des composants et/ou de réutiliser des bibliothèques existantes de composants. Les composants sont généralement utilisés à des fins de conception. Les frameworks à base de composants fournissent aux développeurs des systèmes cyber-physiques un ensemble de lignes directrices, de méta-modèles et d'outils pour structurer autant que possible la conception du système. Les composants peuvent être exploités aussi pour des raisons de validation [Gol+02] ce qui est le cas, par exemple pour *Cheddar* [Sin+04b; Sin+04a]. Dans notre étude, les composants sont utilisés à des fins de simulation [LL02; ZMK19; Lav19]. Lors de la simulation, pour modéliser la variabilité du comportement du système cyber-physique, il est possible d'agir sur :

- la structure : elle peut être modifiée dynamiquement en ajoutant ou en supprimant des composants spécifiques
- les connections : elles peuvent être crées ou supprimées dynamiquement
- les paramètres : modification de la valeur des attributs des composants

Dans cette thèse, on ne considère que la troisième approche, et ce pour des attributs numériques.

2.2.2 Simulation des systèmes cyber-physiques

Dans notre travail, la modélisation est utilisée à des fins de simulation. La simulation exige qu'on dispose d'un environnement logiciel pour exécuter la loi de contrôle, mais également de modèles du système cyber-physique et notamment de l'environnement dans lequel il évolue. Intrinsèquement, simuler un système a pour objectif d'imiter son comportement c'est-à-dire de l'exécuter virtuellement comme le montre la figure 2.1 [Mar97]. En d'autres termes, la simulation s'intéresse à reproduire synthétiquement le comportement des systèmes étudiés pour en capturer les propriétés essentielles qui pourront ensuite être analysées à moindre coût sur le simulateur [ZPK00].
Dans notre étude, la simulation est utilisée à des fins d'exploration visant à reproduire le comportement du système cyber-physique pour différentes conditions environnementales. Dans ce cadre, simuler revient donc à produire, à chaque exécution, un exemple du comportement du système en fonction de :

- ses données d'entrée
- son état initial
- ses conditions environnementales

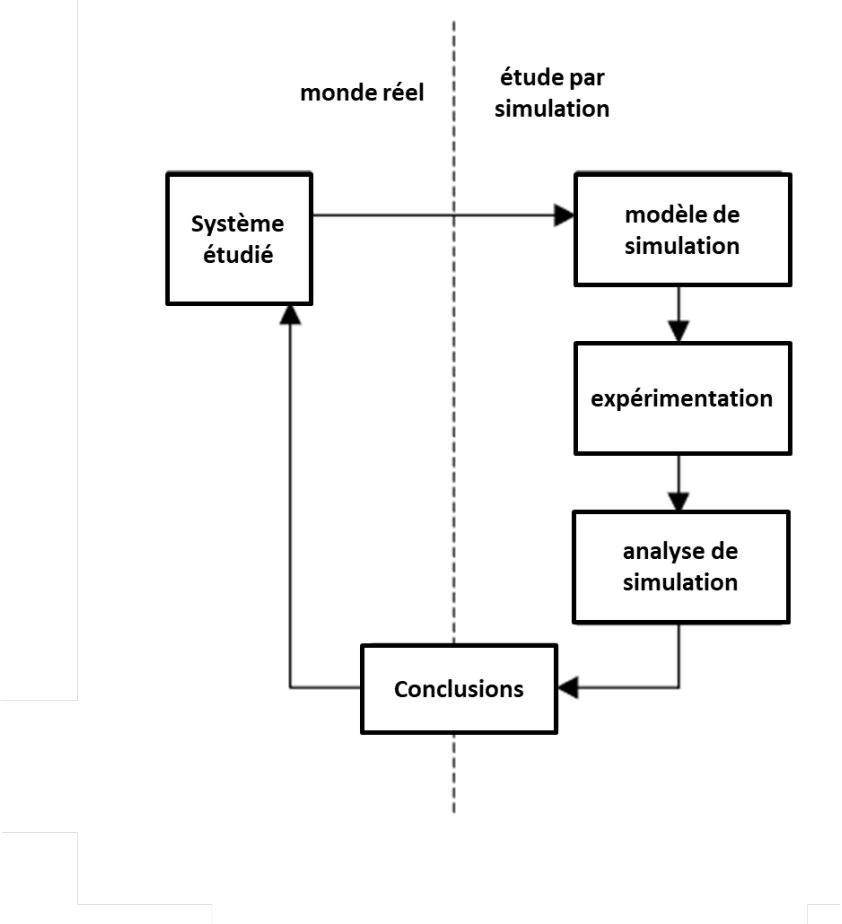

Figure 2.1: Éléments d'une exploration par simulation [Mar97]

D'une manière générale, les concepteurs de simulateurs arbitrent entre degré de réalisme du simulateur et de complexité en temps de calcul des exécutions. Toutefois, il est nécessaire de préciser que le degré de réalisme du simulateur est le garant pour obtenir des résultats fidèles à la réalité.

2.2.2.1 Composant et Simulation

La mise en place d'un simulateur de systèmes cyber-physiques est facilitée par l'adoption de la modélisation à base de composants. En particulier, l'utilisation d'un modèle à composants permet de structurer le simulateur. Les composants désignent des éléments du système à simuler et sont implémentés par des morceaux de code source ou de fichiers exécutables qui peuvent être assemblés et déployés dans le but de créer le simulateur. Chaque composant décrit un aspect spécifique du système et peut-être facilement configuré, modifié, changé ou réutilisé, sans impacter la description du système complet.
La souplesse des composants peut permettre de simuler le système modélisé, à différents niveaux : soit l'ensemble du système est modélisé par des composants logiciels lors de la simulation, soit certaines parties physiques du système (cap-

teurs, contrôleur, système mécanique...) sont intégrées en tant que composants lors de la simulation. Il est envisageable de remplacer progressivement les composants de simulation par des composants réels pour obtenir finalement le système complet comme expérimenté et illustré dans [Lav19].
La simulation d'un système comportant plusieurs éléments pouvant s'exécuter en parallèle, répartis en diverses couches et éventuellement distribués sur plusieurs plateformes implique de relever divers défis. Il faut pouvoir exécuter chaque élément, les faire communiquer entre eux, gérer leur synchronisation et leur ordonnancement. Selon l'architecture du simulateur, les différents composants peuvent être fortement hétérogènes en termes de nature (composant logiciel, système physique, modélisation complexe, agent de simulation indépendant...) et de plateforme d'exécution (système distribué).
Pour faire face à ces difficultés, l'utilisation du modèle à composants est complété par l'adoption d'une architecture pour la simulation. Cette architecture permet de définir la manière d'assembler et d'exécuter les composants. C'est ce que l'on appelle un style architectural [Fie00]. Ce style doit être compatible avec le domaine d'application du système cyber-physique et doit tenir compte des spécificités liées à la simulation. Les architectures dédiées à la simulation des systèmes cyber-physiques présentent des similitudes. Classiquement, on utilise la simulation à événement discret. L'exécution d'un composant suit le paradigme RunToCompletion [Cho+02]. La configuration se fait via la modification des attributs des composants.
Dans un premier lieu, on y trouve un découpage en blocs de simulation indépendants qui partagent des données et des informations. Dans un deuxième lieu, elles proposent des interfaces standardisées permettant d'intégrer des éléments hétérogènes au sein d'une même simulation. L'inconvénient majeur de la majorité de ces architectures réside dans le fait qu'elles ne proposent pas de stratégie ou de mécanisme permettant la gestion de l'environnement dans lequel le système cyber-physique évolue.
De nombreux frameworks dédiés à la simulation des systèmes cyber-physiques existent dans la littérature. Parmi ceux-ci on peut citer : *FMI* [Gol+02] et *ROS/GAZEBO* [KH04]. Pour le développement d'un environnement de simulation du système cyber-physique, nous avons dans la thèse choisi d'utiliser *Simulink* [Esh16] et le framework *AMSA* [Lav+16].

2.2.2.2 Simulink

Simulink[KH04] est un outil de modélisation et de simulation du comportement des systèmes intégré à l'environnement de programmation scientifique *MATLAB* [Nej+19]. *Simulink* permet de développer des systèmes de contrôle et de tester ses performances [MNB17]. L'établissement du modèle de système (ou d'une de ses fonctions) est effectué moyennant des "blocs" présents en librairie (il suffit de rassembler les "blocs" et de les connecter). *Simulink* offre des facilités pour paramétrer les blocs, voire même personnaliser ces derniers. Placer des "instruments de visualisation" en sortie du modèle rend possible la visualisation des résultats. *Simulink* est utilisé par plus de 60% des ingénieurs pour la simulation de systèmes cyber-physiques [Zhe+17]. Son utilisation est bénéfique dans la

mesure ou son environnement graphique rend le processus de modélisation du système cyber-physique rapide et facile. Par la suite, la simulation du modèle permet d'évaluer son comportement à des fins de validation ou d'évaluation de l'influence de certains paramètres. La communauté *Simulink* est très importante et de ce fait, de nombreuses librairies sont proposées pour simuler les systèmes cyber-physiques. *Simulink* offre également des outils nécessaires à l'analyse et à l'évaluation des performances du *CPS* (évaluer la dynamique de ces systèmes). De plus, le générateur de code peut faciliter l'implémentation du système [Mat+19]. A notre connaissance et malgré les facilités offertes par *Simulink* pour la simulation, ce dernier ne permet pas de mener l'exploration exhaustive pour la découverte de modes de fonctionnement. Il n'existe pas de stratégie pour automatiser la simulation et le test de plusieurs configurations (différents paramétrages du contrôleur et conditions environnementales).

2.2.2.3 AMSA

Le framework *AMSA* [LGB18], a pour objectif de faciliter la mise en place de simulateurs pour les contrôleurs de systèmes cyber-physiques. *AMSA* offre des outils pour définir et structurer des modèles d'application des simulation. Ce framework propose un style architectural permettant la construction d'un assemblage de composants tout en respectant des principes liés à un domaine d'application [Lav19]. Ceci permet également de faciliter l'intégration de composants hétérogènes dans un simulateur. Le framework repose sur un modèle générique de simulation à base de composants paramétrables. Le modèle reprend les principes généraux définis par les modèles à composants pour les systèmes embarqués temps-réel tels que l'échange de données, composites sous formes de blocs, la fréquence d'activation, run-to-completion. Pour la simulation, *AMSA* reprend deux principes définis par le standard de simulation *FMI* [Gol+02] : interfaces communes à base de méthodes d'initialisation *Init()* et d'avancement pas à pas *doStep()*. Des paramètres sont associés aux composants. Ces paramètres sont utilisés par un générateur de code à base d'*Acceleo* [Del11] pour configurer les composants lors de la simulation. De plus, *AMSA* offre la possibilité de définir des scénarios basés sur des événements permettant de configurer la simulation. Ceci facilite le contrôle de l'évolution d'une simulation et rend possible l'exploration de divers contextes et comportements. En automatisant certaines tâches comme l'exploration paramétrique ou la production de critères de comparaison des simulations, *AMSA* permet de tester différentes situations (possibilités de balayage de scénarios). Finalement, ce framework a été utilisé pour mettre au point une loi de contrôle pour l'adapter au modèle analytique de voilier [Lav19].

2.2.2.4 Bilan

La modélisation à base de composants et la paramétrisation des composants combinées avec l'adoption d'un style architectural permet de faciliter la simulation de diverses situations que le système cyber-physique pourrait rencontrer. Le style architectural joue un rôle clé dans l'assemblage des

composants. Dans cette thèse, nous utilisons *Simulink* et le framework *AMSA*. *Simulink* offre toutes les briques pour simuler un système cyber-physique. *AMSA* propose un style architectural pour faciliter la modélisation du simulateur d'un système cyber-physique. De plus, ce framework propose de définir des scénarios de simulation sous la forme d'une succession d'évènements de paramétrage. Ceci rend possible l'exploration de diverses situations. Néanmoins, l'existence de nombreux paramètres nécessite d'automatiser l'évaluation des simulations via la modélisation de scénarios. L'idée est d'évaluer la qualité du contrôle pour différentes conditions environnementales ce qui exige l'élaboration de plusieurs simulations et le test de différentes configurations.

2.3 Régulation PID

Il existe différents types de régulateurs pour les systèmes cyber-physiques. Parmi ceux-ci, nous nous concentrons sur celui le plus utilisé dans l'industrie qui est la régulation par *PID* (Proportionnelle/Intégrale/Dérivée) [Yu06; Åst06]. Le principe de régulation par PID en boucle fermée [WH99; Nis00] est donné dans la figure 2.2.

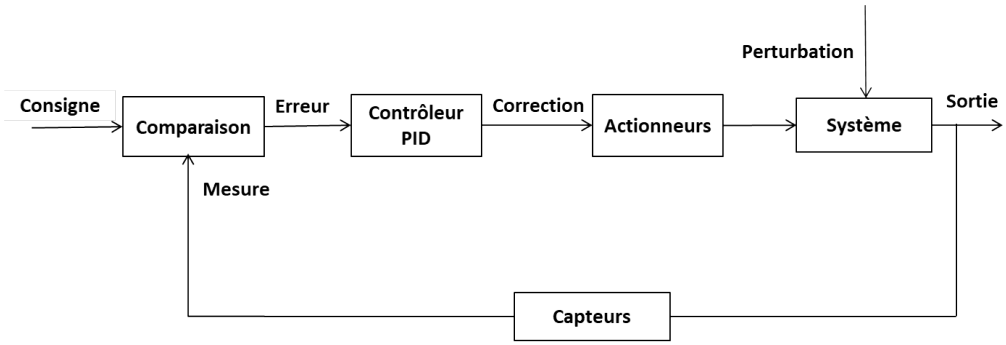

Figure 2.2: Éléments constituant une boucle de régulation PID en boucle fermée

Selon cette régulation, une *Consigne* est donnée et comparée à une *Mesure* issue de *Capteurs*. La différence entre la consigne et la valeur mesurée constitue ce qu'on appelle l'*Erreur*. L'objectif du régulateur *PID* est de minimiser l'*Erreur* qui subsiste entre la valeur souhaitée et la sortie du système contrôlé. La *Correction* calculée par PID pour atteindre la consigne est transmise aux *Actionneurs*. La *Correction* est la somme de trois termes : le terme proportionnel (P), le terme intégral (I) et le terme dérivé (D). Chaque terme comporte un coefficient multiplicateur appelé gain. Dans ce qui suit, ces gains seront notés K_P, K_I et K_D et donnent respectivement les poids des actions proportionnelles, intégrales et dérivées. La *Correction* appliquée se calcule avec la formule :

$$Correction = K_P \cdot Erreur + K_I \cdot \int Erreur\, dt + K_D \cdot \frac{d\quad Erreur}{dt}$$

L'évaluation de la qualité d'un régulateur est effectuée sur la base de la réponse indicielle du système en boucle fermée. La réponse indicielle d'un système est le signal en sortie lorsque l'entrée est un échelon unité (voir figure 2.3). Elle correspond au temps de réaction de la sortie par rapport à la consigne.

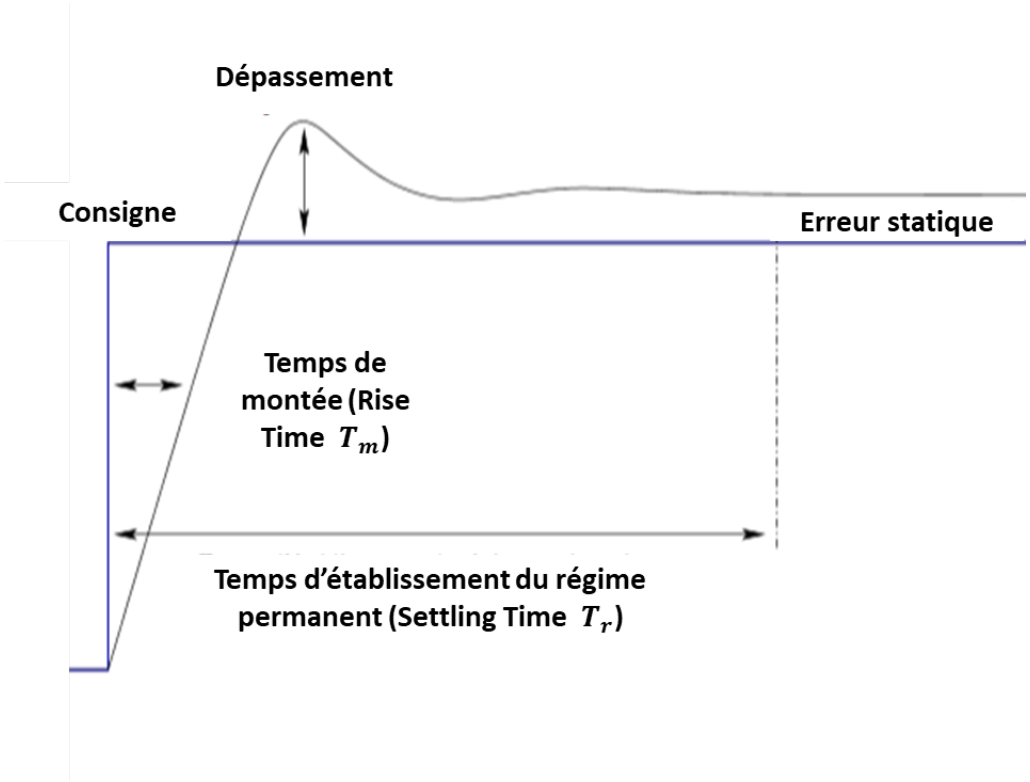

Figure 2.3: Réponse d'un système à un échelon

Les performances d'un système régulé s'établissent selon plusieurs critères.

- Le temps de montée (Rise time en anglais) noté T_m est le temps nécessaire à la réponse pour passer de 10% à 90% de la réponse en régime permanent lorsque le système est soumis à une entrée en échelon.
- Le temps de réponse T_r (Settling time en anglais) est le temps nécessaire pour que la sortie du système tombe à moins de 5% de la consigne de référence.
- L'erreur statique correspond à l'écart en régime permanent (c'est à dire lorsque le système est stabilisé) entre la réponse et la consigne du système.
- Le dépassement (Overshoot en anglais) représente l'écart entre la consigne et la valeur maximale atteinte. On exprime habituellement le dépassement en %.

La qualité d'un régulateur PID est généralement évaluée sur les critères de rapidité, stabilité, précision et robustesse [Jel13].

- La rapidité : la rapidité d'un système de régulation automatique peut être évaluée sur la base de sa réponse indicielle en boucle fermée. Les critères standards de rapidité utilisés sont le temps de montée et le temps d'établissement du régime permanent. Le système régulé est d'autant plus rapide que le temps de réponse à 5 % est court.

- La stabilité : un système de régulation est stable si à une variation bornée du signal d'entrée correspond une variation bornée du signal de sortie. Pour une consigne constante la sortie doit tendre vers une constante en régime permanent. La stabilité en boucle fermée d'un système de régulation automatique est une condition impérative et elle dépend étroitement des pôles de sa fonction de transfert. Les pôles d'une fonction de transfert sont les valeurs pour lesquelles le dénominateur s'annule. En effet, le système est stable si et seulement si les pôles de sa fonction de transfert sont à partie réelle strictement négative.

- La précision : la précision caractérise la capacité du système à se rapprocher le plus possible de la valeur de consigne. Dans ce contexte, un système est précis, si en régime permanent, la sortie suit la consigne en toutes circonstances avec un petit écart. i.e., plus l'erreur statique est faible meilleure est la précision.

- La robustesse : un système est dit robuste s'il continue à fonctionner correctement malgré les modifications de son environnement. Le système doit résister aux variations de paramètres et/ou à l'imprécision du modèle

2.3.1 L'impact des trois gains K_P, K_I et K_D

Dans le cas de la régulation proportionnelle, l'erreur est multipliée par un gain constant noté K_P. Augmenter K_P permet de réagir plus rapidement aux changements de consigne ou aux écarts à la consigne. Dans le cas d'un simple régulateur proportionnel, une erreur statique généralement subsiste. Le terme proportionnel sert à diminuer le temps de réponse et l'erreur statique.
Si un terme intégral est ajouté une régulation *PI* est obtenue. L'erreur entre la consigne et la mesure est intégrée par rapport au temps et multipliée par un gain K_I. Ce terme intégral permet de compenser l'erreur statique. Par conséquent, on obtiendra un système plus stable en régime permanent. Plus le gain K_I est élevé, plus l'erreur statique est corrigée. Par contre, augmenter K_I revient à augmenter le dépassement.
Le terme dérivé est le produit de K_D par la dérivée de l'erreur par rapport au temps. Le terme dérivé permet de limiter le dépassement de la consigne engendré par le contrôle PI. En effet, lorsque le système se rapproche de la consigne, ce terme freine le système en appliquant une action dans le sens opposé et permet donc une stabilisation plus rapide.
Toute la difficulté réside dans le choix de bons coefficients K_P , K_I *et* K_D afin d'assurer un compromis entre l'exigence d'un contrôle rapide et le besoin d'un contrôle stable.

2.3.2 Réglage des paramètres des régulateurs PID

Le régulateur idéal n'existant pas, la synthèse d'une boucle de régulation et notamment le calcul des paramètres du régulateur doit permettre de répondre au plus grand nombre de contraintes exigées par le cahier des charges du procédé à réguler. Les exigences du cahier des charges sont décrites soit dans le domaine temporel, soit dans le domaine fréquentiel. Le critère de réglage est alors fixé à partir soit de la forme de la réponse temporelle souhaitée pour un type d'entrée, soit à partir des marges de stabilité. Dans la littérature, plusieurs méthodes et règles ont été proposées pour le réglage du contrôleur *PID* et pour la détermination des paramètres K_P,K_I *et* K_D. Ces méthodes peuvent être divisées en trois catégories : méthodes analytiques (par modélisation), méthodes empiriques (par expérimentation) qui nécessitent une connaissance de la fonction de transfert du système et méthodes empiriques qui n'exigent pas la connaissance d'un modèle du système à réguler. Les méthodes analytiques se basent sur un modèle du système à réguler et procèdent à un calcul pour déterminer des valeurs plausibles pour les paramètres K_P, K_I *et* K_D. Parmi ces méthodes on peut citer la commande optimale de type *LQ/LQG* [Doy78] qui est apparue dans les années 70, avec les premiers développements des calculateurs numériques. Pour faire face aux incertitudes des modèles (e.g. incertitudes sur les paramètres, dynamiques non modélisées, perturbations), des méthodes dites robustes sont apparues tel que la synthèse H_∞ [ZF81] et la *µanalyse* [Lar+01]. Ces méthodes permettent de considérer certaines spécifications de performance. Ces méthodes exigent un modèle décrivant la dynamique du système ce qui n'est pas toujours le cas. D'autre part, les méthodes empiriques reposent sur l'expérimentation et partent de la réponse du système pour régler d'abord grossièrement puis finement les paramètres K_P, K_I *et* K_D. Parmi les méthodes empiriques qui nécessitent une connaissance de la fonction de transfert du système à commander, on peut citer les diagrammes de *Bode* [Bod40] , *Nyquist* [Nyq32], et la méthode du lieu des racines (*Root Locus*) [Eva50; Eva48]. Finalement, la méthode *Ziegler-Nichols* [ZN93a] fait partie des méthodes empiriques qui n'exigent aucune connaissance du système à réguler. On s'intéresse ici aux méthodes empiriques. Parmi celles-ci, nous retenons les méthodes de *Ziegler-Nichols* et du lieu des racines car elles sont particulièrement utilisées [Sen+14; BSP12], outillées [CSV96] et documentées [BK18; RF90].

2.3.2.1 Méthode de *Ziegler-Nichols*

En 1942, Ziegler et Nichols [ZN93b] ont proposé deux approches expérimentales dans le but de faciliter la mise au point des paramètres des régulateurs à structure *Proportionnelle (P)*, *Proportionnelle-Intégrale (PI)* et *Proportionnel-Intégral-Dérivée(PID)*. La première exige l'enregistrement de la réponse indicielle du système à régler seul, alors que la deuxième nécessite d'amener le système en boucle fermée à sa limite de stabilité. Il convient néanmoins de noter que ces méthodes ne s'appliquent en général qu'à des systèmes sans comportement oscillant et qui peuvent être amenés vers l'instabilité en augmentant le gain proportionnel K_P.

2.3.2.1.1 Méthode de Ziegler et Nichols en boucle ouverte

Le point de départ de cette approche est l'enregistrement de la réponse indicielle du système à régler en absence du régulateur à une entrée en échelon. Ensuite, on trace la tangente au point d'inflexion noté Q de la courbe (voir figure 2.4). On mesure ensuite le temps de retard T_u correspondant au point d'intersection entre l'axe des abscisses et la tangente ainsi que T_g le temps *de montée de la tangente* correspondant à l'intersection entre la tangente et la ligne de niveau en régime permanent. On peut donc calculer les coefficients du régu-

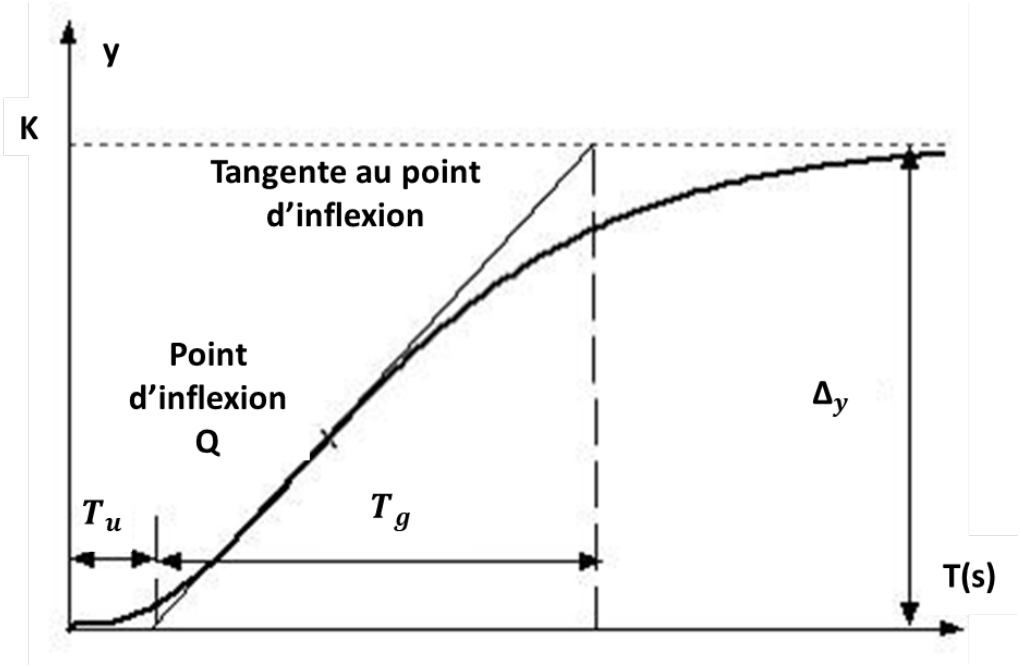

Figure 2.4: Réponse indicielle du système à régler seul

lateur (K_P, K_I, K_D) choisi à l'aide des recommandations du tableau 2.1 ci-dessous [Åst06; Mud06]. Il est à noter que $K_I = {}^{K_P}/_{T_i}$ et que $K_D = K_P\ T_d$.

Type	K_P	T_i	T_d
P	${}^{T_g}/_{K \cdot T_u}$		
PI	$0.9 \cdot {}^{T_g}/_{K \cdot T_u}$	$3.3\ T_u$	
PID	$1.2 \cdot {}^{T_g}/_{K \cdot T_u}$	$2.0\ T_u$	$0.5\ T_u$

Table 2.1: Principes de réglage de gains de régulateurs P, PI et PID selon la première méthode de *Ziegler-Nichols*

2.3.2.1.2 Méthode de Ziegler et Nichols en boucle fermée

La deuxième approche proposée par *Ziegler-Nichols* est appelée méthode du point critique. Il s'agit d'une approche heuristique expérimentale qui se base sur la simulation du système pour ajuster les paramètres du régulateur PID. Cette méthode demande d'amener le système bouclé à sa limite de stabilité. En partant d'une boucle composée du processus avec un simple régulateur proportionnel, on augmente le gain K_P jusqu'à une valeur K_{cr} appelée gain critique qui amène le système à osciller de manière permanente (voir figure 2.5). Cet état est appelé la limite de stabilité. La méthode est facile à mettre en oeuvre car

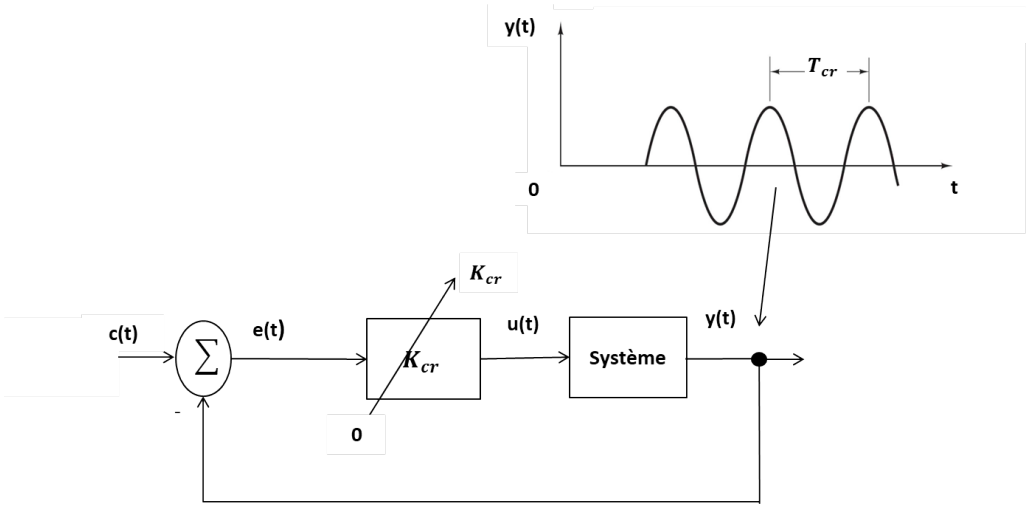

Figure 2.5: Méthode de Ziegler-Nichols en boucle fermée

des abaques ont été proposées par différents auteurs (par exemple le tableau 2.2 [Åst06; Mud06]) qui permettent à partir du gain critique K_{cr} et de la période d'oscillation T_{cr} d'obtenir K_P, K_I et K_D. Généralement, les valeurs obtenues par

Type	K_P	T_i	T_d	$K_I = \frac{K_P}{T_i}$	$K_D = K_P\ T_d$
P	$0.5\ K_{cr}$				
PI	$0.4\ K_{cr}$	$0.8\ T_{cr}$		$0.5\ K_{cr}/\ T_{cr}$	
PID	$0.6\ K_{cr}$	$0.5\ T_{cr}$	$0.125\ T_{cr}$	$1.2\ K_{cr}/\ T_{cr}$	$0.075\ K_{cr}\ T_{cr}$

Table 2.2: Paramètres PID obtenus à partir du point critique

la méthode *Ziegler-Nichols* donnent un dépassement élevé et un temps de montée relativement court. Parfois, il faut intervenir pour corriger le paramétrage proposé notamment en diminuant le gain K_P.

2.3.2.1.3 Bilan

L'avantage majeur de la méthode de *Ziegler-Nichols* en boucle ouverte est que cette dernière ne nécessite pas d'avoir un système déjà asservi. Néanmoins, et

dans le cas ou la réponse du système est trop différente de la réponse donnée dans la figure 2.4, cette méthode peut donner des valeurs inadaptées [Jac11].
La deuxième méthode de *Ziegler-Nichols* présente l'avantage d'être facile à mettre en oeuvre notamment en termes de calcul. De plus, cette approche peut être appliquée sur le système en production, correspond à la réalité, et peut être utilisée à la volée si les caractéristiques du système sont modifiées (usure, changement de l'environnement) [Kur07]. Cependant, le système peut devenir instable voire même passer dans des états critiques. Cette procédure peut prendre beaucoup de temps si le système réagit très lentement ou s'il faut répéter ce processus de nombreuses fois dans le cas des systèmes évoluant en environnement incertain [Jac11].

2.3.2.2 Root Locus

En 1948, *W. R.Evans* [Eva50; Eva48] a proposé une approche graphique appelée lieu des racines (*root locus* en anglais), encore appelée lieu d'Evans.

2.3.2.2.1 Principe de la méthode

Considérons un système G bouclé avec un simple contrôleur proportionnel caractérisé par un gain K comme le montre la figure 2.6.

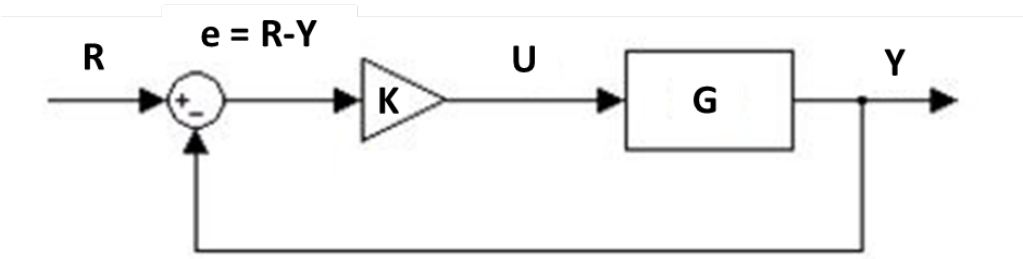

Figure 2.6: Lieu des racines : interprétation avec correcteur

Pour étudier un système de contrôle par la méthode *Root Locus* il est convient de suivre les étapes suivantes [Oga01] :

- on commence par la détermination de la fonction de transfert en boucle fermée du système
$$\frac{Y}{R} = \frac{K \cdot G}{1 + K \cdot G}$$
- les pôles correspondent aux racines (*Roots* en anglais) de la fonction caractéristique (dénominateur)
$$\Delta = 1 + K \cdot G = 0$$
- le changement de la valeur du gain K engendre un changement dans l'emplacement des pôles

- positionner les racines à la main [Bav17] ou via un logiciel adapté tel que *Matlab* et sa fonction *rlocus* [CSV96].
- paramétrer les positions des racines en fonction du facteur de gain K.
- analyser le tracé des positions des racines afin de préciser les zones de stabilité en fonction du gain K (voir figure 2.7)

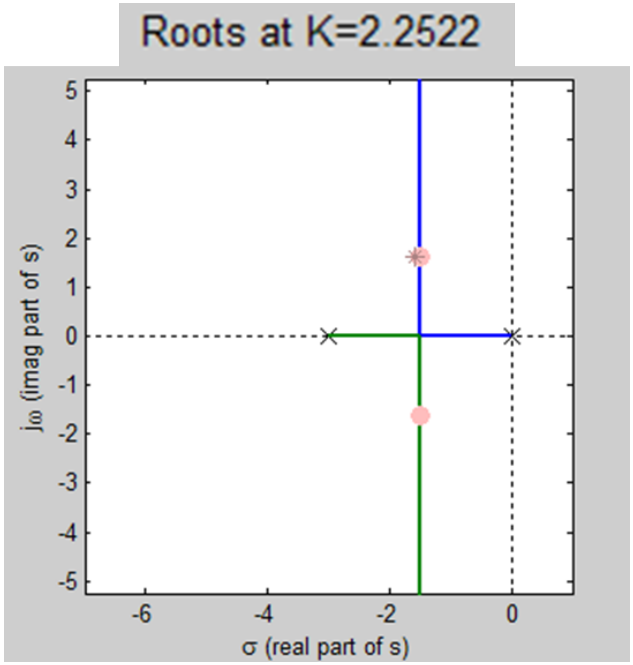

Figure 2.7: Exemple de tracé des lieux des racines

- réaliser la synthèse du correcteur adéquat répondant au exigences fixées.

Cette méthode de synthèse est utilisable quel que soit le système. Il faut tout de même s'assurer du traçage du lieu des racines. Les emplacements des pôles en boucle fermée ont un impact direct sur le comportement dynamique souhaité du système à commander comme spécifié dans le cahier des charges (en termes de temps de montée, de temps de stabilisation et d'oscillations transitoires). Il est à noter que les racines du système doivent être placées dans le demi-plan complexe gauche. Si l'un des pôles sélectionnés se trouve sur le plan complexe de la moitié droite, le système en boucle fermée est instable. Cette méthode graphique consiste à choisir une valeur pour le gain dans le but d'amener les pôles de la fonction de transfert du système en boucle fermée à une zone répondant aux exigences fixées par le cahier des charges.

2.3.2.2.2 Bilan

L'intérêt de la méthode *Root Locus* réside dans la possibilité d'ajuster les paramètres du régulateur graphiquement. En particulier, il est possible de tracer

le lieu des racines moyennant *Matlab Simulink* [CSV96]. Avec la souris, on déplace le curseur de mesure sur le lieu des racines. Ceci permet de visualiser l'impact des valeurs de gain sur l'emplacement des pôles et donc l'impact de ce dernier sur les performances du système à commander en boucle fermée. L'inconvénient majeur de cette méthode est que cette dernière est très sensible aux changement de l'emplacement des pôles. De ce fait, cette méthode exige un modèle fidèle du système à commander ce qui n'est pas toujours évident à obtenir.

2.3.3 Bilan

Dans cette thèse, on s'intéresse au réglage des paramètres du contrôleur. Il pourra être intéressant de se comparer aux techniques de réglages en automatique les plus utilisées, outillées et documentées. Parmi celles-ci, nous retenons les méthodes de *Ziegler-Nichols* et *Root Locus*. Ces méthodes itératives reposant sur l'intervention humaine rendent possible le réglage des paramètres du contrôleur. Il est possible de trouver une solution (un bon paramétrage) qui répond à des exigences fixées (en termes de temps de réponse et/ou de montée, dépassement ...). La méthode *Root Locus* impose une connaissance de la fonction de transfert du système, ce qui est en pratique rarement possible. Même si les deux méthodes proposées par *Ziegler-Nichols* dépassent cette limitation, ces deux techniques proposent dans de très nombreuses situations des paramètres qui conduisent souvent à un temps de montée relativement court assorti d'un dépassement élevé [Jac11]. Enfin, il est à noter pour toutes ces techniques, qu'il faut expérimenter, tâtonner pour trouver un bon paramétrage pour la loi de commande ce qui conduit rapidement à l'explosion du nombre de cas à tester et nécessite donc de s'intéresser à ce qu'on appelle la réduction de dimensionnalité.

2.4 Réduction de dimensionnalité

L'objectif de cette partie est de présenter différentes méthodes de réduction de dimensionnalité.
Dans de nombreux domaines, la résolution d'un problème est basé sur un ensemble de variables appelées également caractéristiques (*features* en anglais). L'augmentation du nombre de variables qui modélise le problème pose des difficultés en termes de complexité, de temps de calcul et de résolution en particulier en présence de données bruitées. L'augmentation du nombre de variables tend à disperser les données et conduit au "fléau de la dimension"[KM17]. Par exemple, pour un problème de classification, en grande dimension, les dissimilarités s'accentuent et on perd la capacité à trouver des individus qui se ressemblent. Pour faire face à ce problème, une solution consiste à faire appel à des méthodes de réduction de dimensionnalité. On désigne par réduction de dimensionnalité [MPH09] toute méthode qui permet la projection d'un ensemble ou espace de données de grande dimension dans un espace de plus petite dimension. Le terme dimension est employé au sens algébrique pour désigner la dimension de

l'espace vectoriel sous-jacent aux valeurs des vecteurs de caractéristiques (*features*). La réduction de dimensionnalité permet de réduire la complexité du problème d'exploration en limitant le nombre de possibilités à considérer. Pour la réduction de la taille de l'ensemble de données, on distingue souvent dans cette catégorie les méthodes par extraction de caractéristiques et les méthodes par sélection de caractéristiques.

- Extraction de caractéristiques[Guy+06]: consiste à combiner les variables et les transformer dans le but d'obtenir un plus petit nombre de nouvelles variables plus expressives et/ou moins redondantes. Parmi les méthodes les plus connues, on peut citer l'*ACP* (Analyse en Composantes Principales). L'extraction implique une transformation des entités, souvent non réversible. Certaines informations sont donc perdues au cours du processus de réduction de la dimensionnalité.

- Sélection de caractéristiques [KM14; SIL07; TAL14]: consiste à extraire un sous ensemble de variables de l'ensemble de départ sans transformation. Cela revient à supprimer des variables. Contrairement à l'extraction de caractéristiques, les techniques de sélection conservent un sous-ensemble de variables telles quelles qui sont donc plus facilement interprétables par un expert du domaine.

Nous avons choisi d'utiliser des méthodes de sélection plutôt que d'extraction car elles permettent comme il vient d'être dit de rester dans un sous-ensemble de variables connues. Nous présentons maintenant les principes généraux de ces méthodes.

2.4.1 Définition de la sélection

La sélection de caractéristiques est un processus de recherche permettant de trouver un sous-ensemble *"pertinent"* de caractéristiques à partir de celles de l'ensemble de départ. La notion de pertinence d'un sous-ensemble de caractéristiques est étroitement liée aux critères et objectifs du système. Considérons un ensemble de caractéristiques F = $\{f_1, f_2, ..., f_n\}$ de taille n où n correspond au nombre total de caractéristiques. Soit E_v une fonction à valeurs réelles qui permet l'évaluation d'un sous-ensemble de caractéristiques. Nous supposons que plus la valeur de E_v est élevée, plus le sous-ensemble de caractéristiques est pertinent. La sélection a pour objectif de trouver un sous-ensemble F' ($F' \subset F$) de taille n' ($n' \leq n$) tel que :

$$E_v(F') = \max_{\{\mathbb{G} \subset F\}} E_v(\mathbb{G})$$

Ou $|\mathbb{G}| = n'$ et n' est un nombre prédéfini par l'utilisateur. La procédure générale pour une méthode de sélection de caractéristiques proposée par [DL97] est illustrée par la figure 2.8.

2.4.2 La pertinence d'une caractéristique

Selon [JKP94], dans le cadre des algorithmes d'apprentissage une caractéristique peut être qualifiée comme étant très pertinente, peu pertinente ou non perti-

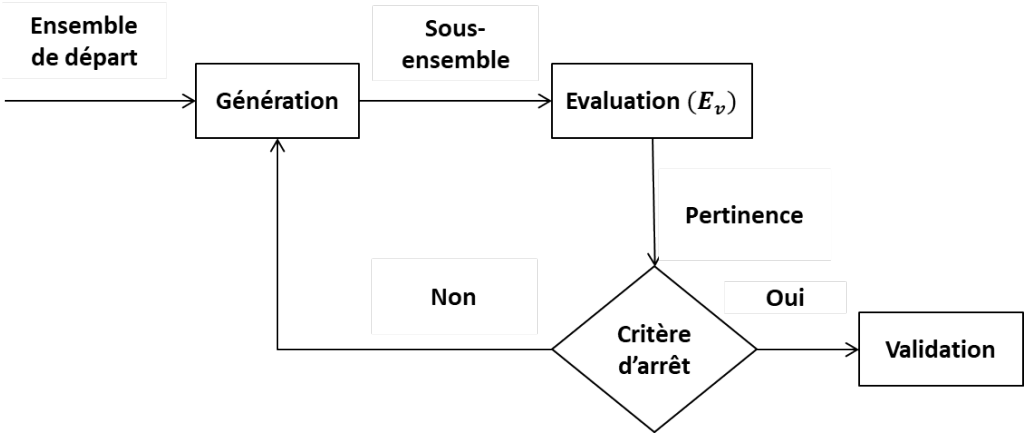

Figure 2.8: Procédure générale d'un algorithme de sélection de caractéristiques

nente.

- Une caractéristique f_i est dite très pertinente si son absence provoque une détérioration majeure de la performance du système de classification utilisé. Quelque soit le sous ensemble V avec V ne contenant pas f_i, la performance de V $E_v(V)$ est très en dessous de $E_v(V \cup f_i)$.
- Une caractéristique f_i est dite peu pertinente si elle n'est pas très pertinente et s'il existe un sous-ensemble V tel que la performance de $V \cup f_i$ soit significativement meilleure que la performance de V.
- Les caractéristiques qui sont à la fois ni peu pertinentes ni très pertinentes sont dites non pertinentes. Ces dernières sont supprimées de l'ensemble de caractéristique de départ.

2.4.3 Principe de sélection de caractéristiques

Chaque méthode de sélection comporte quatre étapes [LY05] (voir figure 2.8). La première étape consiste à définir un ensemble de caractéristiques initial. Ensuite un sous-ensemble de caractéristiques est généré (étape 2) puis évalué (étape 3). Ces deux étapes sont répétées jusqu'à satisfaction d'un critère d'arrêt. Finalement un sous-ensemble de caractéristiques, s'il existe, est sélectionné (étape 4).

2.4.3.1 Initialisation

Pour commencer, il est nécessaire de définir un ensemble de départ. La recherche peut partir de l'ensemble de toutes les caractéristiques et procéder à l'élimination d'une caractéristique à chaque itération si elle est jugée non pertinente ou la recherche peut partir de l'ensemble vide et procéder par l'ajout d'une caractéristique à chaque itération si elle est suffisamment pertinente. Une autre possibilité consiste à commencer la procédure de recherche à partir d'un sous-ensemble quelconque de caractéristiques. Une fois cet ensemble de départ choisi, une procédure de recherche doit être définie.

2.4.3.2 Génération

Cette procédure a pour objectif de générer des sous ensembles de caractéristiques. On distingue trois catégories de stratégie de recherche: aléatoire, exhaustive et heuristique.

2.4.3.2.1 Génération aléatoire

La procédure de recherche aléatoire dite aussi stochastique ou non-déterministe consiste à générer aléatoirement un nombre fini de sous-ensembles de caractéristiques dans le but de sélectionner le meilleur. Ces stratégies de recherche aléatoire convergent dans la plupart des cas rapidement vers une solution *"semi-optimale"*.

2.4.3.2.2 Génération exhaustive

Une recherche exhaustive sur tous les sous-ensembles de caractéristiques est menée dans le but de sélectionner le "meilleur" sous-ensemble de caractéristiques. Elle offre des garanties élevées pour trouver le sous-ensemble optimal. L'inconvénient majeur de cette stratégie est que le nombre de combinaisons croît exponentiellement en fonction du nombre de caractéristiques. Le nombre de sous-ensembles d'un ensemble à n éléments est 2^n.

2.4.3.2.3 Génération heuristique

Pour ces raisons, on s'appuie plutôt sur une approche heuristique pour guider la recherche. Les algorithmes utilisant cette stratégie sont généralement des algorithmes itératifs. A l'issue de chaque itération, soit on rejette soit on sélectionne une ou plusieurs caractéristiques. Contrairement à l'approche exhaustive, ces algorithmes ne rendent pas possible le parcours total de l'ensemble des sous-ensembles possibles de caractéristiques. Les avantages de ces algorithmes résident dans leur simplicité et leur rapidité. Dans la littérature, on distingue trois sous-catégories :

- *Forward* : on commence avec une espace de caractéristiques vide et à chaque itération on y ajoute une ou plusieurs caractéristiques.
- *Backward* : on part de l'ensemble de toutes les caractéristiques et à chaque itération, une ou plusieurs caractéristiques seront éliminées. Il s'agit d'une approche descendante.
- *Stepwise* : cette approche est une combinaison des deux précédentes et consiste à ajouter ou supprimer des caractéristiques au sous-ensemble courant.

Il est à noter que pour un ensemble de données et une initialisation particulière, une stratégie de recherche heuristique renvoie toujours le même sous-ensemble.

2.4.3.3 Évaluation

Dans la littérature, on distingue plusieurs méthodes d'évaluation d'un sous-ensemble de caractéristiques dont les plus classiques sont les :

- Approche *filtre*
- Approche *wrapper*

2.4.3.3.1 Approche filtre

Dans le domaine de l'apprentissage, l'approche *"filtre"* est considérée comme une étape de pré-traitement avant la phase d'apprentissage. Le processus d'évaluation des sous-ensembles de caractéristiques est indépendant de l'algorithme d'apprentissage [JKP94]. Ces méthodes reposent généralement sur une approche heuristique comme stratégie de recherche. Considérons un ensemble de m exemples d'apprentissage x_k, y_k avec $(k = 1,2, \ldots m)$, x_k étant composé de N valeurs des caractéristiques x_{ik} et d'une variable de sortie y_k qui représente l'étiquette de la classe de chaque combinaison de valeurs de caractéristiques $(x_1,x_2,\ldots,x_N)$ (voir tableau 2.3).

	Caractéristiques				*Variable de sortie*
Nom de variable	x_1	x_2	..	x_N	y_k
valeur de la variable	x_{11}	x_1	..	x_{N1}	y_1
	x_{12}	x_{22}	..	x_{N2}	y_2
	..	..	..	..	..
	x_{1m}	x_{2m}	..	x_{Nm}	y_m

Table 2.3: Exemple illustratif de l'approche *filtre*

Les caractéristiques sont évaluées à base de mesures calculées pour chacune de caractéristiques x_i. Ces mesures $E_v(x_i)$ calculées à partir des valeurs x_{ik} et y_k permettent l'évaluation de la pertinence de chacune des caractéristiques. Par convention, nous supposons qu'un score élevé indique une caractéristique pertinente et que nous trions les variables dans l'ordre décroissant de $E_v(x_i)$. La figure 2.9 illustre la procédure suivie pour l'approche *"filter"*.

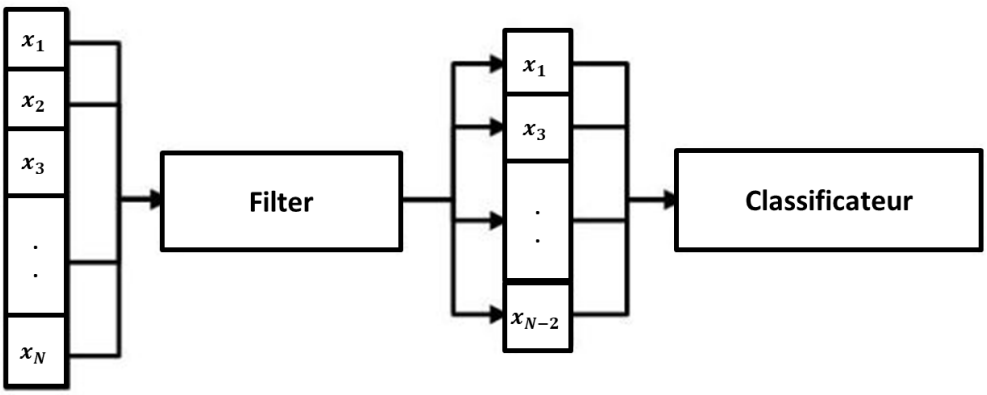

Figure 2.9: Schéma de l'approche filtre

Parmi les principales mesures utilisées dans la littérature, on peut citer [GE03a]:

- Le coefficient de corrélation est défini comme suit ou *cov* désigne la covariance et *var* la variance :

$$E_v(x_i) = \frac{cov\ (X_i, Y)}{\sqrt{var(X_i)var(Y)}}$$

- Le critère de Fisher : Cette mesure cherche un espace vectoriel de faible dimension qui maximise la variance inter-classe et minimise la variance intra-classe [DHS00a]. Ce critère est défini par :

$$E_v(x_i) = \frac{\sum_{c=1}^{C} \eta_c (\mu_c^i - \mu^i)^2}{\sum_{c=1}^{C} \eta_c (\sigma_c^i)^2}$$

  où μ_c, μ_c^i et σ_c^i représentent respectivement l'effectif, la moyenne et l'écart type de la i^{me} caractéristique au sein de la classe c. μ^i est la moyenne globale de la i^{me} caractéristique.

- L'information mutuelle: cette mesure caractérise la dépendance entre les distributions de deux populations [FS86].

2.4.3.3.2 Approche wrapper

Les approches *wrapper* ont été introduites en 1997 par Kohavi et John [KJ97] pour l'évaluation des caractéristiques. Leur principe consiste à générer des sous-ensembles de caractéristiques et de les évaluer moyennant un algorithme d'apprentissage. Cette évaluation est faite par le biais d'un classificateur qui mesure la pertinence d'un sous-ensemble donné de caractéristiques. La figure 2.10 met en évidence la procédure du modèle *"wrapper"*. Les sous-ensembles de caractéristiques sélectionnés par cette approche dépendent étroitement de l'algorithme de classification e mployé. D e c e f ait, l e c hangement du classificateur peut éventuellement engendrer un changement du sous-ensemble de caractéristiques sélectionné. Les multiples appels à l'algorithme de classification rendent cette approche très coûteuse en terme de temps de calcul. Le mécanisme de validation croisée [STO77; Gei75] (*cross-validation* en anglais est une technique permettant l'utilisation de l'intégralité du jeu de données pour l'entraînement et pour la validation de l'algorithme d'apprentissage) est généralement utilisé dans le but de diminuer le temps de calcul et afin d'éviter les problèmes de sur-apprentissage.

2.4.3.3.3 Bilan

L'approche *filtre* p résente l 'avantage d 'être m oins c oûteuse e n t erme de temps de calcul car celle-ci n'exige pas d'exécutions répétitives d'algorithmes d'apprentissage sur différents sous-ensembles de caractéristiques. En revanche, cette approche ne tient pas compte de l'impact des caractéristiques sur

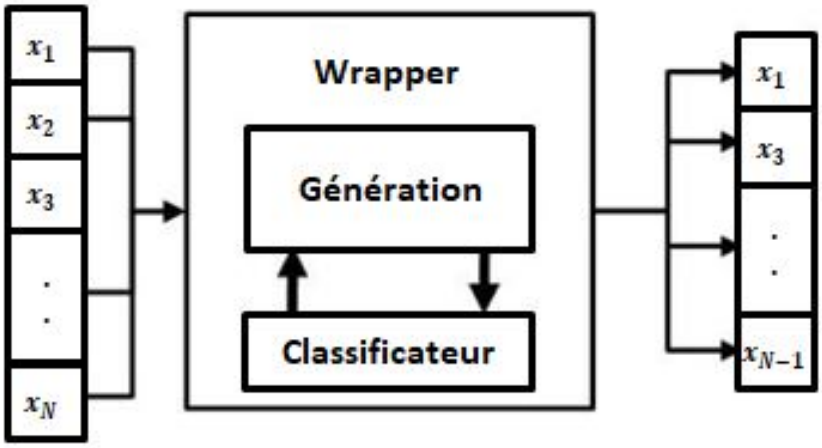

Figure 2.10: Schéma de l'approche wrapper

l'algorithme d'apprentissage qui suit la procédure de sélection [KJ97]. Selon [GE03b], cette méthode a tendance à sélectionner un sous-ensemble de caractéristiques caractérisées par la redondance plutôt que la complémentarité.
Les approches *"wrapper"* sont généralement considérées comme étant beaucoup plus performantes que les approches *"filtre"* [HYC08]. En prenant en compte le biais intrinsèque de l'algorithme de classification, ces méthodes peuvent sélectionner des sous-ensembles de caractéristiques qui sont pertinents. Cependant, les principaux inconvénients liés à ces méthodes sont :

- Le temps de calcul : l'inconvénient majeur de l'approche *"wrapper"* réside dans le temps nécessaire pour la sélection de caractéristiques qui est beaucoup plus long que celui des approches *"filtre"*. Ceci est dû essentiellement à l'emploi d'un algorithme de classification pour l'évaluation des sous-ensembles, sans oublier l'utilisation de validation croisée.

- La dépendance des caractéristiques sélectionnées à l'algorithme de classification employé : le sous-ensemble sélectionné dépend étroitement du classificateur choisi pour la phase d'évaluation. Chaque classificateur s'appuie sur ses propres spécificités et hypothèses.

2.4.3.4 Critère d'arrêt

Une fois que le processus de recherche du sous-ensemble de caractéristiques est lancé, il faut définir un critère d'arrêt. Le critère doit être capable d'arrêter le processus à partir de plusieurs sous-ensembles de caractéristiques. Le choix du critère dépend de la méthode d'évaluation utilisée. Le critère généralement employé avec les méthodes *"filtre"* est basé sur l'ordre des caractéristiques, classées selon des scores de pertinence. Ces scores correspondent à des

mesures statistiques. Les caractéristiques ayant les scores les plus élevés sont sélectionnées et utilisées par un classificateur. En ce qui concerne le critère d'arrêt employé avec les méthodes d'évaluation de type *"wrapper"*, le processus de recherche prend fin lorsqu'il n'y a pas d'autre possibilité pour trouver un sous-ensemble pertinent que le sous-ensemble actuel. Ainsi on poursuit le processus de recherche jusqu'au moment ou la précision dépasse un seuil fixé auparavant par l'utilisateur. De nombreuses méthodes utilisent soit des critères de maximum de vraisemblance [DB00; FM17] , soit des critères de séparabilité des classes [GB06]. A noter que les critères de séparabilité ne sont plus utilisables lorsque le nombre de caractéristiques est important car leur évaluation fait intervenir soit l'inversion, soit le calcul du déterminant de matrices de covariance. D'autres auteurs utilisent une combinaison de la mesure de pertinence et de la procédure de recherche.

2.4.4 Revue des méthodes de sélection adoptées

Dans cette partie, nous présentons les méthodes de sélection de caractéristiques retenues.

2.4.4.1 Step Forward Feature Selection

Cette méthode connue aussi sous le nom de sélection séquentielle croissante fut la première méthode proposée pour la sélection de caractéristiques et a été proposée en 1963 par Marill et Green [MG63]. Cette méthode repose sur une approche heuristique pour la génération des sous-ensembles à évaluer. Elle part d'un ensemble vide de caractéristiques. A chaque itération, la meilleure caractéristique est retenue et est supprimée de l'ensemble de départ. Pour l'évaluation, une approche *wrapper* est utilisée pour évaluer la pertinence des sous-ensembles générés. Ce processus de sélection se répète jusqu'à la satisfaction d'un critère d'arrêt.

2.4.4.2 Step Backward Feature Selection

Cette méthode appelée aussi sélection séquentielle arrière a été proposée par Whitney [Whi71] en 1971. Cette méthode part d'un ensemble contenant toutes les caractéristiques contrairement à la méthode de sélection séquentielle croissante. A chaque itération, une approche *wrapper* est utilisée pour évaluer l'ensemble de départ privé de chacune des caractéristiques. La caractéristique la plus mauvaise est supprimée. On refait ce processus de suppression de caractéristiques jusqu'à obtenir un sous-ensemble contenant le nombre de caractéristiques souhaitées. Malgré les similitudes entre les deux méthodes, une étude comparative menée par [AB95] a montré que la méthode de sélection arrière offre une meilleure performance par rapport à la sélection croissante. Ceci est dû au fait que ces méthodes tiennent compte de l'interaction d'une caractéristique avec un ensemble de caractéristiques plus large, tandis que les méthodes de

sélection croissante ne prennent en considération que l'interaction de cette caractéristique avec le sous-ensemble déjà sélectionné. Une extension des ces approches a été proposée par Kittler [Kit78]. Au lieu de se contenter de l'exclusion ou de l'inclusion d'une caractéristique à chaque itération, les auteurs proposent d'inclure ou exclure un sous-ensemble de caractéristiques.

2.4.5 Bilan

Pour faire face aux nombreux enjeux engendrés par le fléau de la dimension, l'utilisation d'algorithmes de sélection de caractéristiques permet de ne conserver que les variables les plus pertinentes. Étant donné que la génération aléatoire converge généralement vers une solution *"semi-optimale"* et que la génération exhaustive est compliquée (trop coûteuse), nous optons pour une génération heuristique de type *Forward* ou *Backward* dans le but de guider la recherche. La génération heuristique présente l'avantage d'être simple et rapide (pas besoin d'effectuer un parcours total de tous les sous-ensembles de caractéristiques possibles). Pour évaluer les caractéristiques, nous optons pour une approche *wrapper* puisque cette approche est plus performante que l'approche de type *filtre*. Ceci nous permet d'éviter de choisir des caractéristiques redondantes. Le processus de recherche prend fin lorsqu'il n'y a pas d'autre possibilité de trouver un sous-ensemble pertinent que le sous-ensemble courant. Finalement, nous retenons les deux méthodes *Step Forward Feature Selection* et *Step Backward Feature Selection* car elles sont particulièrement utilisées, outillées et documentées [KD19; HG15].

2.5 Clustering

2.5.1 Introduction

Dans cette thèse, on s'intéresse au réglage des contrôleurs associés aux systèmes cyber-physiques évoluant en environnement incertain. Le principal défi lors du réglage de ces contrôleurs provient du fait qu'un réglage n'est adapté que pour un contexte spécifique. Un bon réglage (paramétrage) de contrôle (en termes d'exigences relatives à la qualité du contrôle) pour un contexte spécifique peut ne pas être adapté à un autre contexte. Mais si on considère un bon réglage par contexte, l'ensemble des réglages devient vite exponentiel du fait du nombre de contextes à considérer.
Pour faciliter le réglage nous proposons de regrouper les contextes. L'idée est d'associer à chaque groupe un seul réglage qui représentera l'ensemble des réglages du groupe. Le défi consiste donc ici à déterminer un nombre limité mais pertinent de ce qu'on appellera modes de fonctionnement du contrôleur. Un mode de fonctionnement correspond à un réglage donné, considéré comme pertinent pour un ensemble de contextes.
Dans cette partie, on s'intéresse aux approches utilisées dans la littérature pour réaliser un regroupement de données, à savoir, les techniques de *clustering*. Cette partie en présente ses concepts clefs. Le clustering est généralement utilisé pour deux raisons principales :

- Le clustering est considéré comme une étape de prétraitement. Dans ce cas, les résultats du clustering sont exploités par la suite dans une autre tâche. Il n'est donc pas nécessaire d'interpréter les résultats obtenus. On cherche dans ce cas à associer chacune des observations à l'un des clusters sans avoir besoin d'interpréter les résultats issus de l'algorithme de clustering utilisé.
- Les clusters obtenus suite à l'application d'une algorithme de clustering constituent un résultat final. Dans ce cas, le clustering constitue à lui seul un processus global de découverte de groupes. L'exploitation des clusters pour une application donnée passe alors par une description de ces derniers.

Dans cette thèse, le clustering est appliqué pour la deuxième raison.

2.5.2 Définition

Le *clustering* a comme objectif principal de grouper des observations (éléments) en fonction d'une notion de *similarité* et de représenter chaque groupe par un seul représentant. Les observations qui sont considérées comme similaires sont associées au même groupe appelé aussi *cluster* tandis que celles qui sont considérées comme différentes sont associées à des groupes différents. Plus formellement, étant donné un ensemble d'éléments qu'on le note $E = \{e_1, e_2, ...e_n\}$, on cherche à partitionner les n éléments en k ensembles (clusters) $S = \{S_1, S_2, ..., S_k\}$ avec ($1 \leq k \leq n$). Chaque élément d'un cluster S_k doit être à la fois similaire aux autres éléments du même cluster et différent des éléments appartenant aux autres clusters. Le *clustering* fait partie de la classification non-supervisée dans la mesure où l'on ne dispose pas de classes prédéfinies auparavant.

2.5.2.1 Notion de similarité

Il n'existe pas une définition unique (commune) de la similarité entre observations (éléments). La notion de *similarité* dépend étroitement du type de données considérées. Généralement, la *similarité* est basée sur la notion de distance. Dans la littérature, différentes mesures de distance existent [DHS00b]. Parmi celles ci, on peut citer : la distance simple d(x,y) = |x-y| ou |x-y| / d_{max}, distance *euclidienne* (racine carrée de la somme des distances au carré), distance de *Minkowski*, distance de *Mahalanobis*, distance de *Manhattan*. La qualité du clustering dépend de la mesure de *similarité* adoptée.

2.5.3 Étapes du clustering

Le processus de *clustering* comporte trois étapes principales comme le montre la figure 2.11 ci-dessous :

- tendance au clustering ;
- choix de l'algorithme de clustering ;
- évaluation et exploitation des clusters ;

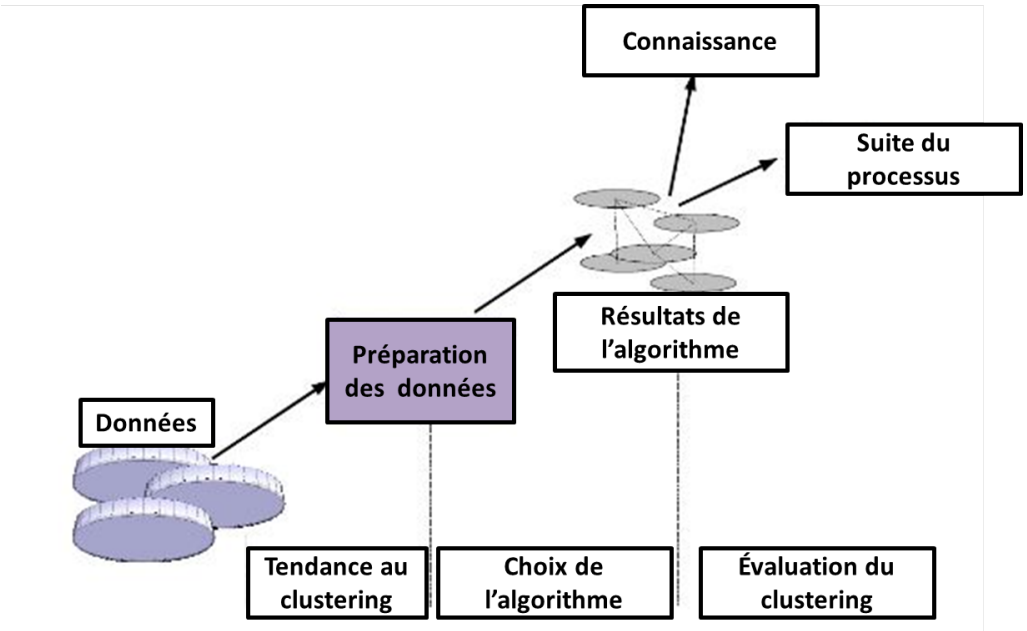

Figure 2.11: Les différentes étapes du processus de clustering

2.5.3.1 Tendance au clustering

Avant d'appliquer une technique de *clustering* aux données, il est recommandé de savoir si les données peuvent être partitionnées en clusters ou non. C'est ce que l'on appelle la tendance au clustering. Ainsi, tester la tendance au clustering permet d'éviter l'application inutile d'un algorithme de clustering. En procédant ainsi on évite de produire des résultats non pertinents.
Des données ayant une tendance au clustering sont généralement distribuées de manière non uniforme sur leur espace de définition. Ainsi, la tâche principale du test de tendance au clustering se ramène à déterminer si les données sont uniformément distribuées, en amont du clustering. Parmi les techniques utilisées, on peut citer les techniques basées sur des tests statistiques [JD88], les approches basées sur des interprétations visuelles [BH02], l'approche de *Holgate* [Hol65] et l'approche *Eberhardt* [Ebe67]. Selon [BD04], une technique qui consiste à déterminer l'indice d'Hopkins [HS54] est la plus simple, intuitive et explicite à utiliser par rapport aux autres approches.

Pour déterminer cet indice, on procède comme suit :

- on commence par calculer la dissimilarité entre les observations et leur plus proche voisin
- on génère aléatoirement autant de points qu'il y en a dans les données suivant une distribution uniforme
- on calcule la dissimilarité moyenne entre les observations et leur plus proche voisin aléatoirement engendré.

2.5.3.2 Choix de l'algorithme de clustering

Après avoir vérifié l'existence d'une tendance au clustering dans les données, le principal problème est de choisir l'algorithme de clustering approprié. Dans la littérature, de nombreux algorithmes de clustering existent, ce qui rend difficile le choix de l'algorithme approprié. Le choix de l'algorithme de clustering dépend étroitement de la nature des variables contenues dans le jeu de données (e.g., quantitative et qualitative) et des clusters attendus (e.g., nombre, forme, densité, etc). Cependant on peut classer les méthodes existantes en catégories selon le principe sur lequel elles reposent. Dans la littérature, on distingue deux grandes familles d'approche de clustering qui peuvent être appliquées dans notre étude et qui sont, respectivement, les méthodes de *partitionnement* et celles basées sur la *densité* [HKP11; JMF99; ZM97].

2.5.3.3 Les méthodes de partitionnement

Les méthodes de partitionnement représentent le moyen le plus simple pour regrouper des points (éléments, observations, situations) dans des clusters. On part d'un jeu de données contenant n points (éléments) et on essaye de construire K clusters ($K \leq n$) en s'assurant que chaque point appartient exactement à un seul cluster. La mesure (métrique) utilisée pour différencier les clusters dans la plupart des méthodes de partitionnement est la distance entre les points. Par conséquent, les points appartenant au même cluster sont probablement proches [HKP11]. L'algorithme de partitionnement le plus connu de la littérature est *K-means* (K-moyennes). Même s'il a été proposé il y a longtemps, *K-means* (*K-moyennes*) [Mac67] reste l'un des algorithmes de partitionnement les plus utilisés par les chercheurs grâce à sa simplicité et à sa facilité de mise en œuvre.
Considérant un jeu de données $E = \{e_1,....,e_n\}$ contenant n points (éléments) et K désignant le nombre de clusters souhaité

- *K-means* sélectionne arbitrairement k points (éléments) parmi les points du dataset qu'on appelle *centroïdes* (centres initiaux) ($c_1,.....,c_k$) de cluster
- Par la suite, chaque point restant de E est assigné à l'un des k clusters S_i dont le centre c_i est le plus proche
- On répète ce processus plusieurs fois et à chaque fois, les valeurs des *centroïdes* sont mis à jour en fonction des points qui appartiennent au cluster correspondant
- Une fois ceci fait, tous les points sont réassignés tout en tenant compte des nouveaux centroïdes de chaque cluster
- Ce processus prend fin lorsque les valeurs des centroïdes et les affectations des points de E sont immuables [Jai10]

2.5.3.4 Les méthodes basées sur la densité

Les méthodes basées sur la densité permettent le partitionnement des groupes en régions denses séparées par des régions qui le sont moins (c.-à.-d., données

isolées). Un cluster correspond donc à une région de points denses, qui est séparée des autres clusters par des régions non denses [HKP11]. Ces méthodes sont utilisées dans le cas ou les clusters sont irréguliers ou mélangés, ou en présence de bruit. L'un des algorithmes basés sur la densité le plus couramment utilisé et aussi le plus souvent cité dans la littérature est le *DBSCAN* (en anglais Density Based Spatial Clustering of Applications with Noise). Cet algorithme a été proposé pour la première fois en 1996 [Est+96]. *DBSCAN* itère sur les points du jeu de données. Pour chacun des points qu'il analyse, il construit l'ensemble des points atteignables par densité depuis ce point. Contrairement aux algorithmes de partitionnement, *DBSCAN* ne nécessite aucune spécification concernant le nombre de clusters. Par contre cet algorithme nécessite deux paramètres: *MinPts* qui correspond au nombre minimum de points requis pour former un cluster et ϵ qui correspond à ce qu'on appelle l'epsilon-voisinage. L'epsilon-voisinage d'un point x correspond à l'ensemble des points du jeu de données dont la distance à x est inférieure à ϵ [Lu+16].

- En partant d'un jeu de données E, ϵ et *MinPts*, tous les points inclus dans E sont marqués comme non visités.
- L'algorithme commence par sélectionner aléatoirement un point non visité e et le marque comme visité.
- Ensuite, l'algorithme détermine le nombre de points dans le epsilon-voisinage de e (epsilon-voisinage = {Q∈E / distance(e, Q) $\leq \epsilon$}).
  - Dans le cas ou le nombre de points est supérieur à *MinPts*, un nouveau cluster est créé contenant tous les points identifiés.
  - Dans le cas contraire, le point e est considéré comme du bruit et donc n'appartient à aucun cluster.
- On répète ce processus jusqu'à ce que tous les points soient attribués à un groupe ou considérés comme du bruit [Est+96; AG17].

2.5.4 Évaluation du clustering

Comme mentionné précédemment, la qualité d'un regroupement dépend de la mesure de similarité utilisée. Un partitionnement idéal est constitué donc de groupes bien séparés et compacts. Pour obtenir un bon partitionnement, il est donc nécessaire de garantir :

- une grande similarité intra-groupes pour obtenir des clusters les plus homogènes (similaires) possible
- une faible similarité inter-groupes pour obtenir des clusters distincts

Les indices à base de distance sont les plus utilisés pour évaluer la qualité du clustering. En effet, plus les données (éléments, observations) à l'intérieur des clusters sont homogènes, plus leurs distances par rapport au point représentant la classe sont faibles. De ce fait, une valeur faible de la distance intra-groupes

décrit une homogénéité des données à l'intérieur des clusters. Plus les clusters sont hétérogènes entre eux, plus les distances entres les points représentants les profils des clusters sont élevées. Ainsi, une valeur élevée de la distance inter-groupes révèle une hétérogénéité entre les clusters.

2.5.5 Bilan

Dans cette thèse, nous cherchons à minimiser le nombre de modes de fonctionnement du contrôleur (loi de commande). Nous cherchons à mettre en place un algorithme de clustering. Nous proposons d'effectuer un regroupement des solutions (bons paramétrages de contrôle) semblables en cluster. Chaque cluster sera représenté par l'un de ses membres. Les algorithmes de clustering classiques ne sont pas adaptés à notre problème où des clusters qui se chevauchent (puisque un "bon" paramétrage de contrôle est valable pour différents contextes) [GHM10]. Lorsque le chevauchement est autorisé, le problème de clustering devient considérablement plus complexe en raison de la possibilité de dégénérescences. Nous nous inspirerons des principes de ces algorithmes (i.e., de *partitionnement* et celles de *densité*) pour mettre en place un algorithme de clustering adapté aux spécificités du problème traité.

2.6 Régression

2.6.1 Introduction

Dans cette thèse, nous envisageons d'exploiter les données de simulation pour prédire un bon réglage de contrôle pour des situations (contextes) inconnues. Ce problème peut être vu comme étant un problème de prédiction et plus précisément un problème de régression. La difficulté de ce problème provient du fait que la relation entre les variables (paramètres) est rarement linéaire ce qui rend difficile la tâche pour trouver un modèle qui caractérise l'interaction entre les différents paramètres. Pour faire face à cette problématique, l'utilisation des réseaux de neurones et particulièrement des perceptrons multi-couches (en anglais MultiLayer Perceptron) [ZPH98; Mur03] présente une alternative intéressante aux approches de régression traditionnelles comme les approches polynomiales. Dans ce chapitre, nous décrivons les réseaux de neurones classiques à savoir le perceptron multi-couches.

2.6.2 Définition de la régression

La régression a pour objectif de trouver une relation entre un ensemble de variables indépendantes (prédicteurs) et une ou plusieurs variables cibles à valeur réelles (*variables dépendantes*) en se basant sur des données historiques (des observations passées) connues a priori [HK14]. La régression est un processus à deux étapes : une étape d'apprentissage et une étape de prédiction. Dans l'étape d'apprentissage, on construit un modèle en analysant les données historiques dans lequel les variables cibles sont supposées connues. Dans l'étape de prédiction, le modèle construit est utilisé pour prédire de nouvelles instances.

De manière plus formelle, soit D un jeu de données d'apprentissage composé de n variables prédictives x et de d variables cibles y [Rub20] :

$$D = \{(x_1, x_2, ..., x_n), (y_1, ..., y_d)\}$$

A chaque instance de variables $\{x_1^{(i)},...,x_n^{(i)}\}$, on associe un vecteur de valeurs cibles $\{y_1^{(i)},...,y_d^{(i)}\}$ avec i $\in$ {1,..,$Nbrs_{Observations}$}. L'étape d'apprentissage a pour but d'apprendre (de trouver), à partir des données d'apprentissage, une fonction f qui attribue à chaque instance, donnée par le vecteur x, un vecteur $\widetilde{y}$ de d valeurs cibles :

$$f : X_1 \times ... \times X_m \rightarrow Y_1 \times ... \times Y_d$$

$$x = (x_1, ..., x_m) \longmapsto \widetilde{y} = (\widetilde{y}_1, ..., \widetilde{y}_d)$$

L'objectif est alors de minimiser l'erreur entre les valeurs prédites $\widetilde{y} = (\widetilde{y}_1, ..., \widetilde{y}_d)$ et les valeurs réelles y = $(y_1, ..., y_d)$. Le taux d'erreur peut être mesuré en termes d'erreur quadratique moyenne *RMSE*, de carré moyen des erreurs *MSE* ou encore d'erreur absolue moyenne *MAE*. Dans l'étape de prédiction, la fonction f construite dans la première étape est utilisée pour prédire les valeurs qui correspondent à des entrées non utilisées dans la 1ère étape.
Parmi les méthodes permettant d'atteindre cet objectif, on trouve par exemple la régression linéaire simple [Oli17], la régression polynomiale [HN09], la régression logistique [HL89], les machines à vecteurs de support (SVM) [SC08] et forêts aléatoires (RF) [Bre01]. Nous optons pour l'utilisation des réseaux artificiels [Dre+02] q ui c onstituent l 'un d es p aradigmes l es p lus u tilisés [Jah18; Kas+18] en raison de leur flexibilité en termes de modélisation et d'identification de modèle à partir de données complexes.

2.6.3 Réseaux de neurones artificiels

2.6.3.1 Historique des réseaux de neurones

L'apparition des réseaux de neurones artificiels r emonte à 1 943. L es mathématiciens *Warren McCulloch* et *Walter Pitts* [LWG11] ont construit un système de circuits qui imite le fonctionnement du cerveau en permettant l'exécution d'algorithmes simples. En 1957, le chercheur *Frank Rosenblatt* de l'université Cornell développe le modèle du perceptron [Ros58]. Il s'agit d'un algorithme de reconnaissance avancée de modèles. Ce modèle contient deux couches de neurones: une couche de précision et une couche dédiée à la prise de décision. Par la suite, le professeur *Bernard Widrow* de l'université Stanford a développé le modèle *ADALINE* (pour Adaptive Linear Neuron ou plus tard Adaptive Linear Element) qui constitue le modèle de base des réseaux multi-couches [WL98]. Chaque neurone est composé d'un poids, un biais (une constante qu'on ajoute à l'entrée) et d'une fonction de sommation. En 1969, les chercheurs *Marvin Minsky* et *Seymour Papert* du MIT ont publié le livre Perceptrons [MP17], qui aborde différentes problématiques des réseaux de neurones, notamment le manque de puissance des ordinateurs de l'époque empêchant de traiter les données nécessaires au fonctionnement normal de ces derniers. A partir de 1982, les réseaux

de neurones ont connu un regain d'intérêt [MMR96]. De nos jours, les réseaux de neurones sont utilisés dans de nombreux domaines [ZPH98; ML10].

2.6.3.2 Définition

Les réseaux de neurones artificiels (en anglais Artificial Neural Networks) notés (*RNA*) [Dre+02] disposent de capacités d'apprentissage à partir de données complexes pour en déduire leur modèle et leur tendance. En général, les *RNA* sont utilisés pour identifier les modèles de données qui ne sont pas facilement perceptibles par les méthodes de régression classiques. Les *RNA* peuvent gérer des relations complexes entre les variables indépendantes et les variables dépendantes y compris des relations non linéaires entre plusieurs variables prédictives et plusieurs variables cibles. Les *RNA* s'inspirent du fonctionnement du cerveau humain et sont dotés de capacité de prédiction même en présence de données fortement bruitées. Généralement, un réseau de neurones artificiel est caractérisé en termes de :

- le modèle de neurones : les caractéristiques de l'unité de traitement (neurone).
- la structure d'interconnexion : la topologie de l'architecture et les poids des connexions qui encodent la connaissance.
- l'algorithme d'apprentissage : les étapes à suivre pour ajuster les poids des connexions

Dans la littérature, on distingue plusieurs modèles de *RNA*. Dans ce qui suit nous présentons les concepts clés relatifs à ces réseaux.

2.6.3.3 Fonctionnement des réseaux de neurones artificiels

Les réseaux de neurones fonctionnent en répartissant les valeurs des variables d'entrées (les prédicteurs) dans ce qu'on appelle des neurones. Ces neurones constituent les briques élémentaires d'un réseau de neurone. Un neurone est une fonction algébrique non linéaire et bornée [Dre+02]. Les neurones sont chargées de combiner entre elles leurs informations pour déterminer la valeur des variables cibles (dépendantes). Pour y parvenir, chaque neurone reçoit des informations numériques en provenance de neurones voisins. On associe à chacune de ces valeurs un poids représentatif de la force de la connexion. A partir de ces informations, chaque neurone effectue un calcul dont le résultat est transmis ensuite aux autres neurones. La figure 2.12 illustre un modèle de neurone artificiel.

Plus formellement, un neurone est une fonction f_j considérant l'entrée x = $[x_1, ..., x_n]$ pondérée par un vecteur de poids de connexion noté $w_j = [w_{1,j}, ..., w_{n,j}]$ complété par une constante qu'on ajoute à l'entrée appelée biais neuronal b_j et associée à une fonction d'activation ϕ. La sortie du neurone est donnée par l'équation suivante :

$$y_j = f_j(x) = \varphi(\sum_{i=1}^{n}(w_{i,j} \times x_i + b_j))$$

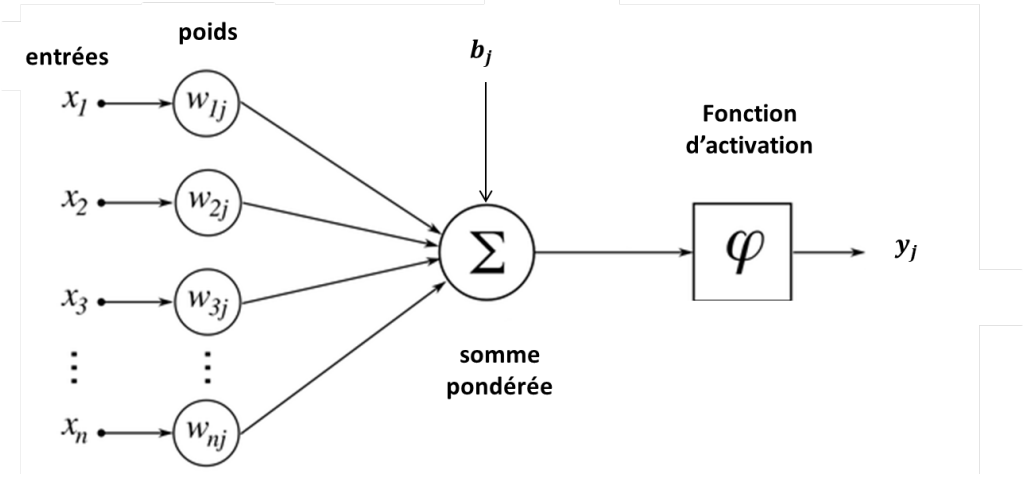

Figure 2.12: Modèle de neurone

2.6.3.4 Fonction d'activation

La fonction d'activation [HDR19] est aussi appelée fonction de transfert. Le terme de "fonction d'activation" vient de l'équivalent biologique "potentiel d'activation", seuil de stimulation qui, une fois atteint entraîne une réponse du neurone. Généralement, la fonction d'activation est une fonction non-linéaire. De nombreuses fonctions d'activation peuvent être utilisées comme fonctions d'activation du neurone. Parmi celles-ci, on peut citer : la fonction *identité*, la fonction *sigmoïde*, la fonction unité linéaire rectifiée (*ReLU*).
La mise en place d'un réseau de neurones artificiels c onsiste à t rouver des valeurs convenables aux poids de connexions entre les différents neurones pour qu'il puisse associer une réponse adéquate à chaque configuration d'entrée. L'utilisation d'un *RNA* se fait donc en deux phases. Dans un premier lieu, une phase d'apprentissage dont l'objectif est de déterminer des valeurs pour chacune des connexions du réseau. Dans un deuxième lieu, une phase d'exploitation (utilisation) proprement dite, où une entrée est présentée au réseau pour calculer (estimer) une sortie.

2.6.3.5 Architecture du réseau de neurones ou types de réseaux de neurones

De nombreux types de réseaux de neurones ont été développés depuis l'apparition des neurones formels. Ces réseaux peuvent être classés en deux catégories (classes) : les réseaux feed-forward (l'information circulent des entrées vers les sorties sans retour en arrière) et les réseaux récurrents (voir figure 2.14) [Bre+12].
Dans cette thèse, nous optons pour l'utilisation du perceptron multi-couches (Multilayer Perceptron en anglais) que l'on note *PMC* (ou *MLP* en anglais) [MP17] qui est particulièrement utilisé dans les applications scientifiques et

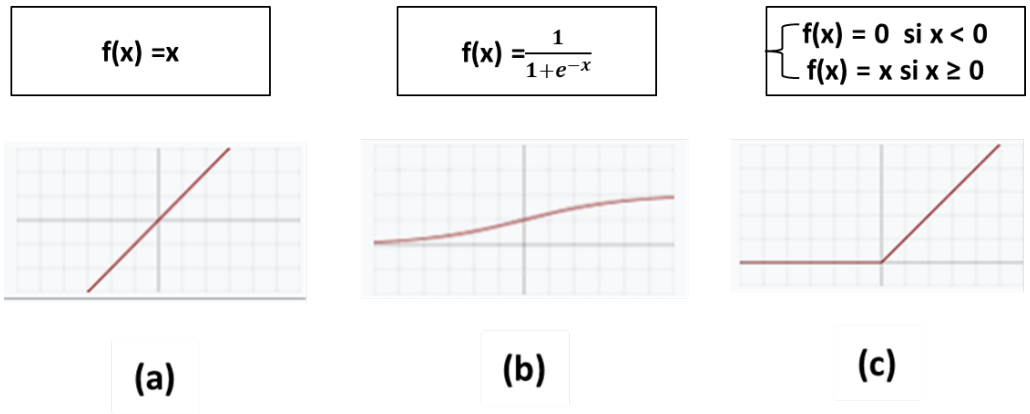

Figure 2.13: Fonctions d'activation : (a) identité ; (b) sigmoïde; (c) ReLU

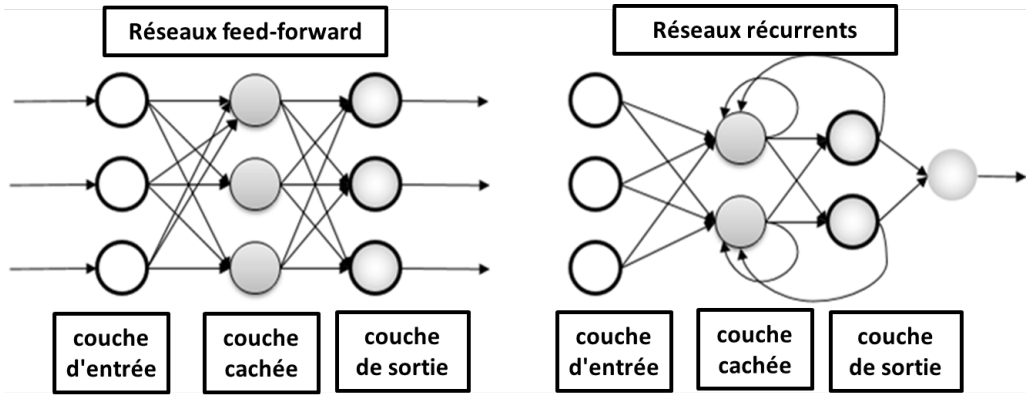

Figure 2.14: Réseaux feed-fordward et réseaux récurrents

industrielles, et documenté [MMR96; SSH07; Kow97; Fri19] . Le *PMC* est un modèle de réseau à propagation par couche (voir figure 2 .15). Il s'agit d'un réseau à propagation directe (feed-forward). Le paradigme *PMC* est construit sur la base d'un système de couches avec pas moins de 3 couches de neurones. La première couche du *PMC*, à savoir la couche d'entrée, est l'endroit où les observations d'entraînement sont alimentées. Le nombre de variables prédictives (prédicteurs) est également spécifié ici via les neurones. La dernière couche, à savoir la couche de sortie, est chargée de fournir la solution du problème (i.e., les valeurs cibles). Les couches intermédiaires entre les couches d'entrée et de sortie sont appelées couches cachées ("hidden layers" en anglais). Il est nécessaire de préciser qu'aucune connexion n'existe entre les neurones d'une même couche. D'autre part, tout neurone d'une couche est connecté à tous les neurones de la couche suivante. La présence d'une couche cachée permet de modéliser des relations non linéaires entre les entrées et les sorties. En pratique, un modèle *PMC* comporte quelques dizaines à quelques centaines de neurones.

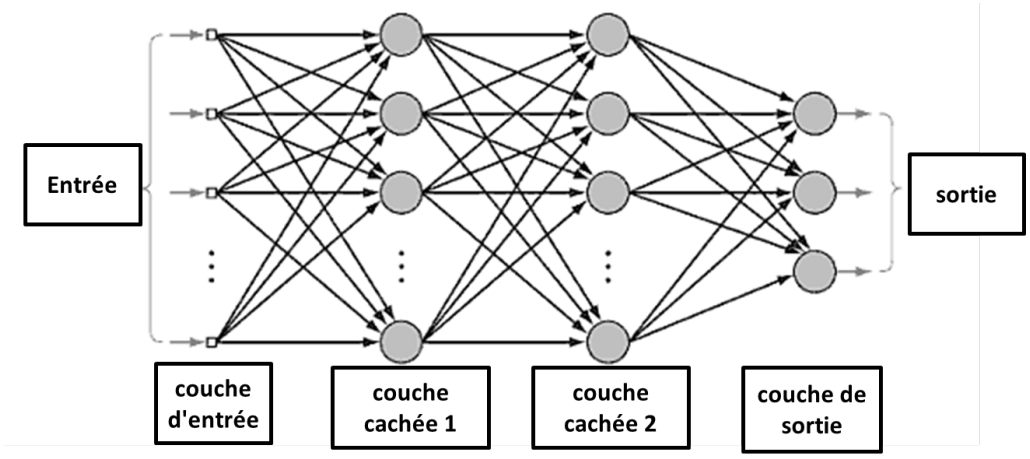

Figure 2.15: Une structure *PMC* typique avec deux couches cachées et trois sorties

2.6.3.6 Apprentissage des réseaux de neurones

Une étape importante avant la prédiction consiste à déterminer les valeurs des poids de connexion. Dans le cadre du *PMC*, les valeurs des poids de connexion sont choisies au hasard au début du processus. Ces valeurs sont mises à jour afin de minimiser l'erreur entre le résultat réel et le résultat prévu. Le calcul de l'erreur est requis au niveau des neurones pour effectuer des ajustements des valeurs de poids. L'erreur se propage vers l'arrière à travers le réseau après le calcul. Ce concept est connu sous le nom de rétro-propagation d'erreur. L'algorithme de descente en gradient (*Gradient descent*) est généralement utilisé pour déterminer plus efficacement les poids optimaux en agissant comme un guide lors de la recherche de la valeur optimale d'une fonction de coût.

2.6.4 Bilan

Dans cette thèse, nous proposons de mettre en place un module prédictif que nous considérons comme complément de la première phase d'exploration. Ce module exploite les données recueillies par simulation et permet d'améliorer l'adaptabilité du contrôleur en proposant un bon réglage de contrôle pour des contextes inconnus. Nous avons fait référence aux techniques de prédiction à base de réseau de neurones. Pour cela, nous proposons d'utiliser le perceptron multi-couches qui est l'un des réseaux de neurones les plus utilisés. Il est nécessaire, par la suite, de fixer le nombre de couches cachées et le nombre de neurones par couche cachée. Ces deux choix conditionnent directement le nombre de poids à estimer et donc la complexité du modèle. Concernant le choix de la fonction d'activation, de façon usuelle et pour les problèmes de régressions, la dernière couche, à savoir la couche de sortie, est généralement associée à une fonction d'activation *identité* tandis que les couches cachées sont munies de la fonction *sigmoïde*. L'estimation des valeurs de poids est effectuée par le biais d'une algorithme d'optimisation basé sur une évaluation du gradient par rétro-

propagation qui cherche à minimiser une fonction de perte quadratique. Grâce à la collecte et à l'analyse d'observations passées, le *PMC* tente d'extraire la relation entre les différents paramètres afin de proposer un modèle permettant de prédire des réglages pour des nouveaux contextes. La performance du modèle peut être évaluée grâce à plusieurs indicateurs, parmi lesquels, nous avons retenu le taux d'erreur quadratique moyenne *RMSE* et le carré moyen des erreurs *MSE*.

2.7 Bilan Général

Dans cette thèse, nous proposons une approche pour automatiser et faciliter la mise au point et le réglage des contrôleurs associés aux systèmes cyber-physiques évoluant en environnement incertain. Nous commençons par la modélisation du système cyber-physique dans sa globalité en prenant en considération son environnement, ses capteurs, ses actionneurs et son dispositif de commande (contrôleur). Pour ce faire, nous adoptons la modélisation à base de composants. Les composants sont utilisés pour faciliter la simulation et plus précisément pour aider l'exploration. Dans le but de mettre en oeuvre la variabilité de l'environnement (dégradation, incertitude, changement ...) dans lequel évolue le système, nous optons pour la définition de paramètres variables (généralement de type numérique) que l'on associe aux composants sous forme d'attributs. La variation de ces paramètres lors de la simulation rend possible l'exploration de différents contextes que le système cyber-physique pourrait rencontrer. En particulier, le contexte du contrôleur est composé de tous les paramètres associés aux composants qui ne sont pas contrôlés par le système (c'est-à-dire les paramètres associés aux capteurs, aux actionneurs et à l'environnement). Pour le contrôle des systèmes cyber-physiques, nous nous concentrons sur la technique de contrôle la plus utilisée dans l'industrie à savoir la régulation *PID*. Nous cherchons à automatiser la recherche de bons coefficients K_P, K_I et K_D pour différents contextes dans le but d'assurer un compromis entre l'exigence d'un contrôle rapide et le besoin d'un contrôle stable. L'idée est d'évaluer la qualité du contrôle pour différentes conditions environnementales ce qui exige l'élaboration de plusieurs simulations et le test de différentes configurations.
Les nombreux paramètres intervenant dans la dynamique globale du système entraînent une explosion combinatoire des cas possibles. Pour faire face à ce problème connu sous le nom de fléau de la dimension, nous proposons d'utiliser des algorithmes de sélection de caractéristiques pour réduire le nombre de contextes à considérer en ne conservant que les paramètres contextuels pertinents. Pour évaluer la pertinence des paramètres contextuels, nous optons pour une approche *wrapper* puisque cette approche est beaucoup plus performante que l'approche de type *filtre*. Plus précisément, nous retenons les deux méthodes *Step Forward Feature Selection* et *Step Backward Feature Selection* car elles sont particulièrement utilisées et outillées. Pour faciliter le réglage des contrôleurs, nous proposons par la suite d'effectuer un regroupement (clustering) des "bons" paramétrages de contrôle semblables par contexte en clusters. L'idée est d'associer à chaque cluster un seul bon paramétrage qui est considéré comme

étant un représentant du cluster. Nous adoptons l'intersection entre les différents ensemble de paramétrage possible comme mesure de similarité. Pour assurer une bonne qualité de clustering, nous imposons une contrainte sur la taille d'intersection entre deux ensembles de solution. Cette étape de clustering a comme principal objectif de minimiser le nombre de modes de fonctionnement du contrôleur. Nous complétons cette phase d'exploration par un module de prédiction. En exploitant les données historiques recueillies par simulation, nous proposons de mettre en place un réseau de neurone pour prédire un bon réglage de contrôle pour des contextes inconnues. Dans cette thèse, nous utilisons le perceptron multi-couches qui est l'un des réseaux de neurones les plus utilisés.

Chapitre 3

Contribution

3.1 Approche automatisée pour le réglage des paramètres des contrôleurs

3.1.1 Introduction

Dans ce chapitre nous présentons notre approche pour l'aide à la mise au point des paramètres des contrôleurs des systèmes cyber-physiques évoluant en environnement fortement perturbé. L'approche proposée s'appuie sur 8 étapes comme le montre la figure 3.1.

1. *Modeling* : le modèle du système cyber-physique doit suivre un style architectural minimal. Le modèle est constitué des éléments standards d'une boucle de contrôle à savoir : le système contrôlé, le contrôleur, les capteurs, les actionneurs et l'environnement. Comme le comportement du système cyber-physique dépend fortement de l'environnement dans lequel il évolue, l'inclusion d'un modèle de l'environnement dans la boucle de contrôle est ici nécessaire. Chaque élément modélisé appartient nécessairement à un et un seul de ces cinq éléments. Par la suite, on définit des paramètres et on les associe aux éléments du modèle. Pour chaque élément, on classe les paramètres (attributs numériques des composants) en deux catégories :

   - paramètres contextuels : ces paramètres décrivent le comportement de l'environnement, le système contrôlé, les capteurs et les actionneurs.
   - paramètres contrôlables : ce sont les paramètres des lois de contrôle

   Par la suite, pour chaque combinaison de paramètres contextuels, on cherche à tester différentes valeurs des paramètres de contrôle.

2. *Exploration* : Dans le but de guider l'exploration et pour éviter une exploration manuelle des paramètres, on fixe un minimum (*Min*) et un maximum (*Max*) pour chaque paramètre du modèle. Les paramètres ne peuvent alors évoluer que dans ces limites. Il est ensuite nécessaire d'adopter une politique de balayage de l'espace ainsi défini.

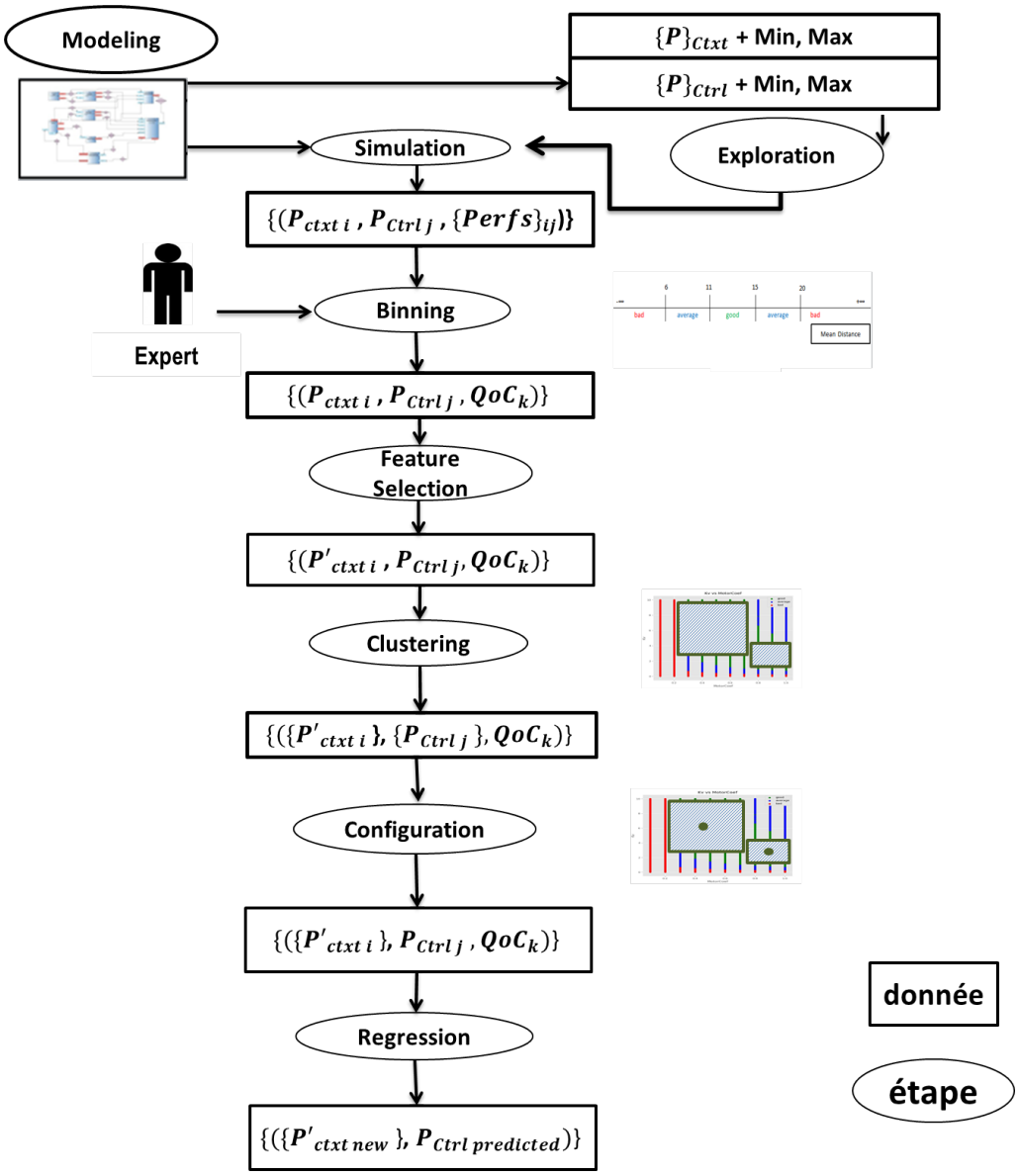

Figure 3.1: Les 8 étapes de l'approche proposée

3. *Simulation* : Une simulation prend en entrée une combinaison de valuations des paramètres contextuels et des paramètres de contrôles, combinaison appelée configuration. Chaque simulation produit un ensemble de mesures de performance sous forme de mesures statistiques .

4. *Binning* : Un expert du domaine définit les classes de qualité de contrôle qu'on les note *QoC* via des plages de mesure de performance sur les résultats de la simulation. A l'issue de cette étape, on associe à chaque *configuration* une classe de *QoC*.

5. *Feature Selection* : Pour éviter le problème de l'explosion combinatoire,

nous utilisons des algorithmes de réduction de dimensionnalité et plus précisément de sélection de caractéristiques. Ces algorithmes sont utilisés pour identifier les paramètres contextuels qui contribuent le plus au choix des "bons" paramètres de contrôle . Cette étape a comme objectif de réduire le nombre de contextes à explorer en sélectionnant un nombre fixé de paramètres contextuels inférieur au nombre de paramètres contextuels de départ qui contribuent le plus à l'association d'une configuration à une classe de *QoC*.

6. *Clustering* : On regroupe les valeurs des paramètres de contrôle qui sont valables pour un maximum de contextes. Ceci aboutit à un ensemble restreint mais pertinent de modes de fonctionnement. Un mode de fonctionnement est ici défini comme un paramétrage de la loi de contrôle pour plusieurs contextes.

7. *Configuration* : Pour chaque regroupement de configurations, on définit un unique paramétrage de la loi de contrôle.

8. *Regression* : Un module de régression permet l'évaluation de paramètres de contrôle pour les contextes non explorés (inconnus) dans les étapes précédentes.

3.2 Exploration hors ligne

3.2.1 Première étape : *modeling*

Un aspect important du développement des systèmes de contrôle de processus évoluant dans un environnement incertain est leur forte dépendance avec leur environnement. Pour des raisons de sécurité et de robustesse, la configuration du contrôle utilisée (la loi de commande et son paramétrage) doit pouvoir convenir à différents contextes. La multitude potentielle des contextes rend fastidieuse la conduite de tests en situation réelle pour aider à la mise au point des lois de contrôle. Dans notre approche, nous avons donc besoin d'un modèle du système cyber-physique et de son environnement afin de conduire des simulations.
Un modèle du système cyber-physique avec son environnement constitue le point de départ de notre approche. On impose que ce modèle suive un style architectural minimal. Le modèle doit contenir les éléments standards d'une boucle de contrôle : le contrôleur, les capteurs, les actionneurs, le système contrôlé et l'environnement. Chaque élément modélisé est associé à un et à un seul de ces cinq éléments. Pour des raisons de simplicité, nous optons pour la définition de paramètres variables numériques dans le but d'illustrer la variabilité du comportement du système cyber-physique ainsi que la variabilité de l'environnement dans lequel ce dernier évolue. On associe ces paramètres aux composants sous formes d'attributs. On classe les paramètres contenus dans le modèle en deux catégories :

- les paramètres contextuels qui sont liés aux *capteurs*, *actionneurs* et au *système contrôlé* et qui décrivent les changements dans l'*environnement*.
- les paramètres de contrôle qui apparaissent dans les lois de contrôle

Pour aider à la mise au point du contrôleur, on s'intéresse à l'exploration des divers contextes auxquels le système contrôlé pourra être confronté. Un *contexte* est défini comme une combinaison de valeurs de paramètres contextuels notés P_{Ctxt}. Pour un contexte spécifique, on cherche à tester différentes valeurs des paramètres de contrôle notés P_{Ctrl}. La valuation d'une combinaison de paramètres contextuels et de paramètres de contrôle définit une *configuration* et sera notée (P_{Ctxt_i}, P_{Ctrl_j}).
L'approche proposée est générique et ne se limite pas à un *framework* de modélisation spécifique. En effet, n'importe quel *framework* de modélisation qui suit un paradigme à base de composants peut être utilisé à partir du moment où il offre la possibilité d'identifier des paramètres de contexte et de contrôle.

3.2.2 Deuxième étape : *exploration*

A partir du modèle du système cyber-physique et de son environnement, on extrait les paramètres contextuels P_{Ctxt} qui décrivent les changements dans l'environnement ainsi que les paramètres de contrôles P_{Ctrl} à régler. Pour chaque paramètre contextuel P_{Ctxt} et de contrôle P_{Ctrl}, on fixe un minimum *Min*, un maximum *Max* et un pas *Step*. Chaque paramètre ne peut alors évoluer que dans cette plage :

$$\begin{cases} P = Min + k \cdot Step \quad avec\ k\ entier \\ P \leq Max \end{cases}$$

Les paramètres non sélectionnés sont considérés comme statiques et conservent leur valeur par défaut.
Pour explorer l'espace de valeurs des paramètres, on fixe une politique de balayage. Cette politique a comme objectif de guider l'exploration. Trois politiques de balayage sont possibles :

- considérer toutes les combinaisons de paramètres : balayage exhaustif des paramètres.
- considérer un balayage d'un paramètre à la fois : les autres paramètres sont fixes.
- considérer un certain nombre de tirages aléatoires (une combinaison aléatoire) de paramètres : cette politique nécessite la définition préalable du nombre maximal de simulations à effectuer.

A noter que la deuxième politique ne tient pas compte des interactions entre les différents paramètres alors que la troisième ne garanti pas l'exploration de tous les contextes possibles. Ces politiques d'exploration font l'hypothèse forte d'indépendance des paramètres entre eux. D'autres politiques d'exploration existent et pourraient être envisagées [Cla+16].

3.2.3 Troisième étape : *simulation*

Une fois la politique de balayage fixée, le simulateur est utilisé pour effectuer plusieurs simulations couvrant toutes les configuration possibles. Ce dernier effectue une simulation par configuration (i.e., pour chaque couple de (P_{Ctxt_i},P_{Ctrl_j})). Chaque exécution du simulateur produit un ensemble de mesures de performance qu'on le note $Perfs_{ij}$ sous forme de mesures statistiques (moyenne, écart type...). Ces mesures de performance sont utilisées pour qualifier les performances du contrôleur en terme de stabilité, de vitesse de réponse, d'erreur en régime permanent ou de prévention des oscillations.

3.2.4 Quatrième étape : *binning*

A partir des résultats de simulation, nous effectuons une étape de *binning* qui consiste à caractériser chaque configuration testée en termes de qualité de contrôle. Différentes classes de *QoC* sont définies via des plages de mesure de performance. Ces plages sont définies par un expert puisque la qualité de contrôle *QoC* dépend du domaine métier. Ce dernier doit en plus fournir un classement des classes de *QoC* de la pire ("bad" par exemple) à la meilleure ("good" par exemple). On impose que l'expert définisse au moins une "bonne" classe de *QoC*.Au cours de cette étape de *binning*, on affecte une classe de *QoC* à chaque configuration testée (P_{Ctxt_i},P_{Ctrl_j}) à selon les mesures de performance obtenues : $\{Perfs\}_{ij} \rightarrow QoC_k$.
Durant cette étape, on impose que chaque configuration soit affectée à une unique classe de *QoC* et que l'ensemble des classes de *QoC* couvre toutes les configurations possibles.
A la fin de cette étape, on obtient une table contenant toutes les configurations (P_{Ctxt_i},P_{Ctrl_j}) et leur classe correspondante de *QoC* (QoC_k). Pour mieux appréhender cette étape, supposons qu'on dispose d'un modèle de système contenant deux paramètres contextuels notés respectivement P_{Ctxt_1} et P_{Ctxt_2} dont les valeurs varient dans l'intervalle [1,5] avec un pas de 1 et deux paramètres de contrôle notés P_{Ctrl_1} et P_{Ctrl_2} variant dans l'intervalle [0.8,1] avec un pas de 0.1. Chaque valuation du couple (P_{Ctxt_1}, P_{Ctxt_2}) constitue un contexte unique. En considérant un balayage exhaustif, on obtient 25 (5×5) contextes (voir tableau 3.1).

Ctxt	P_{Ctxt_1} =1	P_{Ctxt_1} = 2	P_{Ctxt_1} =3 ,	P_{Ctxt_1} = 4	P_{Ctxt_1} = 5
	(P_{Ctxt_1}, P_{Ctxt_2})				
P_{Ctxt_2} = 1	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)
P_{Ctxt_2} = 2	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)
P_{Ctxt_2} = 3	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)
P_{Ctxt_2} = 4	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)
P_{Ctxt_2} = 5	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)

Table 3.1: Liste des contextes

Chaque valuation du quadruplet (P_{Ctxt_1},P_{Ctxt_2},P_{Ctrl_1},P_{Ctrl_2}) constitue une con-

figuration. Prenons à titre d'exemple le quadruplé (1,2,0.8,0.9) qui est associé au contexte (P_{Ctxt_1},P_{Ctxt_2}) = (1,2) et au paramétrage de contrôle (P_{Ctrl_1},P_{Ctrl_2}) = (0.8,0.9). Vu que l'on a trois réglages possibles (0.8,0.9 et 1) pour chaque paramètre de contrôle, on obtient 225 (25 $\times$ 3 $\times$ 3) configurations. Chaque configuration est ensuite simulée pour obtenir des mesures de performance $\{Perfs\}_{ij}$. Chaque configuration est associée par la suite à une classe de qualité de contrôle QoC_k. Supposons qu'on puisse classifier les configurations en trois classes de QoC notées QoC_1, QoC_2 et QoC_3. L'attribution d'une configuration à une classe est effectuée en comparant les mesures associées à chaque configuration avec les contraintes fixées par un expert. Ces contraintes désignent des conditions sur les mesures de performance $\{Perfs\}_{ij}$ permettant de classifier les configurations testées. Par exemple, une configuration est associée à la classe QoC_1 ("bonnes" performances) si et seulement si le temps de montée T_m correspondant varie entre 10 et 12 secondes et le temps d'établissement T_r correspondant est inférieur à 20 secondes. Une configuration est associée à la classe QoC_2 si et seulement si le temps de montée correspondant varie entre 13 et 17 secondes et le temps d'établissement correspondant varie entre 21 et 30 secondes. Pour s'assurer que les classes de QoC soient disjointes, toutes les configurations avec des mesures de performance en dehors de ces limites sont associées à la classe QoC_3 ("mauvaises" performances). Le tableau 3.2 illustre le résultat de l'étape de binning.

Num config	(P_{Ctxt_1},P_{Ctxt_2})	P_{Ctrl_1}	P_{Ctrl_2}	T_m	T_r	*classe de QoC*
$Config\_1$	(1,1)	0.8	0.8	10	15	QoC_1
$Config\_2$	(1,2)	0.8	0.8	12	16	QoC_1
$Config\_3$	(1,3)	0.8	0.8	11	14	QoC_2
$Config\_4$	(1,4)	0.8	0.8	14	22	QoC_1
$Config\_5$	(1,5)	0.8	0.8	15	22	QoC_1
...	..	..	..	..	..	..
$Config\_221$	(5,1)	1	1	18	25	QoC_3
$Config\_222$	(5,2)	1	1	14	22	QoC_2
$Config\_223$	(5,3)	1	1	20	28	QoC_3
$Config\_224$	(5,4)	1	1	19	34	QoC_3
$Config\_225$	(5,5)	1	1	15	32	QoC_3

Table 3.2: Exemple illustratif du résultat de l'étape de binning

3.2.5 Cinquième étape : *Feature selection*

La cinquième étape de l'approche consiste à appliquer un algorithme de *Feature selection* (sélection de caractéristiques) sur les données récupérées à l'issue de l'étape précédente. La sélection a pour but d'identifier les paramètres contextuels $\{P_{Ctxt}\}$ qui contribuent le plus à la variable de prédiction, la classe de QoC. La classe de QoC étant une variable dite catégorielle (quantitative), nous sommes face à un problème de classification qui consiste à sélectionner un nombre fixé de paramètres contextuels $\{P'_{Ctxt}\}$ (inférieur au nombre de

paramètres contextuels de départ) qui contribuent le plus à l'association d'une configuration à une classe de *QoC*. Pour ce faire, l'algorithme de sélection effectue une évaluation de la pertinence de chaque paramètre contextuel $\{P_{Ctxt}\}$ avec le paramètre cible ce qui permet de sélectionner les paramètres contextuels $\{P'_{Ctxt}\}$ qui sont les plus pertinents. Cette procédure nous permet d'éliminer les paramètres contextuels peu pertinents voir redondants pour le processus de classification des configurations. A l'issue de cette étape, on peut se focaliser exclusivement sur les paramètres sélectionnés réduisant ainsi l'espace de valeurs à explorer. En conséquence, nous créons une nouvelle table avec moins de paramètres contextuels $\{P'_{Ctxt_i}\}$ tout en gardant leurs mesures de performance et leur classe correspondante de *QoC* (QoC_k).
Dans le but de guider la recherche des paramètres contextuels pertinents, nous optons pour la génération heuristique de type *Forward* ou *Backward* pour des raisons de simplicité et de rapidité. Pour évaluer la pertinence de ces paramètres, nous optons pour une approche *wrapper*. Nous retenons les deux méthodes *Step Forward Feature Selection* et *Step Backward Feature Selection* car elles sont particulièrement utilisées et outillées. Dans notre approche, nous utilisons un de ces deux algorithmes :

- Step Forward Feature Selection: on commence par un ensemble de caractéristiques vide, puis on envisage d'ajouter une ou plusieurs caractéristiques.

Algorithm 1 Step Forward Feature Selection

1: **Entrées :**
2: $F = \{f_1, f_2, ..., f_N\}$: L'ensemble de toutes les caractéristiques
3: M : taille souhaitée du sous-ensemble sélectionné, doit être spécifié a priori
4: **Sorties :**
5: $E = \{f_{s1}, f_{s2}, ..., f_{sM}\}$
6: **Initialisation :**
7: $E = \varnothing$, f_{max} =f_1 (f_{max} *la meilleure caractéristique*)
8: **Inclusion :**
9: **Pour** i =1 à M **Faire**
10: **Pour** j =1 à $|F|$ **Faire**
11: **Si** E_v $(f_j \bigcup E) \geq E_v$ $(f_{max} \bigcup E)$ **alors**
12: $f_{max} = f_j$
13: **Fin Si**
14: **Fin Pour**
15: $E = E \bigcup f_{max}$, $F = F \backslash f_{max}$
16: **Fin Pour**
17: **Retourner** E

- Step Backward Feature Selection: on commence par l'ensemble contenant toutes les caractéristiques et on envisage de supprimer une ou plusieurs

caractéristiques de cet ensemble.

Algorithm 2 Step Backward Feature Selection

1: **Entrées :**
2: $F = \{f_1, f_2, ..., f_N\}$: L'ensemble de toutes les caractéristiques
3: M : taille souhaitée du sous-ensemble sélectionné, doit être spécifié a priori
4: **Sorties :**
5: $E = \{f_{s1}, f_{s2}, ..., f_{sM}\}$
6: **Initialisation:**
7: $E = F$, f_{min} = f_1 (f_{min} *la caractéristique la plus mauvaise*)
8: **Exclusion :**
9: **Pour** i =1 à N-M **Faire**
10: **Pour** j =1 à $|E|$ **Faire**
11: **Si** E_v $(E \setminus f_j) \leq E_v$ $(E \setminus f_{min})$ **alors**
12: $f_{min} = f_j$
13: **Fin Si**
14: **Fin Pour**
15: $E = E \setminus f_{min}$
16: **Fin Pour**
17: **Retourner** E

Revenons à l'exemple illustratif précédent, on part d'un ensemble de configurations dont chacune a été associée à un classe de *QoC* unique. Nous appliquons sur ce jeu de données un algorithme de sélection de caractéristiques afin de déterminer entre les deux paramètres contextuels P_{Ctxt_1} et P_{Ctxt_2} celui qui contribue le plus à l'attribution d'une configuration $Config\_i$ à une classe QoC_k donnée (voir tableau 3.3). Si le processus de sélection a choisi le paramètre P_{Ctxt_1}, par la suite, on se concentre uniquement sur ce paramètre contextuel. A l'issue de cette étape, on obtient une nouvelle table de configurations ne contenant que le paramètre contextuel sélectionné P_{Ctxt_1}, les paramètres de contrôle P_{Ctrl_1} et P_{Ctrl_2}, leurs mesures de performance et leurs classes de *QoC* associées . Cette étape de sélection de features peut être vue comme l'équivalent de l'étude de sensibilité utilisée en théorie du contrôle [BKS66].

Num config	P_{Ctxt_1}	P_{Ctrl_1}	P_{Ctrl_2}	T_m	T_r	*classe de QoC*
$Config_1$	1	0.8	0.8	10	15	QoC_1
$Config_2$	1	0.8	0.9	12	16	QoC_1
$Config_3$	1	0.8	1	11	14	QoC_1
$Config_4$	1	0.9	0.8	14	25	QoC_2
$Config_5$	1	0.9	0.9	13	32	QoC_3
...	..	..	..	..	..	..
$Config_{41}$	5	0.9	0.9	22	35	QoC_3
$Config_{42}$	5	0.9	1	14	25	QoC_2
$Config_{43}$	5	1	0.8	11	18	QoC_1
$Config_{44}$	5	1	0.9	10	15	QoC_1
$Config_{45}$	5	1	1	20	28	QoC_3

Table 3.3: Exemple illustratif du résultat de l'étape *Feature selection*

3.2.6 Sixième étape : *Clustering*

On part du principe qu'un réglage de contrôle approprié pour un contexte particulier (spécifique) peut ne pas être adapté pour un autre. Pour autant, il existe des situations ou un même réglage peut convenir pour plusieurs contextes. Dans ce cas, il est inapproprié de changer le réglage de contrôle suite à un changement du contexte (changement dans l'environnement, dans l'état des capteurs ou des actionneurs). En faisant ainsi, on risque de pousser le système contrôlé vers l'instabilité en terme de comportement puisque ce dernier nécessite un temps supplémentaire pour se stabiliser suite à chaque changement de réglage. Dans ce cadre, nous suggérons de regrouper les contextes conformément au deux principes suivants :

- on cherche à établir les valeurs de paramètres de contrôle aboutissant à la même classe de *QoC* pour différents contextes.
- si deux contextes partagent un nombre suffisant de valeurs de paramètres de contrôle, on peut regrouper ces contextes en considérant que ces derniers représentent le même mode de fonctionnement. On associe alors un seul réglage à ces deux contextes ce qui permet au système contrôlé d'avoir le même comportement du point de vue qualité de contrôle *QoC*.

Afin d e p roposer u n e nsemble r estreint m ais p ertinent d e " modes" de fonctionnement, on essaye de regrouper les contextes ayant des valeurs de paramètres de contrôle communes. Dans cette perspective, nous avons l'idée d'étudier l'intersection des espaces de réglages de contrôle pour différents contextes, puis de choisir une valeur représentative (aux paramètres de contrôle) à tous ces contextes. Pour cette étape, nous ne considérons que les paramètres contextuels $\{P'_{Ctxt_i}\}$ sélectionnés lors de l'étape précédente.
L'entrée de l'algorithme proposé est un ensemble de triplets composés des valeurs des paramètres contextuels, des valeurs des paramètres de contrôle et de la classe de QoC (P'_{Ctxt_i}, P_{Ctrl_j},QoC_k).
Dans un premier temps, nous considérons les configurations d'une même

classe de QoC. Pour chaque classe QoC_k, et pour chaque contexte P'_{Ctxt_i} de cette cette classe, nous considérons l'ensemble des valeurs correspondantes des paramètres de contrôle. Par conséquent, pour chaque classe de QoC_k, pour chaque contexte P'_{Ctxt_i}, un ensemble de valeurs de paramètres de contrôle $\{P_{Ctrl_j}\}_{ik}$ est défini. Pour n paramètres de contrôle, une liste de n-uplets contenant toutes les combinaisons est construite pour chaque contexte P'_{Ctxt_i} et pour chaque classe de QoC_k.

Algorithm 3 algorithme de clustering

1: **Entrées :**
2: Paramètres : Liste <double> , *percentage_of_confidence*
3: **Sorties :**
4: Liste <double> : $clpfinal_k$
5: **Pour** chaque classe de qualité de contrôle QoC_k **Faire**
6: clp_k = Map <contexte, Liste <paramétrage de contrôle>, QoC_k >
7: **Tant que** taille (clp_k) ≥ 0 **Faire**
8: $max_{index} = 0$
9: $taille_{max} = 0$
10: **Pour** i =2 à n **Faire**
11: intersection = clp_k ($Ctrl_1$) $\cap$ $clp_k(Ctrl_i)$
12: **Si (** length(intersection) $\leq$ confidence $\times$ length($clp_k(Ctrl_1)$ ET
13: length(intersection) $\leq$ confidence $\times$ length(clp_k ($Ctrl_i$))) **)** **alors**
14: intersection = []
15: **Sinon**
16: **Si** (length(intersection) > $taille_{max}$) **alors**
17: $taille_{max}$ = length(intersection)
18: max_{index} = i
19: **Fin Si**
20: **Fin Si**
21: **Fin Pour**
22: **Si** length($Ctrl_1 \cap Ctrl_{max_{index}}$) $\neq 0$ **alors**
23: clp_k.remove($Ctrl_1$)
24: clp_k.remove($Ctrl_{max_{index}}$)
25: clp_k.insert($Ctrl_1 \cap Ctrl_{max_{index}}$)
26: $clpfinal_k$.add($Ctrl_1 \cap Ctrl_{max\_index}$)
27: **Sinon**
28: clp_k.remove($Ctrl_1$)
29: $clpfinal_k$.add($Ctrl_1$)
30: **Fin Si**
31: **Fin faire**
32: **Fin Pour**

Le processus de clustering proposé tient compte de la taille d'intersection des ensembles. Un paramètre nommé *percentage_of_confidence* est créé pour

conditionner que deux ensembles puissent être regroupés. Les valeurs possibles de ce paramètre varient entre 0 et 1. Comme son nom l'indique, ce paramètre représente un pourcentage. Ainsi, la valeur 0.1 est équivalente à 10%, tandis que 1 est équivalente à 100%. Pour une classe QoC_k donnée, considérant deux contextes $P'_{Ctxt_{i1}}$ et $P'_{Ctxt_{i2}}$, si l'intersection de leurs deux ensembles de contrôle correspondant $\{P_{Ctrl_j}\}_{i1k}$ et $\{P_{Ctrl_j}\}_{i2k}$ a une taille supérieure à la taille (min des tailles) des deux ensembles multipliée par la valeur *percentage_of_confidence* , nous considérons que les deux ensembles peuvent être regroupés. Sinon, (i.e.,la taille d'intersection est trop petit) dans ce cas, (logiquement) seules les frontières des deux ensembles se chevauchent et pour ne pas prendre de risque dans ce cas on considère que les deux contextes ne peuvent pas être regroupés. Le regroupement n'est pas considéré.
On part de l'ensemble de paramètres de contrôle P_{Ctrl_1} associé au premier contexte P'_{Ctxt_1}, nous calculons la taille d'intersection avec tous les autres ensembles de contrôle $P_{Ctrl_{j\neq 1}}$. Dans la liste des tailles d'intersection, le contexte associé à la taille maximale $P'_{Ctxt_{maxInter}}$ sera regroupé avec le premier contexte P'_{Ctxt_1} (i.e., on regroupe le premier contexte avec le contexte avec la plus grande intersection avec lui) pour former le nouveau contexte noté $P'_{Ctxt_{1\cap maxInter}}$ et les valeurs des paramètres de contrôle associées seront le résultat de l'intersection de l'ensemble des valeurs de paramètres de contrôle respectifs noté $P_{Ctrl_{1\cap maxInter}}$. Ces deux contextes seront remplacés par leur intersection $P'_{Ctxt_{1\cap maxInter}}$ et par son ensemble de paramètres de contrôle correspondant dénoté $P_{Ctrl_{1\cap maxInter}}$. Le nouveau contexte issu du regroupement précédent est renuméroté 1. Cette procédure est itérée en commençant par calculer la taille d'intersection entre l'ensemble de contrôle associé au contexte nouvellement groupé avec tous les autres ensembles de contrôle. Si aucune intersection n'est trouvée en premier lieu, le contexte 1 est associé à un mode de fonctionnement séparé. Dans ce cas le processus reprend à partir de l'ensemble de contrôle P_{Ctrl_2} associé au deuxième contexte P'_{Ctxt_2}. A chaque fois, on élimine un élément de la liste pour recommencer sur un ensemble plus petit ce qui garantit la terminaison de l'algorithme. Ce processus de clustering est mené pour chaque classe de QoC, jusqu'à ce qu'il ne soit plus possible de regrouper des contextes.
Pour mieux appréhender cette première phase de la procédure de clustering, et en revenant à l'exemple pris, nous associons, pour chaque QoC_k, à chaque valeur du paramètre contextuel sélectionné P_{Ctxt_1} une liste de tous les doublets formés par les valeurs des deux paramètres de contrôle P_{Ctrl_1} et P_{Ctrl_2} (Voir figure 3.2).

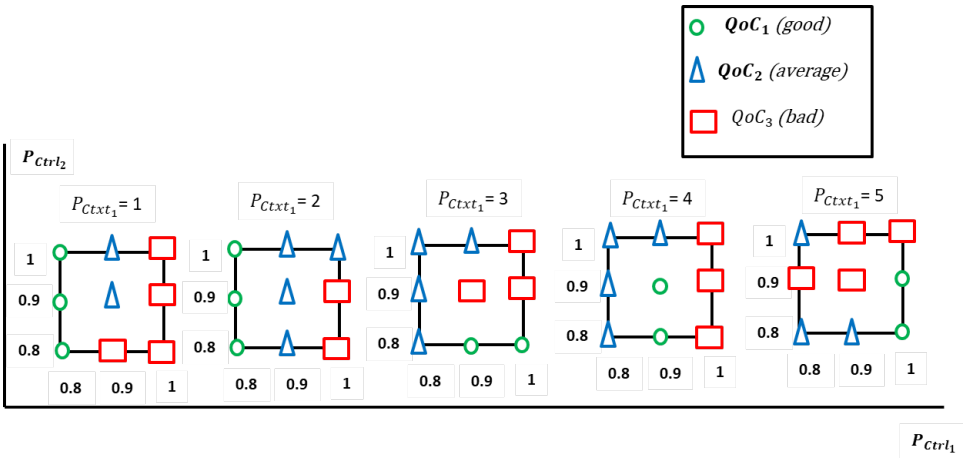

Figure 3.2: Représentation graphique du résultat de l'étape de Binning

Le tableau 3.4 ci-dessous présente les résultats de la première phase de clustering pour la classe QoC_1.

Contexte	*Paramètres de contrôle regroupés*	*Classe de QoC*
P_{Ctxt_1}=1	[(0.8, 0.8), (0.8, 0.9), (0.8, 1.0)]	QoC_1
P_{Ctxt_1}=2	[(0.8, 0.8), (0.8, 0.9), (0.8, 1.0)]	QoC_1
P_{Ctxt_1}=3	[(1.0,0.8) , (0.9, 0.8)]	QoC_1
P_{Ctxt_1}=4	[(0.9, 0.8), (0.9, 0.9)]	QoC_1
P_{Ctxt_1}=5	[(1.0,0.8), (1.0, 0.9)]	QoC_1

Table 3.4: La première phase de l'étape de clustering pour la classe QoC_1

La deuxième phase consiste à regrouper les contextes en inspectant les chevauchements des ensembles (voir figure 3.3). La mesure de similarité adoptée par notre algorithme est la taille de l'intersection. Comme le nombre de clusters ne peut pas être connu à l'avance, l'algorithme conçu ne l'impose pas comme une entrée du problème.

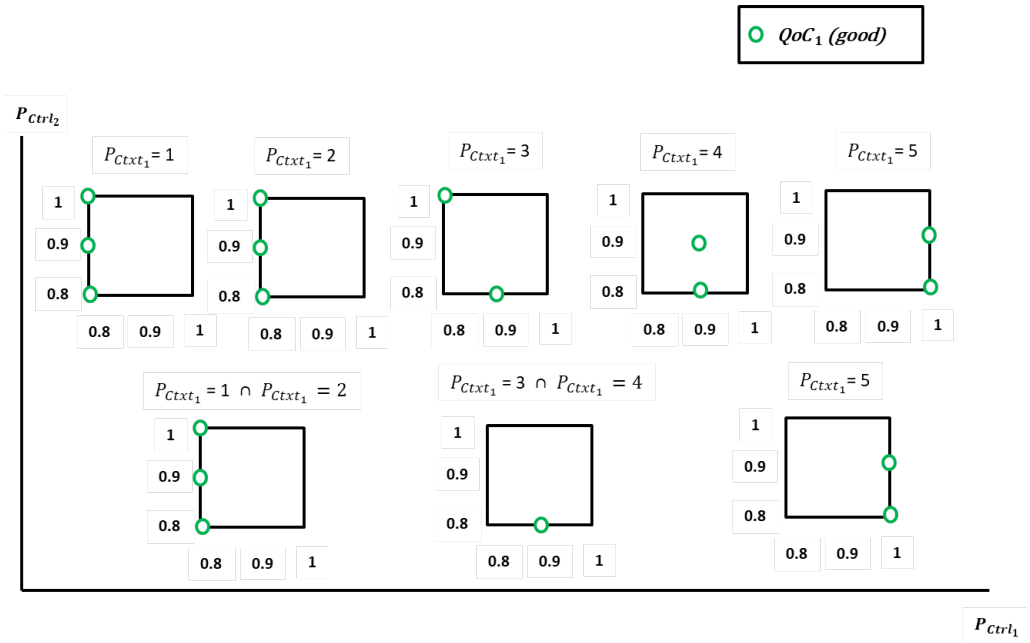

Figure 3.3: Intersection des ensembles pour un percentage_of_confidence inférieur à 0.5

Revenons à l'exemple précédent, à l'issue de cette phase nous obtenons des résultats comme le montre le tableau 3.5 pour un *percentage_of_confidence* inférieur à 0.5. Avec une valeur de *percentage_of_confidence* inférieur à 0.5, il est possible de regrouper les contextes P_{Ctxt_1}=1 et P_{Ctxt_1}=2 d'une part et les contextes dont les valeurs sont P_{Ctxt_1}=3 et P_{Ctxt_1}=4 d'autre part. Nous obtenons ainsi 3 modes de fonctionnement.

Contextes regroupés	*Paramètres de contrôle regroupés*	*Classe de QoC*
P_{Ctxt_1}=1 & P_{Ctxt_1}=2	[(0.8, 0.8), (0.8, 0.9), (0.8, 1.0)]	QoC_1
P_{Ctxt_1}=3 & P_{Ctxt_1}=4	[(0.9, 0.8)]	QoC_1
P_{Ctxt_1}=5	[(1.0, 0.8), (1.0, 0.9)]	QoC_1

Table 3.5: La deuxième phase de l'étape de clustering pour un percentage_of_confidence inférieur à 0.5

Par contre, pour un *percentage_of_confidence* supérieur à 0.5, il n'est pas possible de regrouper les contextes P_{Ctxt_1}=3 et P_{Ctxt_1}=4. On se retrouve donc avec 4 modes de fonctionnement (voir tableau 3.6).

Contextes regroupés	*Paramètres de contrôle regroupés*	*Classe de QoC*
P_{Ctxt_1}=1 & P_{Ctxt_1}=2	[(0.8, 0.8), (0.8, 0.9), (0.8, 1.0)]	QoC_1
P_{Ctxt_1}=3	[(0.8,1.0), (0.9, 0.8)]	QoC_1
P_{Ctxt_1}=4	[(0.9, 0.8), (0.9,0.9)]	QoC_1
P_{Ctxt_1}=5	[(1.0, 0.8), (1.0, 0.9)]	QoC_1

Table 3.6: La deuxième phase de l'étape de clustering pour un percentage_of_confidence supérieur à 0.5

À la fin de cette étape, nous nous retrouvons avec moins de contextes et des ensembles de valeurs de paramètres de contrôle plus restreints. Ceci va nous permettre de définir un nombre limité de modes de fonctionnement.

3.2.7 Septième étape : *Configuration*

A l'issue de l'étape précédente, nous avons défini un ensemble de modes de fonctionnement. Un mode de fonctionnement correspond à un ensemble de contextes (contextes regroupés $P'_{Ctxt_\cap}$) un ensemble de paramètres de contrôle (intersection d'ensembles de paramètres de contrôle de contextes groupés) pour une classe donnée de qualité de contrôle *QoC*. Chaque contexte que nous n'avons pas réussi à regrouper, est associé à son propre mode de fonctionnement.
Suite à cette étape, nous choisissons maintenant une valuation représentative des paramètres de contrôle pour chaque mode de fonctionnement (i.e.,un n-uplet de paramètres de contrôle parmi ceux récupérés après la procédure de clustering). En considérant l'espace défini par les valeurs possibles des paramètres de contrôle, les valeurs limites ne paraissent pas rassurantes et adéquates. Gardant à l'esprit la différence entre simulation et situation réelle, on évite de prendre des valeurs situées en bordure puisque ce choix est risqué (ces valeurs peuvent ne pas être valides). On choisit donc une valeur médiane de l'espace formé par chaque ensemble de paramètres de contrôle regroupés $P_{Ctrl_\cap}$. Pour un ensemble convexe, on choisit le point le plus proche du barycentre. Ainsi, on obtient un seul n-uplet par contexte groupé comme valeur possible. Au moins, on obtient un mode de fonctionnement (pour chaque classe de *QoC*). Nous sélectionnons ensuite les modes de fonctionnement associés à la meilleure classe de *QoC* (la classe *good*) et nous considérons les valeurs des paramètres de contrôle correspondantes.
Revenons à l'exemple illustratif précédent, pour avoir un nombre minimal de modes de fonctionnement, nous devons fixer le percentage_of_confidence à une valeur inférieure à 0.5. Dans ce cas, le doublet (P_{Ctrl_1} = 0.8 , P_{Ctrl_2} = 0.9) peut être choisi pour le premier mode de fonctionnement. Le paramétrage (P_{Ctrl_1} = 0.9 , P_{Ctrl_2} = 0.8) est choisi pour le deuxième mode de fonctionnement. Tandis que (P_{Ctrl_1} = 1 , P_{Ctrl_2} = 0.8) est associé à un contexte P_{Ctxt_1}=5 (Voir tableau 3.7).

mode	*Nbr de contextes*	*contexte*	P_{Ctrl_1}	P_{Ctrl_2}
1	2	1 & 2	0.8	0.9
2	2	3 & 4	0.9	0.8
3	1	5	1	0.8

Table 3.7: Paramétrage trouvé pour un *percentage_of_confidence* inférieur à 0.5

Les résultats obtenues pour un percentage_of_confidence supérieur à à 0.5 sont présentés dans le tableau 3.8.

mode	*Nbr de contextes*	*contexte*	P_{Ctrl_1}	P_{Ctrl_2}
1	2	1 & 2	0.8	0.9
2	1	3	0.8	1
3	1	4	0.9	0.8
4	1	5	1	0.8

Table 3.8: Paramétrage trouvé pour un *percentage_of_confidence* supérieur à 0.5

3.2.8 Exploration itérative

L'approche proposée est appliquée de manière itérative afin de préciser la limite d'adaptabilité du contrôleur. La limite d'adaptabilité est définie lorsque pour un contexte donné, la seule classe de *QoC* est "bad", c'est à dire que nous ne parvenons pas à trouver un "bon" réglage de contrôle pour ce contexte. Pour ce faire, on commence par une valeur de *step* élevée (un dixième de la longueur de l'intervalle par exemple). Les contextes adjacents pour lesquels la QoC vaut "bad" sont regroupés en régions (de fait des intervalles). On itère le processus sur ces régions en choisissant une valeur de *step* plus petite que la précédente (un dixième de la longueur de l'intervalle par exemple). En trouvant les "bons" paramètres de contrôle et les limites d'adaptabilité du contrôleur, nous produisons un contrôleur qui peut fonctionner dans différents contextes et nous pouvons également préciser les contextes pour lesquels aucun contrôle n'est possible.

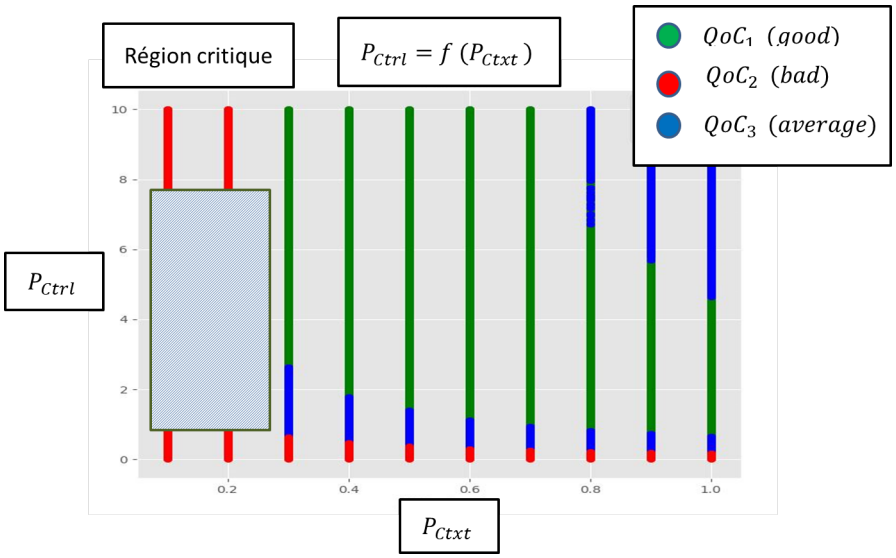

Figure 3.4: Limite d'adaptation

3.3 Prédiction en ligne

3.3.1 Huitième étape : *Regression*

Nous étendons l'approche proposée par un module de prédiction. Ce module est utilisé pour l'attribution de "bons" paramètres de contrôle P_{Ctrl} pour des contextes inconnus c'est-à-dire non explorés dans l'approche précédente. Il permet de modéliser la relation entre les variables *dépendantes* (cibles) qui correspondent aux "bons" paramètres de contrôle P_{Ctrl} (i.e., paramètres de contrôle appartenant à la meilleure classe de *QoC*) et les variables *indépendantes* appelées aussi prédicteurs qui correspondent dans notre cas à des paramètres contextuels non testés $\{P''_{Ctxt_i}\}$. Il s'agit donc d'un mécanisme pour ajuster les paramètres du contrôle pour des contextes non testés (inconnus).
Ce problème peut être vu comme une régression multiple multivariée (en anglais multivariate multiple regression). Les paramètres de contrôle nouvellement proposés sont calculés sur la base des données historiques des paramètres de contrôle et de leurs contextes correspondants collectés précédemment. A noter que les prédictions proposées par le modèle de régression ne sont valables que pour la plage de données utilisée pour estimer le modèle. Cela s'explique par le fait que la relation entre les variables indépendantes ($\{P''_{Ctxt_i}\}$) et les variables dépendantes ($P_{Ctrl_{bons}}$) peut changer en dehors de cette plage.
Dans le but de mettre en place un modèle de régression, nous suivons les étapes ci-dessous (voir figure 3.5) :

- Étape 1 : importation des données
- Étape 2 : partition des données en ensemble d'apprentissage et de test
- Étape 3 : construction du modèle à partir de l'ensemble d'apprentissage

- Étape 4 : application du modèle à l'ensemble de test
- Étape 5 : comparaison des valeurs observées et prédites de l'ensemble de test
- Étape 6 : Évaluation du modèle de prédiction au travers de son taux d'erreur

Les données sont composées de l'ensemble de paramètres contextuels P_{Ctxt} et de paramètres de contrôle P_{Ctrl} appartenant à la meilleure classe de *QoC*. Le principal objectif de la régression est de construire un modèle prédictif et plus précisément de trouver une fonction paramétrée reliant les variables prédictives (P_{Ctxt}) aux variables cibles à prédire (P_{Ctrl}). Ces données ne peuvent pas servir à la constitution du modèle et à l'évaluation. La solution consiste à la subdivision aléatoire des données en ensemble d'apprentissage et de test avec approximativement des proportions à 80% et 20% respectivement (pratique usuelle). L'ensemble d'apprentissage est utilisé pour la construction du modèle tandis que l'ensemble de test est utilisé pour l'évaluation du modèle (taux d'erreur). Le modèle doit assurer la minimisation de l'erreur entre les valeurs observées et les valeurs prédites. Il est nécessaire de disposer d'un critère numérique pour mesurer la qualité du modèle prédictif.

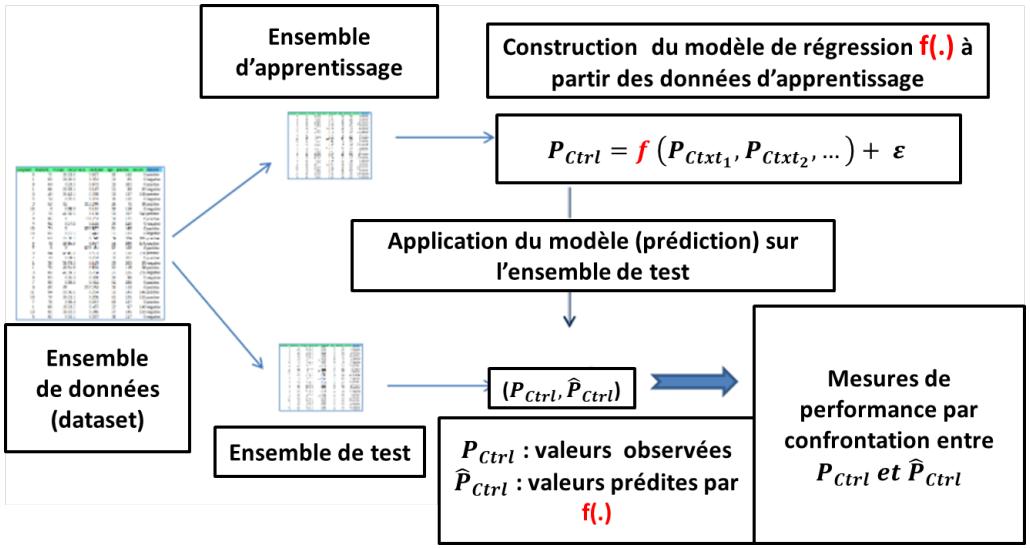

Figure 3.5: Analyse prédictive

Pour notre objectif, nous avons choisi d'utiliser un réseau de neurones car il peut fournir une puissance de prédiction beaucoup plus importante qu'une régression traditionnelle. Ceci est dû au fait que ce dernier peut apprendre rapidement à partir de données complexes (des grandes bases de données) et en déduire son modèle (statistique) et sa tendance . Plus précisément, nous utilisons le perceptron multi-couches *PMC* qui est l'un des réseaux de neurones les plus utilisés. Il est alors nécessaire de fixer le nombre de couches cachées et le nombre de neurones par couche cachée. Ces deux choix conditionnent directement

le nombre de poids à estimer et donc la complexité du modèle. L'estimation des valeurs de poids est effectuée par le biais d'un algorithme d'optimisation basé sur une évaluation du gradient par rétro-propagation qui cherche à minimiser une fonction de perte quadratique. Pour les problèmes de régressions et de manière conventionnelle, la dernière couche du réseau, à savoir la couche de sortie, est généralement associée à une fonction d'activation *identité* tandis que les couches cachées sont munies de la fonction *ReLU*. Grâce à la collecte et à l'analyse d'observations passées, le *PMC* tente d'extraire la relation entre les différents paramètres afin de proposer un modèle permettant de prédire des "bons" réglages pour des contextes inconnus. La performance du réseau de neurones *PMC* créé est ensuite évaluée en utilisant la racine de l'erreur quadratique moyenne *RMSE* et le carré moyen des erreurs *MSE* sur ces prédictions. Les "bons" paramètres de contrôle prédits ($P_{Ctrl_{bons}}$) associés à une combinaison inconnue de paramètres contextuels ($\{P''_{Ctxt_i}\}$) sont testés par simulation afin de s'assurer que les valeurs prédites répondent aux objectifs du contrôleur. Il est important de souligner que la mise au point du réseau de neurones (i.e., choix du nombre de couches cachées et du nombre de nœuds dans chaque couche cachée) nécessite une expérimentation systématique car ces choix dépendent de la taille des données utilisées pour l'apprentissage du modèle.

3.4 Outillage

L'ensemble de la proposition a été outillée. Le point d'entrée de l'outil correspond aux données de simulation des différentes configurations t estées. Ces données collectées doivent être enregistrées dans un fichier log au format excel ou CSV (séparateur : ";"). Chaque ligne correspond à une configuration ($\{P_{Ctxt}\}$ et $\{P_{Ctrl}\}$) et leurs mesures de performance associées. L'en-tête de ce fichier contient les noms de chaque colonne. La première ligne contient d'abord les noms des paramètres contextuels ($\{P_{Ctxt}\}$, puis les noms des paramètres de contrôle $\{P_{Ctrl}\}$ et enfin les noms des mesures de performance $\{Perfs\}_{ij}$. A partir de la seconde ligne, nous disposons les valeurs des configurations et les mesures de performance. On exige que toutes les données soient de type *double*.
A partir de ce fichier, un ensemble d'outils a été réalisé au travers de scripts *Python*. Ce choix est motivé par la facilité d'utilisation de ce langage, la présence d'une grande communauté et notamment le nombre important de bibliothèques dédiées à l'analyse de données. Le nombre de bibliothèques disponibles avoisinent les 70 000 bibliothèques.
Trois scripts ont été développés pour implémenter et évaluer les différents algorithmes de feature selection, de clustering et de régression. Pour analyser les données de simulations, nous nous sommes appuyés sur la librairie *Pandas* (en anglais Python Data Analysis Library) [McK10]. *Pandas* est une librairie Python open-source conçue pour faciliter l'analyse et la manipulation des données (lecture et visualisation). *Pandas* prend en entrée les données du fichier log quel que soit son format CSV ou Excel pour créer un objet Python appelé *DataFrame*. Cette structure est multidimensionnelle. Une fois ceci fait, on est capable d'exploiter les données, les représenter à l'aide de graphique et y appliquer des algorithmes

de machine learning.
Dans un premier lieu, nous commençons par associer chaque configuration à une classe de QoC_k. cette étape de binning est effectuée en utilisant la fonction *apply*. Pour chaque ligne du *DataFrame*, cette dernière compare les mesures de performances $\{Perfs\}_{ij}$ de type flottant avec les ranges fixés par l'expert de type flottant. La qualité de contrôle *QoC* est une variable de type chaîne de caractère. A l'issue de cette comparaison, une nouvelle colonne avec l'entête *QoC* est créée à la suite des colonnes existantes. Chaque configuration est associée à une classe de *QoC* tout en respectant les règles fixées auparavant. Cette étape de binning est nécessaire pour les deux scripts de feature selection et de clustering (Voir figure 3.6).

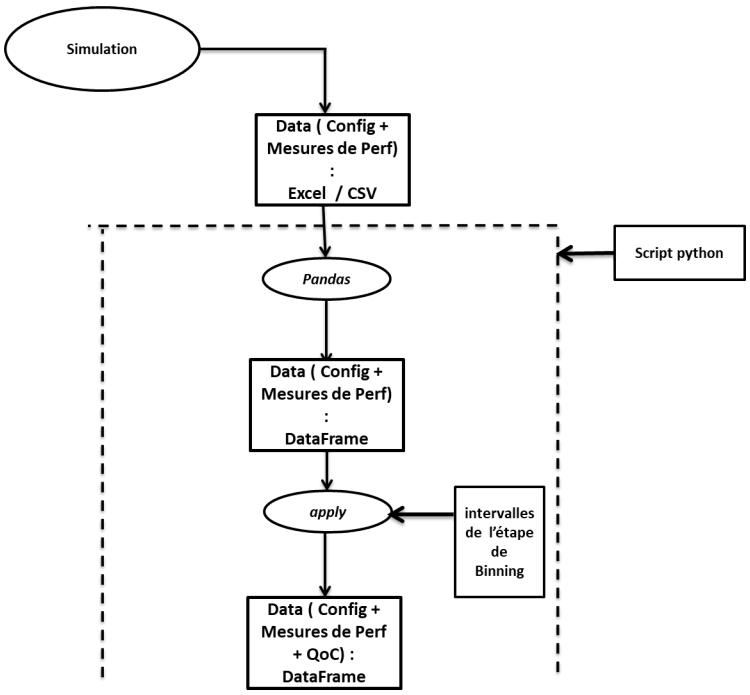

Figure 3.6: Schéma explicatif de l'étape de binning

3.4.1 Feature selection

Pour ce premier script, nous avons eu recours à la bibliothèque *scikit-learn* [Ped+11; Bui+13] destinée à l'apprentissage automatique. Cette bibliothèque fournit une gamme d'algorithmes d'apprentissage supervisés et non supervisés via une interface cohérente en *Python*. *Scikit-learn* dispose d'une documentation fournissant de nombreux exemples, une API uniforme entre tous les algorithmes. Elle dispose d'une grande communauté (800 contributeurs référencés sur GitHub). De plus, cette bibliothèque est intégrée dans la Librairie Pandas. Pour cette étape de *feature selection*, l'entrée de l'algorithme est constituée des paramètres contextuels ($\{P_{Ctxt}\}$, et est représentée par X, tandis que la variable cible est la classe de *QoC* qui est représentée par Y. Dans un pre-

mier temps, le module *model_selection* de *scikit-learn* divise le dataset en ensemble d'apprentissage et ensemble de test. Ceci permet de préparer les données pour le traitement avec un algorithme de classification (*RandomForestClassifier* par exemple). Par la suite, nous importons un *SequentialFeatureSelector* du package mlxtend (pour machine learning extensions) de Sebastian Raschka [Ras18]. L'outil *SequentialFeatureSelector* procède de manière ascendante (forward) ou descendante (backward) pour identifier le meilleur sous-ensemble de features (dans notre cas les paramètres contextuels ($\{P_{Ctxt}\}$ qui contribuent le plus à l'attribution d'une configuration à une classe de *QoC*). En mettant le paramètre *forward* (de type booléen) à "True" on obtient l'algorithme *Step Forward Feature Selection*. Dans le cas ou *forward* est "False", on parle de l'algorithme *Step Backward Feature Selection* (voir figure 3.7). Le paramètre *k_features* (de type entier ou tuple) permet de préciser le nombre maximal de features à sélectionner. *mlxtend* propose plusieurs indicateurs de performance pour évaluer le classificateur. Ceci est effectué à travers le paramètre *scoring* qui prend comme valeur une chaîne de caractères ("accuracy", "precision" ou "roc_auc") . Par défaut, on utilise l'*accuracy* de *scikit-learn*. Dans notre cas, la performance du classificateur est évalué en terme de précision (accuracy).

```
#Forward Feature Selection
from sklearn.ensemble import RandomForestClassifier
from mlxtend.feature_selection import SequentialFeatureSelector
forwarddModel = SequentialFeatureSelector(RandomForestRegressor(n_jobs=-1),
            k_features=2,
            forward=True,
            verbose=2,
            scoring='accuracy',
            cv=4)
forwardfeatures = forwarddModel.fit(np.array(X_train),y_train)
filtered_features_forward = X_train.columns[list(forwardfeatures.k_feature_idx_)]

#Backward Feature Selection
from sklearn.ensemble import RandomForestClassifier
from mlxtend.feature_selection import SequentialFeatureSelector
backwardModel = SequentialFeatureSelector(RandomForestClassifier(n_jobs=-1),
            k_features=2,
            forward=False,
            verbose=2,
            scoring='accuracy',
            cv=4)
backwardfeatures = backwardModel.fit(np.array(X_train), y_train)
filtered_features_backward = X_train.columns[list(backwardfeatures.k_feature_idx_)]
```

Figure 3.7: Forward et Backward Feature Selection

3.4.1.1 clustering

Le deuxième script est dédié à l'implémentation de l'algorithme de clustering décrit auparavant. Pour le clustering, le paramètre à préciser est le *percentage_of_confidence*. N ous c ommençons p ar a ssocier à c haque combinaison de valeurs des paramètres contextuels une liste de toutes les combinaison de paramètres de contrôle par classe de *QoC*. Ceci est fait à travers la fonction *itertuples* proposée par *pandas*. Par la suite, nous inspectons le chevauchement des

listes de paramètres de contrôle comme décrit auparavant tout en prenant en considération le paramètre *percentage_of_confidence*. Ce script génère un fichier en format *txt* contenant la liste des contextes regroupés ainsi que la configuration associée.

3.4.1.2 Régression

Le troisième script est destiné à implanter le module de régression. Les données historiques nécessaires pour définir le modèle de régression et de le tester sont stockées dans un fichier au format CSV ou excel. Comme pour les 2 scripts précédents, la librairie *Pandas* est utilisée pour convertir ce fichier en *DataFrame*. Pour mettre en place le modèle de régression, nous suivons les étapes ci-dessous :

- Step 1 : Chargement des bibliothèques et des modules requis
- Step 2 : Chargement et vérification de base des données
- Step 3 : Définir les variables explicatives (les paramètres contextuels dans notre cas) qu'on note X et les variables cibles à prédire (les paramètres de contrôle) qu'on note Y
- Step 4 : Division des données en données d'entraînement et données de test en utilisant le module *train_test_split* de *sklearn*.
- Step 5 : Définir, compiler et ajuster le modèle de régression
- Step 6 : Prédiction sur les données de test et calcul des métriques d'évaluation.

Pour mettre en place le réseau neurones pour la régression, nous utilisons la bibliothèque open-source en *Python* dédiée au réseau de neurones nommée *Keras* [Cho+15]. Cette bibliothèque fournit une *API* de haut niveau pour faciliter l'utilisation et simplifier la création de modèles de réseaux de neurones. Pour mettre en place le modèle, nous adoptons le modèle séquentiel qui est le type de modèle le plus simple. Ce modèle est constitué d'un empilement linéaire de couches. L'ajout d'une couche est effectué moyennant la méthode *.add()*. La première couche du modèle séquentiel doit recevoir le nombre de dimensions d'entrée qui est égal au nombre de paramètres contextuels dans notre cas. On spécifie ce dernier dans la première couche à travers le paramètre *input_dim*. La couche de sortie dispose d'un nombre de nœuds égal au nombre de paramètres de contrôle P_{Ctrl} à prédire (i.e., le nombre de paramètres de contrôle à prédire est fixé dans la dernière couche). Le nombre de couches cachées et le nombre de nœuds dans chaque couche cachée sont choisis via une expérimentation systématique car ils dépendent de la taille des données utilisées pour l'apprentissage du modèle. Concernant la fonction d'activation à utiliser, la fonction *ReLU* pour Unité de rectification linéaire et la fonction d'identité sont généralement utilisées pour les problèmes de régression. Nous utilisons la fonction *ReLU* comme fonction d'activation pour les couches cachées. Avant d'entraîner le modèle, on doit

configurer le processus d'apprentissage en appelant la méthode *compile()*. Cette méthode prend trois arguments en entrée :

- le paramètre *optimizer* : Les poids des connexions sont appris pendant la phase d'apprentissage à l'aide d'un algorithme d'optimisation. Les plus couramment utilisés sont *ADAM*, *SGD*. *ADAM* a été utilisé pour obtenir une convergence plus rapide.
- le paramètre *loss* : Il s'agit de la fonction de coût que le modèle va utiliser pour minimiser les erreurs. Nous utiliserons le *mean_squared_error (MSE)* comme fonction de perte.
- le paramètre *metrics* : Dans le cas d'un problème de régression, nous utilisons le *mean_squared_error (MSE)*

Le modèle est entraîné sur les données d'entraînement à travers l'appel de la fonction *fit()*. Le modèle est évalué ensuite sur les données de test (non connues) pour évaluer sa précision. La figure 3.8 illustre un exemple de mise en place d'un réseau de neurones pour la régression en utilisant la bibliothèque *Keras*.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=40)
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)

# Define model
model = Sequential()
model.add(Dense(500, input_dim=2, activation= "relu"))
model.add(Dense(100, activation= "relu"))
model.add(Dense(50, activation= "relu"))
#model.add(Dense(2))
model.add(Dense(2, activation='linear'))
#model.summary() #Print model Summary

model.compile(loss= "mean_squared_error" , optimizer="adam", metrics=["mean_squared_error"])
model.fit(X_train, y_train, epochs=20)
```

Figure 3.8: Exemple de réseau de neurones en utilisant *keras*

Chapitre 4

Évaluation

4.1 Introduction

Dans ce chapitre, nous proposons d'évaluer notre approche. Nous commençons par la comparer à deux méthodes classiques fréquemment utilisées par les automaticiens à savoir la méthode *Root Locus* (lieu des racines) et la méthode de *Ziegler-Nichols*. Nous appliquons ensuite notre approche au problème du *Leader-Follower* pour lequel nous ne disposons d'aucun modèle physique ou mathématique. Nous montrons également que notre approche ne se limite pas alors aux mesures de performance classiques et qu'il est possible de mettre en place des critères "externes".

4.2 Comparaison avec des méthodes classiques en automatique

Les méthodes traditionnelles de l'automatique exigent généralement une connaissance du modèle du système contrôlé que se soit sous forme de fonction de transfert ou sous forme d'équations dynamiques. Elles conduisent à des réponses indicielles différentes pour un même problème. Nous choisissons de caractériser ces réponses en termes de temps de montée *Rise Time*, temps d'établissement du régime permanent (*Settling Time*) et de dépassement (*Overshoot*), pour définir a insi u n c ritère d 'évaluation p our n otre m éthode. Par la suite, nous perturbons le système pour montrer que notre approche constitue une alternative intéressante, dans la mesure où cette dernière permet d'automatiser la mise au point du contrôleur pour différentes valeurs de la perturbation ce qui serait fastidieux à faire avec la méthode de *Ziegler-Nichols* ou la méthode *Root-Locus*.

4.2.1 Système du second ordre et contrôleur proportionnel

Dans un premier temps, nous débutons avec un système non perturbé du second ordre caractérisé par la fonction de transfert suivante :

$$G_1(s) = \frac{Y(s)}{U(s)} = \frac{1}{s^2 + 20 \cdot s + 20}$$

On commence par enregistrer la réponse en boucle ouverte du système à un signal d'entrée en échelon (*Step* en anglais) (voir figure 4.1). Cette réponse est souvent appelée réponse indicielle.

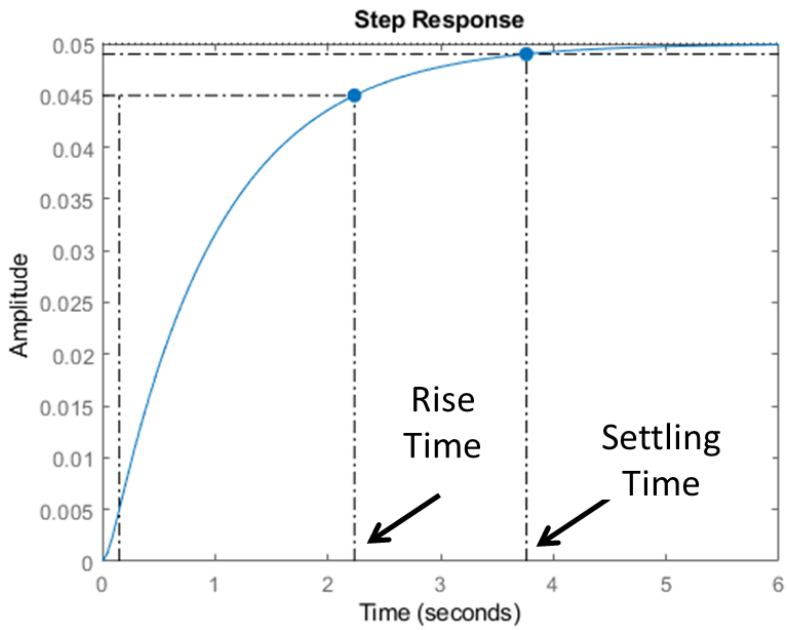

Figure 4.1: Réponse indicielle du système à régler

La valeur finale de la sortie à un signal d'entrée en échelon est de l'ordre de 0.05 ce qui correspond à une valeur d'erreur statique de 0.95. De plus, on constate que le *Rise Time* est d'environ deux secondes et que le *Settling Time* est de l'ordre de 3.7600 secondes. Comme le système ne satisfait pas les performances attendues, il est nécessaire de corriger son comportement en introduisant dans la boucle de commande un contrôleur. *Matlab Simulink* a été utilisé pour modéliser le système connu par sa fonction de transfert, un contrôleur proportionnel et une entrée de référence en échelon. La Figure 4.2 est le schéma bloc du système étudié.

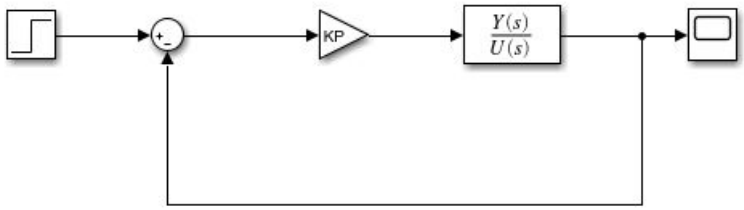

Figure 4.2: Schéma bloc du système étudié : un système du second ordre avec un contrôleur proportionnel

4.2.1.1 *Mise au point avec Ziegler-Nichols*

Premièrement, nous appliquons la méthode en boucle ouverte de *Ziegler-Nichols* pour choisir une valeur adéquate du gain K_P. Ceci a pour objectif de modifier le comportement du système en boucle fermée, soit améliorer les performances du système en termes de stabilité, précision et rapidité. Comme la courbe de la réponse indicielle du système en absence de régulateur (en boucle ouverte) est en forme de 'S', on trace la tangente au point d'inflexion (en rouge dans la figure 4.3).

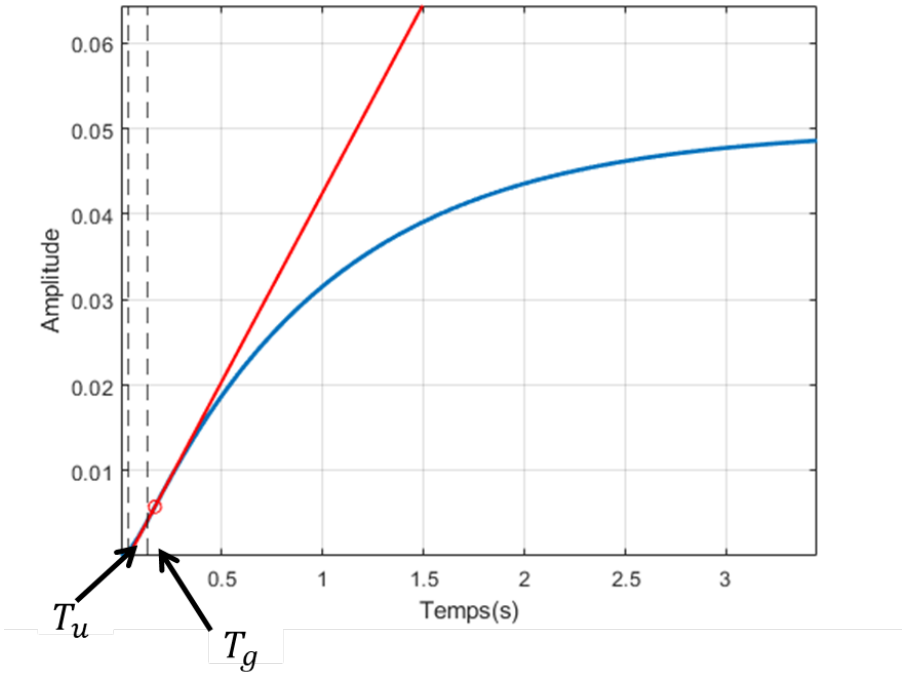

Figure 4.3: Méthode de *Ziegler-Nichols* en boucle ouverte

On en déduit (à partir de la courbe) que le temps de retard noté T_u est de l'ordre de 0.0375 secondes et que le temps de montée de la tangente noté T_g

est égal à 0.130675. En se référant au tableau 2.1 (de *Ziegler-Nichols*), on tire la valeur adéquate du gain proportionnel $K_P = {}^{0.130675}\!/_{0.05 \cdot 0.0375} = 603.19$. La figure 4.4 illustre la réponse du système pour cette valeur de K_P. Le graphique montre que le contrôleur proportionnel a réduit à la fois le temps de montée (*Rise Time*), le temps de stabilisation *Settling Time* et l'erreur en régime permanent. Mais d'un autre côté, il augmente le dépassement (*Overshoot*). Les performances du paramétrage proposé par *Ziegler-Nichols* sont calculées moyennant la fonction *stepinfo* de *Matlab* et sont : *Settling Time* = 0.3369, *Overshoot* = 25.2681% et *Rise Time* = 0.0590.

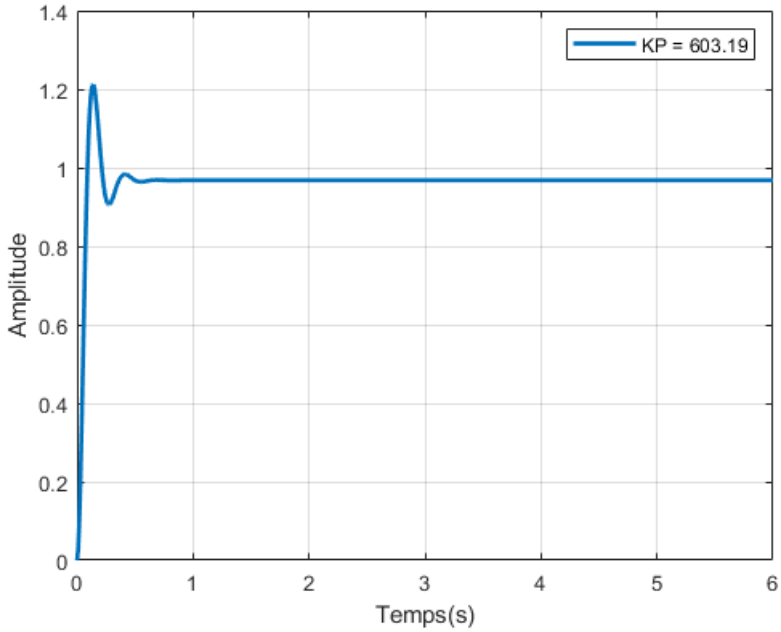

Figure 4.4: Mise au point en utilisant *Ziegler-Nichols*

Nous appliquons ensuite notre approche dans le but de trouver une valeur adéquate du gain pour le contrôleur proportionnel permettant d'atteindre les mêmes performances que celles proposées par *Ziegler-Nichols*. Pour cette expérimentation, un seul paramètre de contrôle P_{Ctrl} noté K_P est considéré sans paramètre contextuel P_{Ctxt}. L'intervalle choisi pour le gain K_P est [50,800] avec un pas de 5. Ainsi 151 configurations sont simulées via *Matlab S imulink*. Nous choisissons d'évaluer la performance du contrôleur proportionnel en termes de *Settling Time* et du pourcentage d'*Overshoot*. Nous associons chaque configuration testée à une des deux classes de *QoC good* ou *bad*. Une configuration est associée à la classe *good* si et seulement si le *Settling Time* varie entre 0.32 et 0.34 et le pourcentage d'*Overshoot* varie entre 25% et 26%. Le résultat du binning donne alors : 6 configurations appartiennent à la classe *good* tandis que le reste, i.e., 145 configurations appartiennent à la classe *b ad*. En absence de paramètres contextuelles P_{Ctxt} pour cette expérimentation, nous avons ignoré les étapes de

Feature Selection et de *clustering*. Les résultats obtenus montrent que pour une valeur du gain K_P comprise entre 585 à 610, le contrôleur proportionnel répond aux exigences fixées. Notre approche donne un gain K_P égal à 600 ce qui correspond au point milieu de cet intervalle. Il est intéressant de noter que le paramétrage trouvé par la méthode de *Ziegler-Nichols* est à l'intérieur de cet intervalle. Nous avons ensuite effectué une simulation en utilisant la valeur proposée (600) qui a aboutit à un *Settling Time* de 0.3378 secondes et un *Overshoot* de l'ordre de 25.1779%. Le tableau 4.1 résume les résultats de la comparaison entre notre méthode et celle de *Ziegler-Nichols*.

	Ziegler-Nichols	Notre approche
K_P	603.19	600
Settling Time	0.3369	0.3378
Overshoot	25.2681	25.1779
Rise Time	0.0590	0.0591

Table 4.1: Comparaison avec la méthode de *Ziegler-Nichols*

4.2.1.2 Mise au point avec *Root Locus*

Contrairement à la méthode de *Ziegler-Nichols*, la méthode *Root Locus* offre la possibilité d'imposer des critères de performance (en termes de *Settling Time* et/ou d'*Overshoot*) pour guider la mise au point (la recherche des valeurs adéquates pour le gain K_P) du contrôleur. Il est important de préciser qu'il n'a pas de correspondance entre le *Rise Time* et l'emplacement des pôles. Dans ce qui suit, nous utilisons la technique *Root Locus* pour concevoir un contrôleur proportionnel simple (trouver une valeur adéquate du gain proportionnel K_P) qui répond à certaines spécifications de performance. Notre objectif consiste à trouver des valeurs pour le gain K_P qui garantissent un pourcentage d'*Overshoot* inférieur à 26% et un *Settling Time* T_r ne dépassant pas 0.48 secondes. Ces exigences se traduisent par certaines restrictions sur l'emplacement des pôles. La fonction *Matlab rlocus* est utilisée pour dessiner les pôles de la fonction de transfert du système. La figure 4.5 illustre tous les emplacements possibles des pôles en boucle fermée pour le contrôleur proportionnel utilisé.

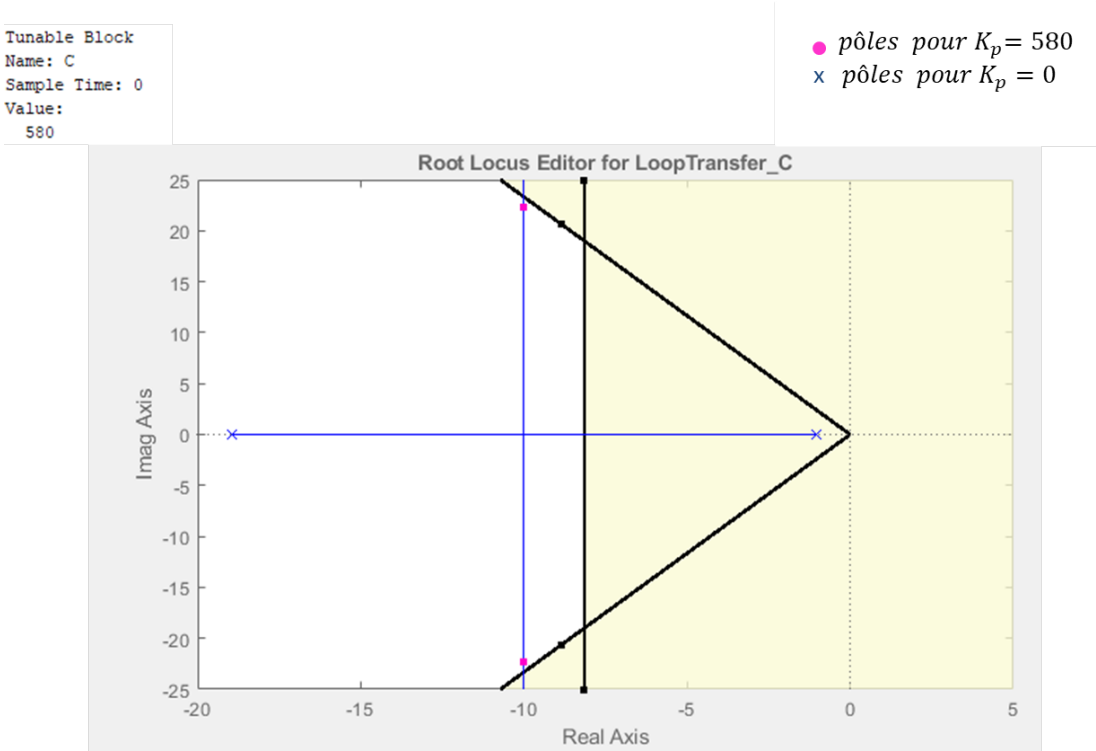

Figure 4.5: *Matlab Simulink* Root Locus fonction

Un pourcentage d'*Overshoot* inférieur à 26% impose que les pôles soient placés à gauche des deux lignes qui partent de l'origine (avec un angle θ_{req} = 62 °). Un *Settling Time* T_r qui ne dépasse pas 0.4 secondes se traduit par une restriction sur la partie réelle des pôles : tout ce qui se trouve à gauche de la ligne verticale répondra à cette exigence. Par conséquent, toutes les valeurs du gain K_P qui déplacent les pôles sur le segment vertical en bleu du plan complexe permettent de répondre aux exigences de performance fixées. Il convient de noter que la valeur trouvée par la méthode de *Ziegler-Nichols* fait partie des valeurs possibles trouvées par *Root Locus*.
En particulier, un K_P de 580 convient (voir figure 4 .5). Cette valeur proposée par la technique *Root Locus* est testée par simulation *Matlab Simulink* et donne les résultats suivants : 0.3433 secondes en termes de *Settling Time* et 24.5358 % en termes d'*Overshoot*. La figure 4.6 illustre les performances et la réactivité du contrôleur avec le gain trouvé.

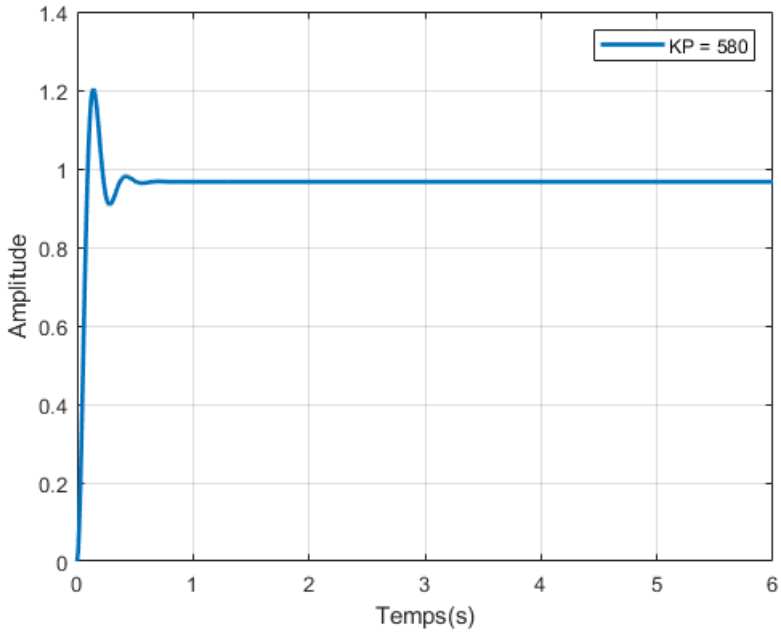

Figure 4.6: Mise au point en utilisant Root Locus

Nous appliquons ensuite notre approche dans le but de trouver une valeur adéquate de K_P. Comme décrit précédemment, nous avons choisi de varier le gain K_P entre 50 et 800 avec un pas de 5. On associe chaque configuration testée à une des deux classes de *QoC* notée *good* et *bad*. Pour cette expérimentation, une configuration est associée à la classe *good* si le *Settling Time* correspondant est compris entre 0.32 et 0.36 et le pourcentage d'*Overshoot* varie entre 24% et 25.7%. En dehors de ces limites (ranges), la configuration est associée à la classe *bad*. Le résultat du binning donne alors : 8 configurations appartiennent à la classe *good* tandis que 143 configurations appartiennent à la classe *bad*. Les résultats obtenus révèlent que pour une valeur du gain K_P comprise entre 565 et 600, le contrôleur proportionnel répond aux exigences fixées. Notre approche choisit un gain K_P égal à 582.5. Nous avons ensuite effectué une simulation avec *Matlab Simulink* en utilisant cette valeur. Le tableau 4.2 résume les résultats de la comparaison entre notre méthode et la méthode *Root locus*.

	Root Locus	Notre approche
K_P	580	582.5
Settling Time	0.3431	0.3426
Overshoot	24.5535	24.6235
Rise Time	0.0603	0.0602

Table 4.2: Comparaison avec la méthode *Root Locus*

Dans ce cas simple, avec notre approche, on est capable de trouver des valeurs de gain qui aboutissent à des performances similaires à celles trouvées

par *Ziegler-Nichols* et *Root Locus*.

4.2.1.3 Mise au point avec notre Approche

Nous appliquons ensuite notre approche dans le but de trouver une valeur du gain K_P permettant d'atteindre un niveau de performance souhaité en termes de *Settling Time*, d'*Overshoot* mais aussi de *Rise time*. L'objectif étant de réduire le pourcentage d'*Overshoot* et le *Settling Time*, et en ajoutant le *Rise Time* qui n'est pas traité par les deux méthodes précédentes. Pour cette expérimentation, on garde le même range choisi pour le gain K_P qui est compris entre 50 et 800 avec un pas de 5. Nous associons chaque configuration testée à une des deux classes de *QoC* notées *good* et *bad*. Une configuration est associée à la classe *good* si et seulement si le *Settling Time* correspondant varie entre 0.3 et 0.32, le pourcentage d'*Overshoot* est compris entre 10 et 13 et le *Rise Time* correspondant ne dépasse pas 0.1 secondes. Au-delà de ces ranges, la configuration est associée à la classe *bad*. Les résultats obtenus montrent qu'une valeur du gain K_P compris entre 310 et 320 permet au contrôleur de répondre aux exigences fixées. Notre approche propose donc un gain K_P égale à 315 ce qui correspond au milieu de cet intervalle . Nous avons ensuite effectué une simulation en utilisant la valeur proposée et cela a conduit à un *Settling Time* de 0.3181 secondes, un *Overshoot* de l'ordre de 12.8284% et un *Rise Time* de 0.0950 secondes (voir figure 4.7). On peut remarquer que ce réglage obtenu par notre méthode conduit à un *Overshoot* moins important.

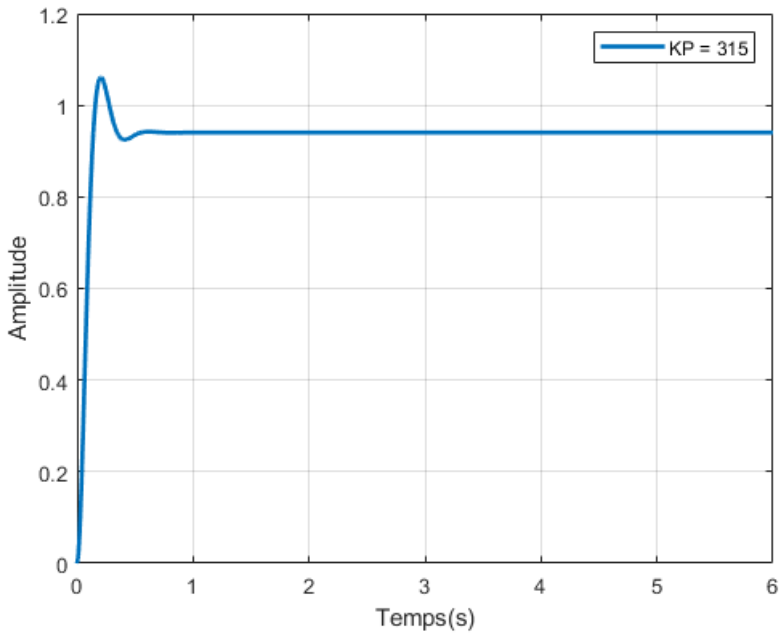

Figure 4.7: Mise au point en utilisant notre approche

Pour conclure, en s'appuyant sur la puissance de calcul, et en permettant un

balayage, notre approche permet d'obtenir des paramétrages qui répondent à des exigences fixées en termes de *QoC* contrairement à *Ziegler-Nichols* et à *Root-Locus* qui impose des critères de performance en termes d'*Overshoot* et de *Settling Time* seulement.

4.2.2 Système d'ordre 2 et contrôleur proportionnel avec perturbation

Dans cette partie, nous étudions un système de second ordre caractérisé par la fonction de transfert ci-dessous :

$$G_1(s) = \frac{Y(s)}{U(s)} = \frac{s+2}{s^2 + 0.01 \cdot s + 0.0025}$$

Le système étudié est soumis à une entrée en échelon. Un contrôleur proportionnel est utilisé pour réguler ce dernier. On perturbe le système avec un échelon de valeur qui varie entre 0 et 2 avec un pas de 0.1. *Matlab Simulink* a été utilisé pour modéliser le système à travers sa fonction de transfert, une entrée en échelon, un contrôleur proportionnel dont le gain est noté K_P et une perturbation en échelon. Le schéma bloc du système défini est illustré dans la figure 4.8.

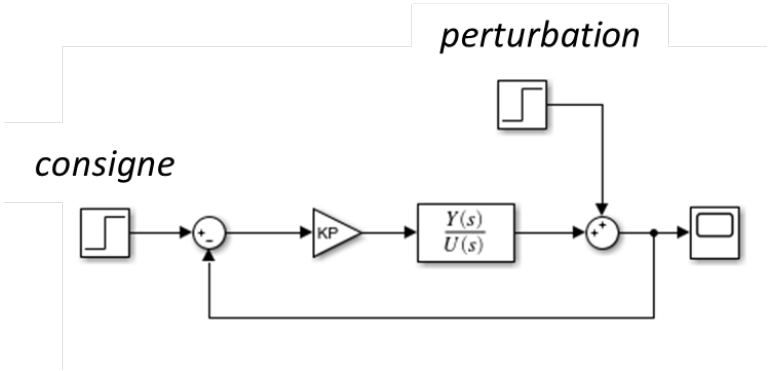

Figure 4.8: Schéma bloc du système étudié avec perturbation

4.2.2.1 Mise au point avec *Ziegler-Nichols*

Nous commençons par l'application de la méthode *Ziegler-Nichols* pour trouver une valeur adéquate du gain proportionnel K_P en absence de perturbation. Comme la réponse du système seul sans régulateur n'est pas en forme de *'S'* (voir figure 4.9), il n'est donc pas possible d'utiliser la méthode en boucle ouverte de *Ziegler-Nichols*. Par la suite, nous avons essayé d'augmenter le gain K_P dans le but d'emmener le système bouclé à sa limite de stabilité. Néanmoins, toutes ces tentatives ont échoué. Ainsi, la méthode du point critique de *Ziegler-Nichols* n'est pas applicable dans ce cas.

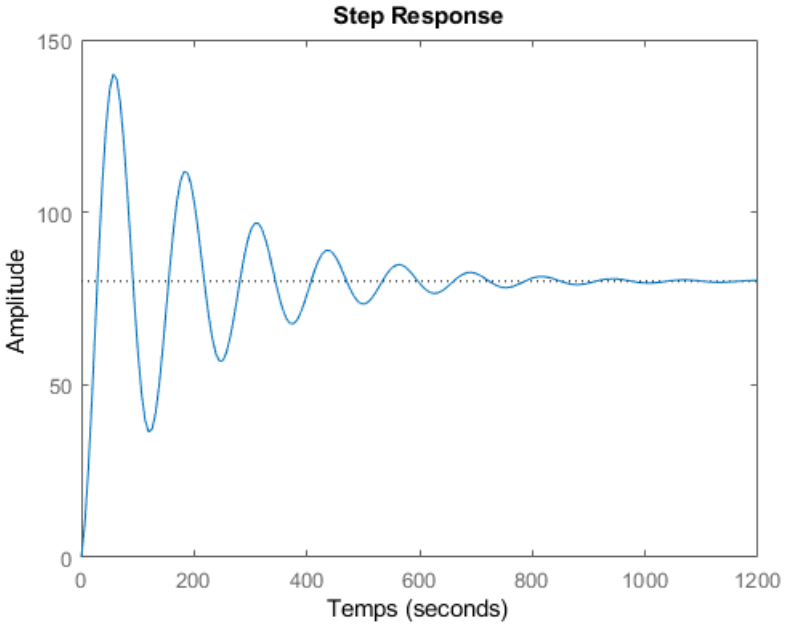

Figure 4.9: Réponse du système en boucle ouverte

4.2.2.2 Mise au point avec *Root Locus*

Ensuite, nous utilisons la technique *Root Locus* pour trouver des valeurs adéquates pour le gain K_P permettant de répondre à certaines spécifications de performance en absence de perturbation. L'objectif consiste à trouver des valeurs pour le gain K_P qui garantissent un *Overshoot* inférieur à 20% et un *Settling Time* qui ne dépasse pas 13 secondes. Ces exigences se traduisent par certaines restrictions sur l'emplacement des pôles. Nous utilisons la fonction *rlocus* de *Matlab* pour dessiner les pôles. La figure 4.10 illustre tous les emplacements possibles des pôles en boucle fermée pour le contrôleur proportionnel utilisé (le cercle).
Tout ce qui se trouve à gauche de la ligne verticale (en rouge) répond à l'exigence sur le *Settling Time*. Pour répondre à la contrainte fixée sur l'*Overshoot*, les pôles doivent être placés à gauche des deux lignes qui partent de l'origine (lignes bleues). Ainsi, toutes les valeurs du gain K_P sur le cercle qui déplacent les pôles à gauche des droites en rouge et en bleu du plan complexe permettent de répondre aux exigences de performances fixées. En particulier un gain K_P égal à 0.74 convient. Ensuite, nous avons effectué une simulation *Matlab Simulink* en utilisant cette valeur. Les résultats obtenus sont : un *Settling Time* égal à 12.8145 secondes et un *Overshoot* de l'ordre de 12.39%. Nous avons par la suite évalué le paramétrage trouvé par *Root Locus* en présence de la perturbation. L'application de la perturbation engendre une augmentation de l'*Overshoot* qui dépasse la barre des 30% à partir d'une perturbation égale à 0.5.

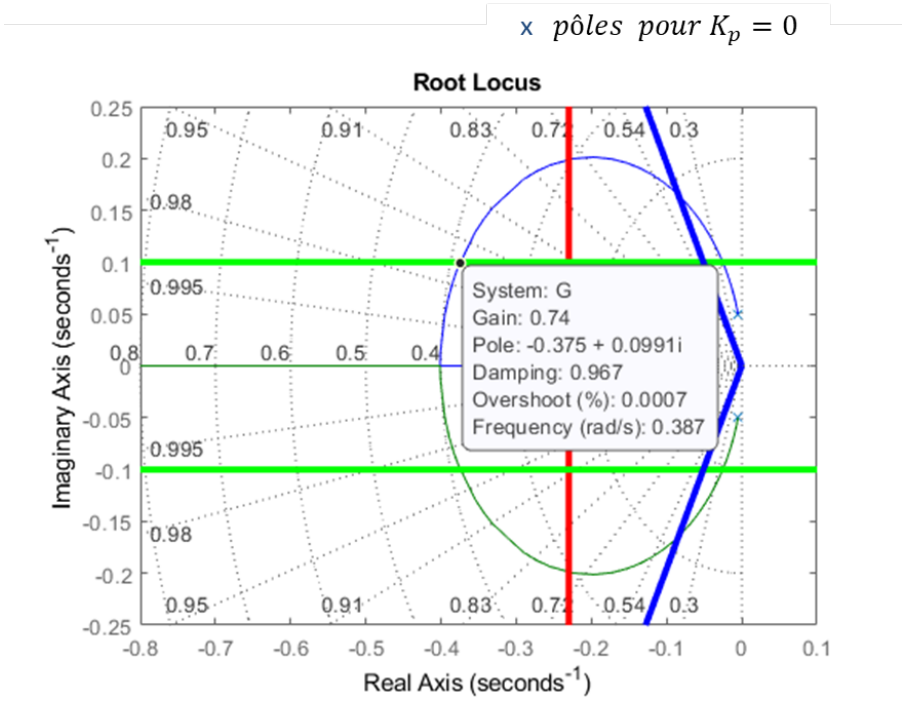

Figure 4.10: Réponse du système en boucle ouverte

4.2.2.3 Mise au point avec notre approche

Nous appliquons notre approche pour trouver automatiquement une valeur du gain K_P valable pour plusieurs contextes. Dans cette expérience, un paramètre contextuel P_{Ctxt} noté *Disturbance* et un paramètre de contrôle P_{Ctrl} noté K_P sont considérés. Les intervalles fixés pour ces paramètres sont présentés dans le tableau 4.3. A noter qu'une perturbation *Disturbance* égale à 0 est l'équivalent d'un système non perturbé.

Paramètre	*range*	*Step*
Disturbance	[0 , 2]	0.1
K_P	[0.1 , 20]	0.1

Table 4.3: Plage de paramètres pour l'étape d'exploration

Dans cette expérimentation, 21 contextes différents sont possibles et 4200 configurations sont simulées moyennant *Matlab Simulink*. Pour l'étape de *binning*, on associe chaque configuration testée à une des deux classes de *QoC* (*good*et *bad)*. Dans un premier temps, on attribue une configuration à la classe *good* si le *Settling Time* correspondant ne dépasse pas 13 secondes et l'*Overshoot* correspondant est inférieur à 20%. Au-delà de ces intervalles, la configuration est associée à la classe *bad*. On obtient ainsi : 397 configurations appartenant à la classe *good* et 3803 configurations appartenant à la classe *bad*. On constate que pour une valeur de perturbation *Disturbance* supérieur à 0.5, nous n'avons

pas trouvé un paramétrage appartenant à la classe *good*. Une fois ceci fait, nous appliquons l'algorithme de clustering proposé pour différentes valeurs de *percentage_of_confidence*. les résultats obtenus pour la classe *good* sont présentés au tableau 4.4 ci-dessous :

percentage_of_confidence	*Nbr de modes de fonctionnement*	*temps nécessaire*
0.2	2	1.01185 s
0.4	3	0.9831 s
0.6	4	0.9758 s
0.8	5	0.9799 s

Table 4.4: Nombre de modes de fonctionnement pour la classe *good*

Nous obtenons un nombre minimal de modes de fonctionnement en fixant le *percentage_of_confidence* à 0.2. Nous avons réussi à regrouper les contextes caractérisés par une perturbation égale à 0.0 et 0.1 d'une part et les contextes dont les valeurs de la perturbation sont 0.2, 0.3 et 0.4. On obtient donc deux modes de fonctionnements. Le paramétrage choisi pour le premier mode de fonctionnement (*Disturbance = 0.0 ou 0.1*) est K_P = 0.6. Pour le deuxième mode, K_P est égal à 0.6. Nous avons ensuite effectué une simulation en utilisant le paramétrage trouvé pour une perturbation égale à 0.3. La réponse du système en boucle fermée avec un K_P égal à 0.6 est illustrée dans la figure 4.11. Les performances en termes de *QoC* sont comme suit : un *Settling Time* égal à 12.1103 secondes et un *Overshoot* de l'ordre de 14.4553 %.

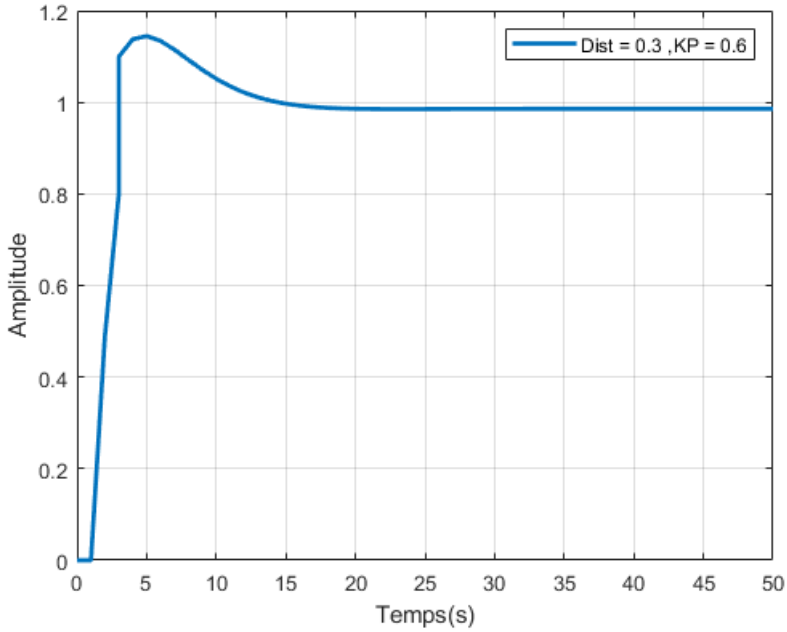

Figure 4.11: Réponse du système en présence d'une perturbation égale à 0.3

Ainsi, notre approche permet non seulement de trouver un paramétrage qui

répond à des exigences en termes de *QoC* mais aussi un paramétrage valable pour différents contextes.

4.2.3 Système d'ordre 3 et contrôleur *PID*

Dans cette section, la performance d'un contrôleur *PID* qui a été mis au point moyennant notre approche est comparée avec celle obtenue par l'approche *Ziegler-Nichols*. *Matlab Simulink* a été utilisé pour modéliser un système d'ordre trois connu par sa fonction de transfert, un contrôleur *PID* et une entrée de référence en échelon (voir figure 4.12).

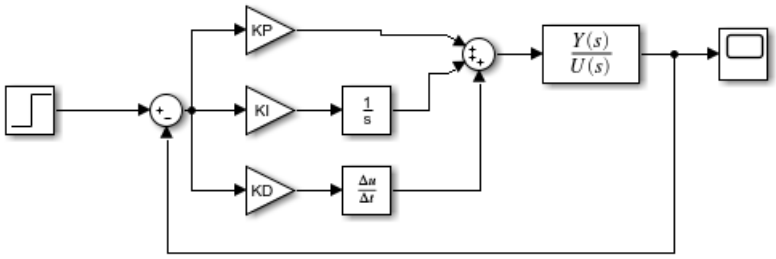

Figure 4.12: Schéma bloc du système étudié : un système avec un contrôleur PID

Le système contrôlé utilisé pour évaluer les performances des différentes approches dispose de la fonction de transfert suivante :

$$G_2(s) = \frac{Y(s)}{U(s)} = \frac{1}{s^3 + 6 \cdot s^2 + 11 \cdot s + 6}$$

4.2.3.1 Mise au point avec la méthode de Ziegler-Nichols

Dans un premier temps, nous avons commencé par appliquer la méthode de *Ziegler-Nichols* en boucle fermée (la méthode du point critique). En partant du modèle du système en boucle fermée avec un contrôleur proportionnel (K_I et K_D sont fixés à zéro), on commence à augmenter le gain proportionnel K_P jusqu'à amener le système à osciller de manière permanente (voir figure 4 .13). Le gain critique trouvé et la période d'oscillation correspondante sont : K_{cr}=60 et T_{cr}=1.9. En se référant au tableau 2.2, le paramétrage obtenu par la méthode *Ziegler-Nichols* est comme suit : K_P=36, K_I=37.89 et K_D=8.55.

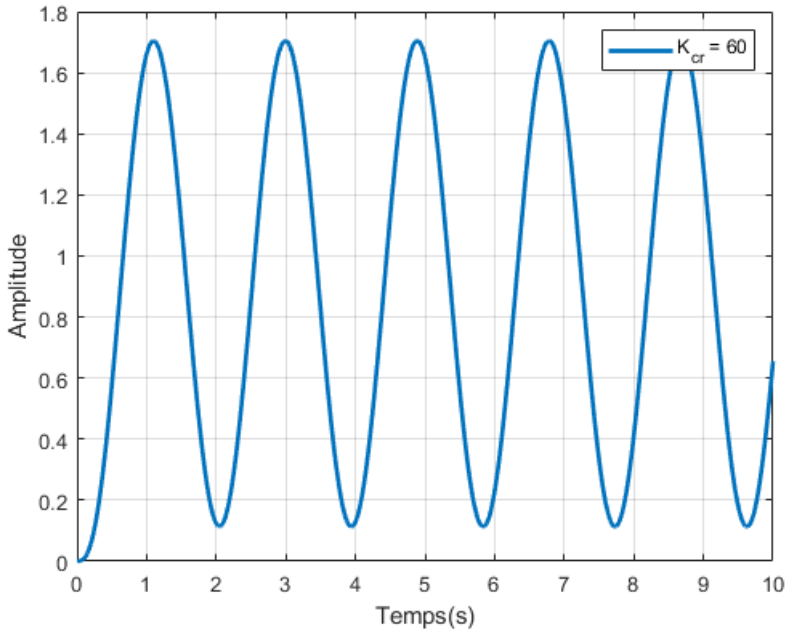

Figure 4.13: Méthode du point critique : mesure de la période d'oscillation T_{cr}

Comme le montre la figure 4.14, ce paramétrage conduit aux performances suivantes : un *Settling Time* de 5.6926 secondes, un *Overshoot* de 49.96% et un *Rise Time* de 0.5267 secondes.

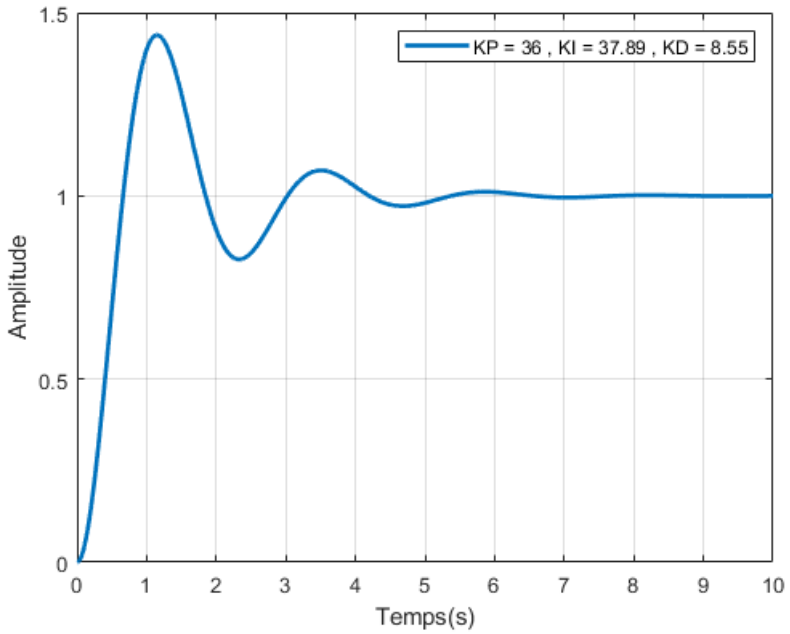

Figure 4.14: Mise au point en utilisant *Ziegler-Nichols*

On peut remarquer que comme souvent le réglage par cette méthode conduit à un overshoot très important.

4.2.3.2 Mise au point avec notre approche

Nous avons ensuite appliqué notre approche tout en fixant le même niveau de performance (avec des intervalles proches pour les critères de performance donnés par *Ziegler-Nichols*). Dans cette expérience, trois paramètres de contrôle $\{P_{Ctrl}\}$ communément appelés K_P, K_I et K_D sont considérés. Les intervalles pour les différents paramètres K_P, K_I et K_D sont illustrés dans le tableau 4.5.

Paramètre	*range*	*Step*
K_P	[1 , 50]	3
K_I	[1 , 50]	3
K_D	[1 , 50]	3

Table 4.5: Plage des paramètres pour l'étape d'exploration

Dans cette expérimentation 4913 ($17 \times 17 \times 17$) configurations sont simulées. La simulation a été effectuée via *Matlab Simulink*. En absence de paramètres contextuels pour cette expérimentation nous avons ignoré les étapes de *Feature Selection* et de *Clustering*.
La performance des configurations testées est évaluée en termes de *Settling Time*, de *Rise Time* et d'*Overshoot*. On associe à chaque configuration testée une des deux classes de *QoC* notée *good* et *bad*. Une configuration est attribuée à la classe *good* si et seulement si le *Settling Time* correspondant est inférieur à 7 secondes, le *Rise Time* correspondant est compris entre 0 et 2 secondes et un *Overshoot* qui varie entre 25% et 55%. Au-delà de ces limites, la configuration est considérée comme *bad*. Parmi les 4913 configurations testées, 308 configurations sont affectées à la classe *good*, tandis que le reste i.e., 4605 configurations appartiennent à la classe *bad*. La figure 4.15 est un diagramme de dispersion qui représente l'ensemble des paramètres de contrôle $\{P_{Ctrl}\}$ K_P , K_I et K_D appartenant à la classe de *QoC good* (points en bleu) trouvé avec notre approche en absence de perturbation. Le paramétrage proposé par *Ziegler-Nichols* est représenté par une étoile rouge tandis et on constate que ce dernier est proche de la frontière. On peut alors supposer que ce paramétrage ne soit pas acceptable en présence de perturbation. Le paramétrage proposé par notre approche est modélisé par une étoile en vert. Il s'agit du point du milieu (le plus proche du barycentre). Notre approche propose un gain K_P égal à 31, un gain K_I égal à 37 et un gain K_D égal à 16.

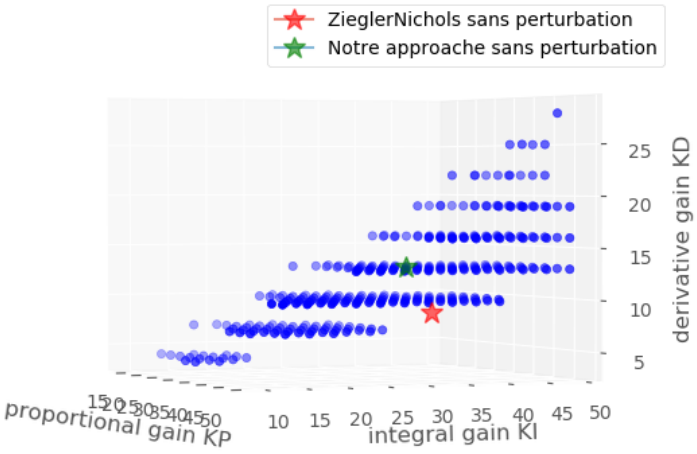

Figure 4.15: Diagramme de dispersion

Nous avons ensuite effectué une simulation en utilisant le paramétrage proposé par notre approche. La figure 4.16 illustre la performance du contrôleur avec ces valeurs trouvées.

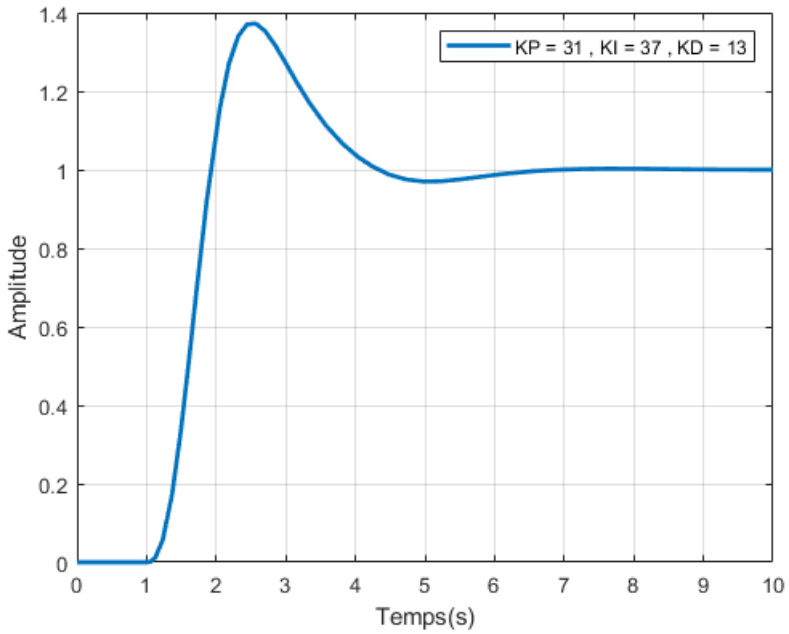

Figure 4.16: Mise au point en utilisant notre approche

Les performances obtenues sont représentées dans le tableau 4.6.

	Ziegler-Nichols	*Notre approche*
K_P	36	31
K_I	37.89	37
K_D	8.55	13
Settling Time	5.69	5.7035
Overshoot	49.96	37.2428
Rise Time	0.52	0.5693

Table 4.6: Notre approche vs *Ziegler-Nichols*

Les valeurs de paramètres de contrôle proposées par notre approche donnent comme prévu des performances similaires au paramétrage proposé par *Ziegler-Nichols* en termes de *Settling Time* et *Rise Time* et meilleures en terme d'*Overshoot*.

4.2.3.3 Perturbation du système

Dans ce qui suit, on perturbe le système et nous utilisons notre approche pour trouver automatiquement des valeurs adéquates des gains (K_P, K_I , K_D). Cette estimation serait fastidieuse à faire avec les méthodes *Ziegler-Nichols* ou *Root Locus*. La perturbation appliquée ici est un échelon d'une valeur qui varie entre 0.0 et 1 (la perturbation est du même ordre de grandeur que le signal lui-même). Elle est appliquée à un temps t égal à 3 secondes (i.e., à t = 3s, la perturbation passe de la valeur initiale fixée à 0 à la valeur finale). *Matlab Simulink* a été utilisé pour modéliser le système, le contrôleur PID, une entrée de référence en échelon et une perturbation sous la forme d'un signal en échelon (voir figure 4.17).

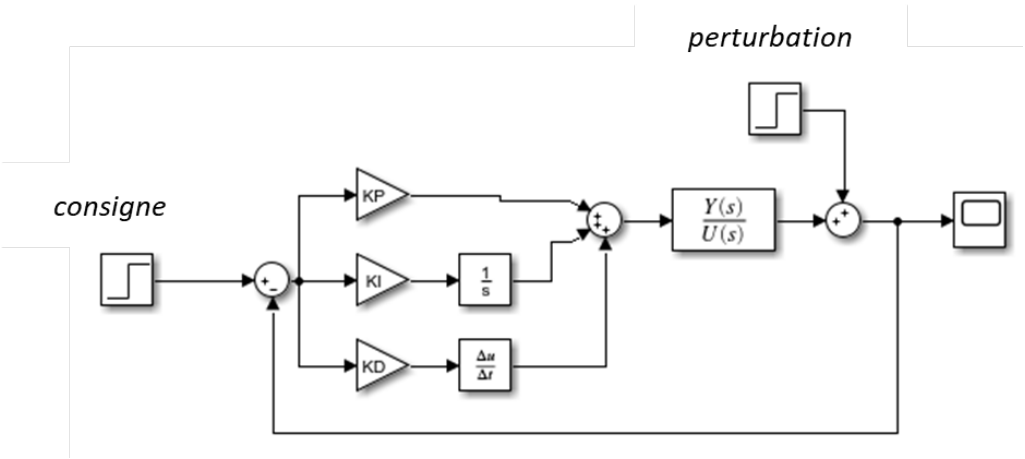

Figure 4.17: Schéma bloc du système avec perturbation

Nous commençons par évaluer les performances du paramétrage trouvé par la méthode *Ziegler-Nichols* en présence de perturbation (cf. 4.2.3.1). Le tableau

4.7 résume les résultats de cette évaluation.

Disturbance	*Settling Time*	*Overshoot*	*Rise Time*
0.1	5.814003	54.334845	0.490779
0.2	5.147701	54.349471	0.490746
0.3	5.400668	54.339454	0.490769
0.4	6.484069	54.334271	0.490781
0.5	6.687708	58.710602	0.490799
0.6	6.789548	68.736822	0.490742
0.7	6.873312	78.705809	0.490806
0.8	6.923139	88.708146	0.490800
0.9	7.728507	98.733953	0.490753
1.0	7.728742	108.740386	0.490743

Table 4.7: Mise au point avec *Ziegler-Nichols* en présence d'une perturbation

En appliquant la perturbation, le paramétrage trouvé moyennant *Ziegler-Nichols* n'est plus correct en termes de qualité de contrôle *QoC*. En fait plus la valeur de la perturbation augmente plus on constate un dépassement (*Overshoot*). A titre indicatif, pour une perturbation qui vaut 0.6, le pourcentage d'*Overshoot* dépasse 60%. Ainsi, ce paramétrage ne permet pas de répondre à l'objectif fixé pour le contrôleur décrit par les contraintes en termes d'*Overshoot* et de *Settling Time* (pour une perturbation qui dépasse 0.9 le *Settling Time* est supérieur à 7).
Nous appliquons ensuite notre approche dans le but de trouver des bons paramètres de contrôle valables pour divers contextes. Dans cette expérience, un paramètre contextuel P_{Ctxt} noté *Disturbance* et trois paramètres de contrôle K_P, K_I et K_D sont considérés. Il est à noter que nous avons ignoré l'étape de *Feature Selection* (puisque un seul paramètre contextuel est considéré). Les intervalles pour les différents paramètres sont illustrés dans le tableau 4.8.

Paramètre	*range*	*Step*
Disturbance	[0 , 1]	0.1
K_P	[1 , 50]	3
K_I	[1 , 50]	3
K_D	[1 , 50]	3

Table 4.8: Plage des paramètres pour l'étape d'exploration

L'espace de recherche englobe l'ensemble des valeurs à virgule flottante dans les plages spécifiées dans le tableau p récédent. En ce qui concerne la perturbation, une valeur de *Disturbance* égale à zéro équivaut à un système non perturbé. Dans cette expérimentation, 11 contextes différents sont possibles et 54043 configurations ont été simulées à l'aide de *Matlab Simulink*.
Nous évaluons les configurations testées en termes de *Settling Time*, d'*Overshoot* et de *Rise Time*. On associe à chaque configuration testée une des deux classes de *QoC good* et *bad*. Dans un premier temps, nous fixons les mêmes intervalles pour

l'étape de *binning*, c'est-à-dire qu'une configuration est associée à la classe *good* si le *Settling Time* ne dépasse pas 7 secondes, le *Rise Time* correspondant varie entre 0 et 2 secondes et l'*Overshoot* est compris entre 25% et 55%. Les résultats du *binning* donne : parmi les 54043 configurations testées : 3809 configurations appartiennent à la classe *good* et 50234 appartiennent à la classe *bad*.
Nous commençons d'abord par diviser les configurations en deux-sous ensembles, une par classe de *QoC* (*good, bad*). Par la suite, pour chaque contexte, un ensemble de paramètres de contrôle composé de triplets (K_P, K_I, K_D) par classe de *QoC* est déterminé. Une fois ceci fait, nous inspectons le chevauchement des ensembles (*clustering*). Le tableau 4.9 représente le nombre de clusters (i.e., le nombre de modes de fonctionnement) et le temps pour les calculer pour différentes valeurs de confiance *percentage_of_confidence*.

percentage_of_confidence	*Nbr de mode de fonctionnement*	*temps nécessaire*
0.2	4	4.5890 s
0.4	5	4.4168 s
0.6	6	4.4210 s
0.8	8	4.4419 s

Table 4.9: Nombre de modes de fonctionnement pour la classe *good*

Dans le but d'avoir un nombre minimal de modes de fonctionnement, nous devons régler (fixer) le *percentage_of_confidence* à 0.2. Dans ce cas, le triplet (K_P = 46, K_I = 25, K_D = 10) peut être choisi pour le premier mode de fonctionnement associé aux 7 contextes caractérisés par une perturbation *Disturbance* comprise entre 0 et 0.6. Le paramétrage suivant (K_P = 37, K_I = 1, K_D = 43) est choisi pour le deuxième mode de fonctionnement caractérisé par une perturbation égale à 0.7. Pour une perturbation *Disturbance* égale à 0.8, le paramétrage trouvé est comme suit : (K_P = 22, K_I = 1, K_D = 22). Tandis que le triplet (K_P = 13, K_I = 1, K_D = 13) est associé à une perturbation égale à 0.9 et caractérise le quatrième mode de fonctionnement. Finalement, pour une perturbation *Disturbance* égale à 1, aucun paramétrage de contrôle vérifiant les critères de *QoC* fixés n'a été trouvé. A l'issue de ce processus de clustering, nous avons obtenu quatre modes de fonctionnement. À titre d'exemple, nous avons ensuite effectué une simulation en utilisant le paramétrage trouvé pour une perturbation (*Disturbance*) égale à 0.5 appliqué à t égal à 3 secondes (voir figure 4.18).

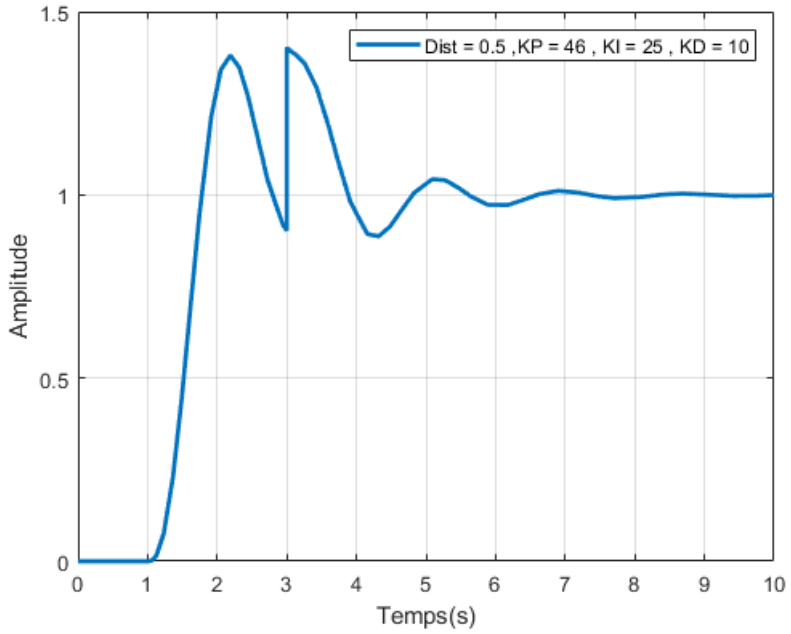

Figure 4.18: Réponse du système en présence d'une perturbation de 0.5

On constate que le système compense la présence de la perturbation extérieure en adaptant la valeur de la commande. Les valeurs de paramètres de contrôle proposées par notre approche engendre les performances suivantes : un *Settling Time* égal à 6.3017 secondes, un *Overshoot* de l'ordre de 40.3829% et un *Rise Time* égal à 0.4679 secondes.
En appliquant l'algorithme de clustering avec un *percentage_of_confidence* égal à 0.4, nous avons réussi à regrouper les contextes dont la valeur de perturbation *Disturbance* est compris entre 0 et 0.5. Pour une perturbation délimitée entre [0.6,0.9], chaque contexte (avec le paramétrage trouvé) est associé à un mode de fonctionnement séparé. De même, aucun paramétrage vérifiant les exigences fixées n'a été trouvé pour une perturbation égale à 1. Les résultats du clustering avec un *percentage_of_confidence* égal à 0.4 sont présentés dans le tableau 4.10 ci-dessous :

mode	*Nbr de contextes*	*Disturbance*	K_P	K_I	K_D
1	6	0.0 & 0.1 & 0.2 & 0.3 & 0.4 & 0.5	43	37	10
2	1	0.6	40	10	31
3	1	0.7	37	1	43
4	1	0.8	22	1	22
5	1	0.9	13	1	13

Table 4.10: Paramétrages trouvés pour 5 modes de fonctionnement

Pour illustrer l'utilité du paramétrage proposé, nous avons effectué une simulation pour une perturbation *Disturbance* égale à 0.6. La figure 4.19 représente

la réponse du système en boucle fermée. On constate que la grandeur perturbatrice entre en compte dans le système réglé à t égal 3 secondes provocant un pic à un temps t égal à 3.0938 secondes de l'ordre de 1.5396. Malgré la perturbation, le contrôleur est capable de suivre la consigne. Les performances du paramétrage proposé sont comme suit : un *Settling Time* égal à 6.5295 secondes, un *Rise Time* de l'ordre de 1.5433 secondes et un *Overshoot* égal à 53.9607 %.

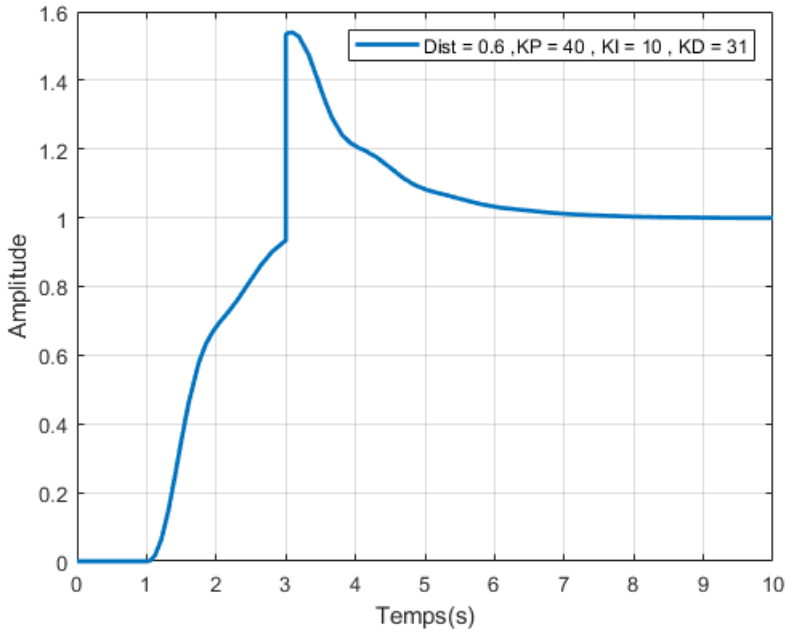

Figure 4.19: Réponse du système en présence d'une perturbation de 0.6

Si nous acceptons d'avoir plus de modes de fonctionnement, nous obtenons 8 modes de fonctionnement en fixant le *percentage_of_confidence* égal à 0.8. Les résultats obtenus sont présentés dans le tableau 4.11.

mode	*Nbr de contextes*	*Disturbance*	K_P	K_I	K_D
1	3	0.0 & 0.1 & 0.2	37	46	13
2	1	0.3	40	40	16
3	1	0.4	40	43	13
4	1	0.5	40	28	7
5	1	0.6	40	10	31
6	1	0.7	37	1	22
7	1	0.8	22	1	22
8	1	0.9	13	1	13

Table 4.11: Plage de paramètres pour l'étape d'exploration

A l'issue de l'étape de clustering avec un *percentage_of_confidence* égal à 0.8, nous avons réussi à regrouper les contextes dont la valeur de la perturbation

varie entre 0.0 et 0.2. De même, nous n'avons pas trouvé de paramétrage qui répond aux exigences en termes de *QoC* pour une perturbation égale à 1. Nous avons effectué une simulation avec *Matlab Simulink* en utilisant le paramétrage trouvé avec une perturbation égale à 0.9 (voir figure 4.20). Les performances en termes de *QoC* sont les suivantes : un *Settling Time* égal à 6.9452 secondes, un *Rise Time* égal à 1.5846 secondes et un *Overshoot* de l'ordre de 51.4361 %.

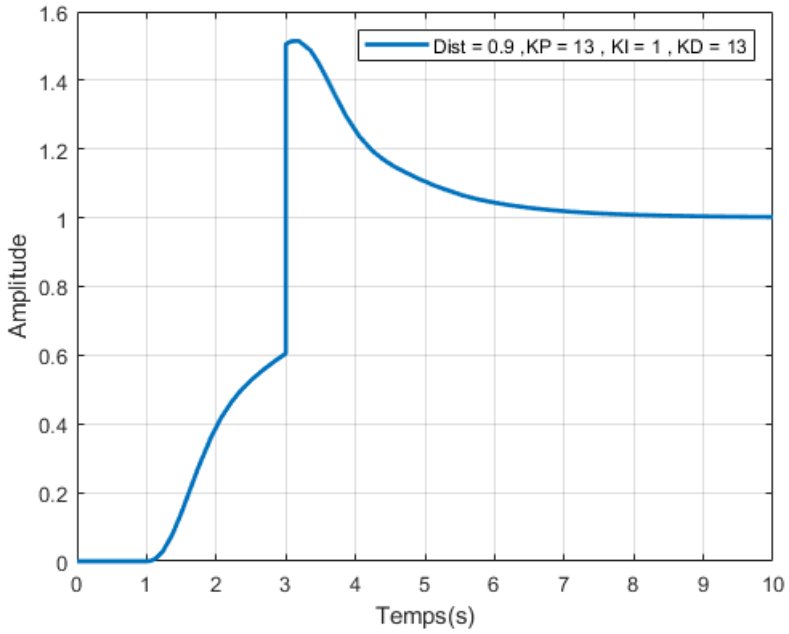

Figure 4.20: Réponse du système en présence d'une perturbation de 0.9

Nous avons ensuite modifié les intervalles lors de l'étape de *binning* dans le but de trouver des valeurs pour les paramètres de contrôle (K_P, K_I et K_D) qui permettent d'avoir moins de dépassement. Une configuration est affectée à la classe *good* si et seulement si l'*Overshoot* correspondant ne dépasse pas 25% tout en gardant les mêmes intervalles pour le *Settling Time* (doit être compris entre 0 et 7) et le *Rise Time* (varie entre 0 et 2). En appliquant ces intervalles, parmi les 54043 configurations testées, 1479 configurations appartiennent à la classe *good* et 52564 configurations appartiennent à la classe *bad*. Nous appliquons par la suite l'algorithme de *clustering* pour différentes valeurs de *percentage_of_confidence*. Le tableau 4.12 ci-dessous récapitule le nombre de modes de fonctionnement trouvé ainsi que le temps nécessaire pour la classe de *QoC good*.

percentage_of_confidence	*Nbr de mode de fonctionnement*	*temps nécessaire*
0.2	5	4.3483 s
0.4	6	4.1188 s
0.6	7	4.1879 s
0.8	8	4.1396 s

Table 4.12: Résultats du clustering avec les nouveaux ranges pour la classe *good*

Pour les différentes valeurs de *percentage_of_confidence* testées, nous n'avons pas réussi à trouver un paramétrage qui répond aux exigences de *QoC* pour une perturbation égale à 0.6, 0.9 et 1. En fixant le paramètre *percentage_of_confidence* à 0.2, nous avons obtenu 5 modes de fonctionnement en réussissant à regrouper les contextes dont la valeur de perturbation varie entre 0 et 0.3 comme le montre le tableau tableau 4.13.

mode	*Nbr de contextes*	*Disturbance*	K_P	K_I	K_D
1	4	0.0 & 0.1 & 0.2 & 0.3	40	7	31
2	1	0.4	34	4	37
3	1	0.5	34	4	43
4	1	0.7	10	1	43
5	1	0.8	4	1	4

Table 4.13: Paramétrages trouvés pour un *percentage_of_confidence* égal à 0.2

Pour illustrer l'intérêt des résultats obtenus, nous avons effectué une simulation en utilisant le paramétrage trouvé pour les contextes regroupés (i.e., (K_P = 40, K_I = 7, K_D =31)) pour une perturbation égale à 0.3.

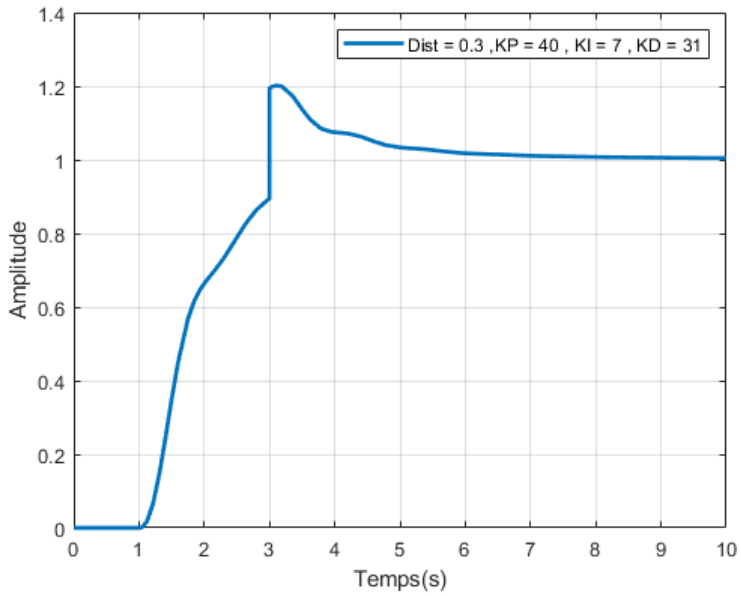

Figure 4.21: Réponse du système en présence d'une perturbation de 0.3

Le résultat de simulation est représenté schématiquement dans la Figure 4.21. Les performances du réglage proposé sont comme suit : un *Settling Time* égal à 5.8439 secondes, un *Rise Time* égal à 1.7443 secondes et un *Overshoot* de l'ordre de 20.1786%. Malgré un début difficile marqués par un pic de l'ordre de 1.2017 (à un instant égal à 3.0938 secondes), le paramétrage trouvé a permis au contrôleur de répondre aux exigences en termes de *QoC*.
Par contre, si nous fixons le *percentage_of_confidence* à 0.8, nous obtenons huit modes de fonctionnement. Nous n'avons pas réussi à regrouper les contextes en imposant cette contrainte. Néanmoins, nous avons effectué une simulation du réglage proposé pour une perturbation égale à 0.4. La réponse du système en boucle fermée avec ce paramétrage est illustré dans la figure 4.22. Le paramétrage trouvé a permis au système de suivre la consigne. Nous constatons que la perturbation a provoqué un pic de l'ordre de 1.1790 à un temps t égal à 3.1875 secondes. Les performances obtenues avec ce réglage sont comme suit : un *Settling Time* égal à 5.5522 secondes, un *Rise Time* égal à 1.7312 secondes et un *Overshoot* de l'ordre de 17.9146%.

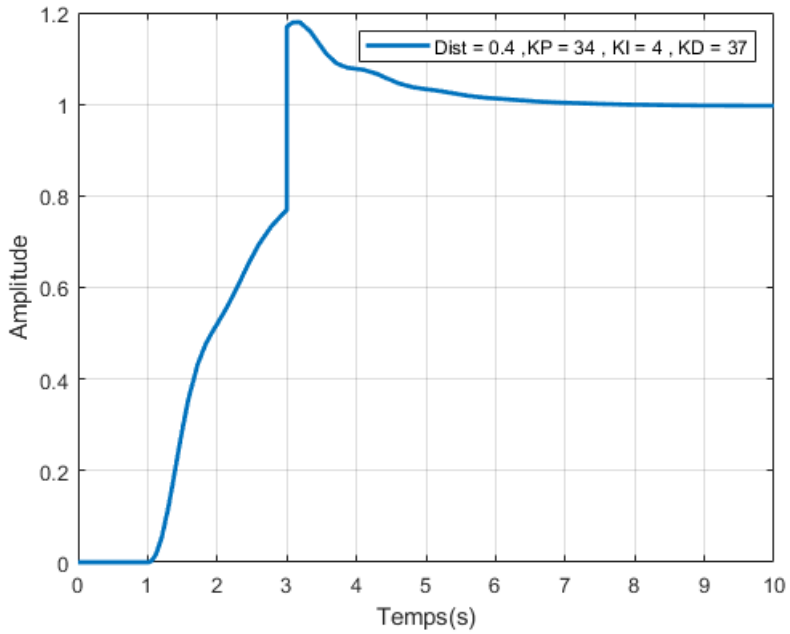

Figure 4.22: Réponse du système en présence d'une perturbation de 0.4

4.2.4 Bilan

L'étude menée permet de vérifier que notre approche constitue un atout dans l'aide à la mise au point des contrôleurs. Évidemment, en simulant plusieurs situations, on peut explorer avantageusement le comportement d'un contrôleur. La comparaison de notre approche avec des techniques parmi les plus utilisées de la théorie du contrôle (*Ziegler-Nichols* et *Root Locus*) nous a permis de vérifier que notre approche est capable de trouver des valeurs de paramètres de contrôle

proches à celles trouvées par les autres techniques en fixant les mêmes exigences en termes de qualité de contrôle. D'autre part notre approche permet de trouver des paramètres de contrôle par rapport à des critères de performance variées (pas seulement en termes de *Settling time*, *Overshoot* ou *Rise time*). On pourrait par exemple imaginer d'autres critères, comme l'erreur statique ou la distance moyenne du leader au follower (ou l'écart type de cette distance) par exemple, ce qui n'est pas possible avec *Ziegler-Nichols* ou *Root locus*).
Le principal avantage de notre approche réside dans le fait qu'elle permet d'explorer de manière automatique différentes situations (des perturbations de toutes intensités) qui nécessiteraient dans les méthodes classiques que nous avons utilisées de relancer à zéro les procédures employées. Refaire ces procédures plusieurs fois pour chaque contexte peut être long et fastidieux. Au contraire, notre approche permet d'automatiser le tuning en trouvant des valeurs valables pour différents contextes, réduisant ainsi le nombre des modes de fonctionnement. Enfin avec notre approche il est possible de mettre au point des contrôleurs associés à des systèmes cyber-physiques pour lesquels on ne dispose pas de modèle physique (équation différentielle , fonction de transfert) à partir du moment ou un modèle permettant de simuler le système est constructible.

4.3 Une étude de cas : leader follower

4.3.1 Le challenge *MDETools'18*

Nous appliquons dans cette partie notre approche à une application de type *Leader-Follower* pour laquelle aucun modèle physique n'est connu. Le cas étudié a été proposé dans le cadre de *MDETools'18* [Pas18] sous la forme d'un challenge (voir figure 4.23).

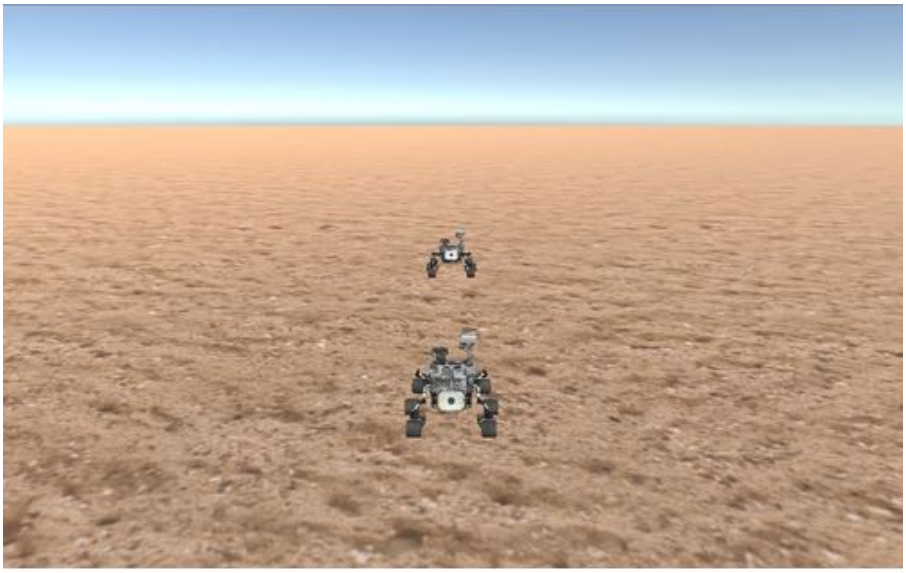

Figure 4.23: Le challenge *Leader Follower*

L'objectif du challenge est de manœuvrer le *follower* dans le but de suivre le *leader* tout en gardant une distance de référence (d_{ref}) fixée à 13. Un contrôleur proportionnel inspiré de [CC12] est utilisé dans le but de corriger la vitesse linéaire et angulaire par rapport à l'erreur de position du *follower* (voir figure 4.24).

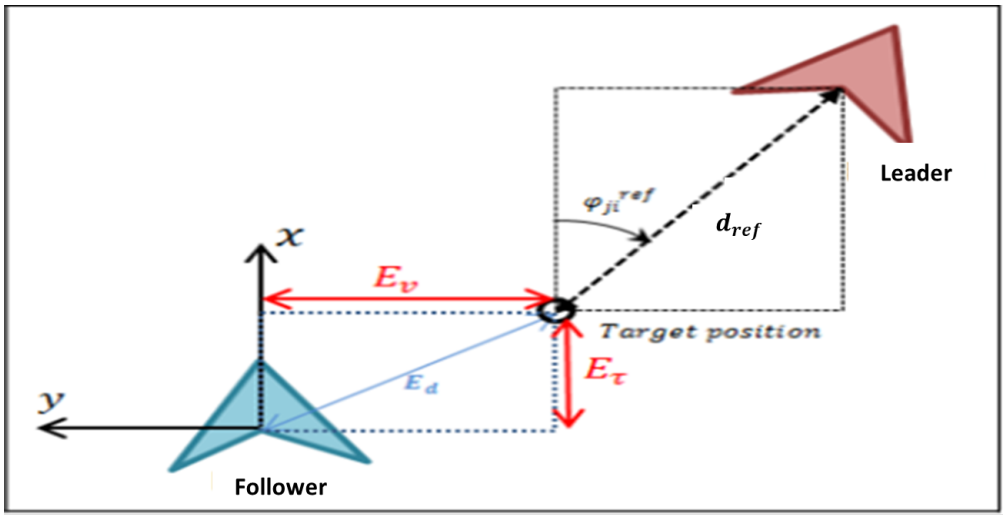

Figure 4.24: Principe du contrôleur proportionnel utilisé

Afin de maintenir la distance de référence à 13, le contrôleur doit connaître à la fois la distance du *follower* par rapport au *leader* qu'on la note $distance_{LF}$ ainsi que la différence (l'écart) de cap entre les deux notée φ_{LF}. Les informations pertinentes telles que la position du *leader* et du *follower* sont obtenues via des capteurs GPS. Pour répondre à l'objectif fixé, la vitesse linéaire (respectivement angulaire) de correction du *follower* est donnée par les équations suivantes :

$$v_{follower} = v_{leader} + (distance_{LF} \cdot \cos(\varphi_{LF}) - d_{ref}) \cdot K_v$$

$$W_{follower} = distance_{LF} \cdot \sin(\varphi_{LF}) \cdot K_w$$

avec d_{ref} désigne la distance de référence et (K_v et K_w) sont les gains pour respectivement la régulation en distance et en angle.
Un simulateur (voir Figure 4.23) est fourni par le challenge qui permet à la fois d'envoyer des commandes via *TCP Socket* au *follower* et de récupérer des données concernant le *leader* et le *follower* (les coordonnées GPS, la distance, le cap ...). Le *follower* est contrôlé en agissant sur la puissance des roues droite et gauche. La puissance est une valeur flottante qui varie entre [-100,100]. La puissance des roues droite et gauche qui doit être appliquée est donnée par les deux équations ci-dessous :

$$P_R = (v_{follower} + {}^{W_{follower}}/2) \cdot P_w$$

$$P_L = (v_{follower} - {}^{W_{follower}}/2) \cdot P_w$$

avec P_w une constante égale à 15.

4.3.2 Notre approche

4.3.2.1 Modélisation du système

Pour pouvoir appliquer notre approche nous avons besoin de faire des simulations. Le simulateur mis à disposition par le challenge *MDETools'18* est un simulateur intégré temps réel. Pour des raisons pratiques (durée de simulation et facilité de paramétrage), nous avons préféré d'utiliser le framework *AMSA* pour modéliser et simuler notre système. Ce framework propose une collection d'outils et de modèles dédiés à la modélisation, la génération de code et la simulation des systèmes cyber-physiques. Le système étudié a été modélisé à l'aide des composants suivants [BGB18] : le système contrôlé (*Follower*), son environnement (*Leader*), les capteurs (*FollowerGPS*, *FollowerCompass*, *LeaderGPS*), l'actionneur (*NormalizedPower*), le contrôleur (*FollowerController*) et un évaluateur de performance (*DistanceMonitor*) (voir figure 4.25).

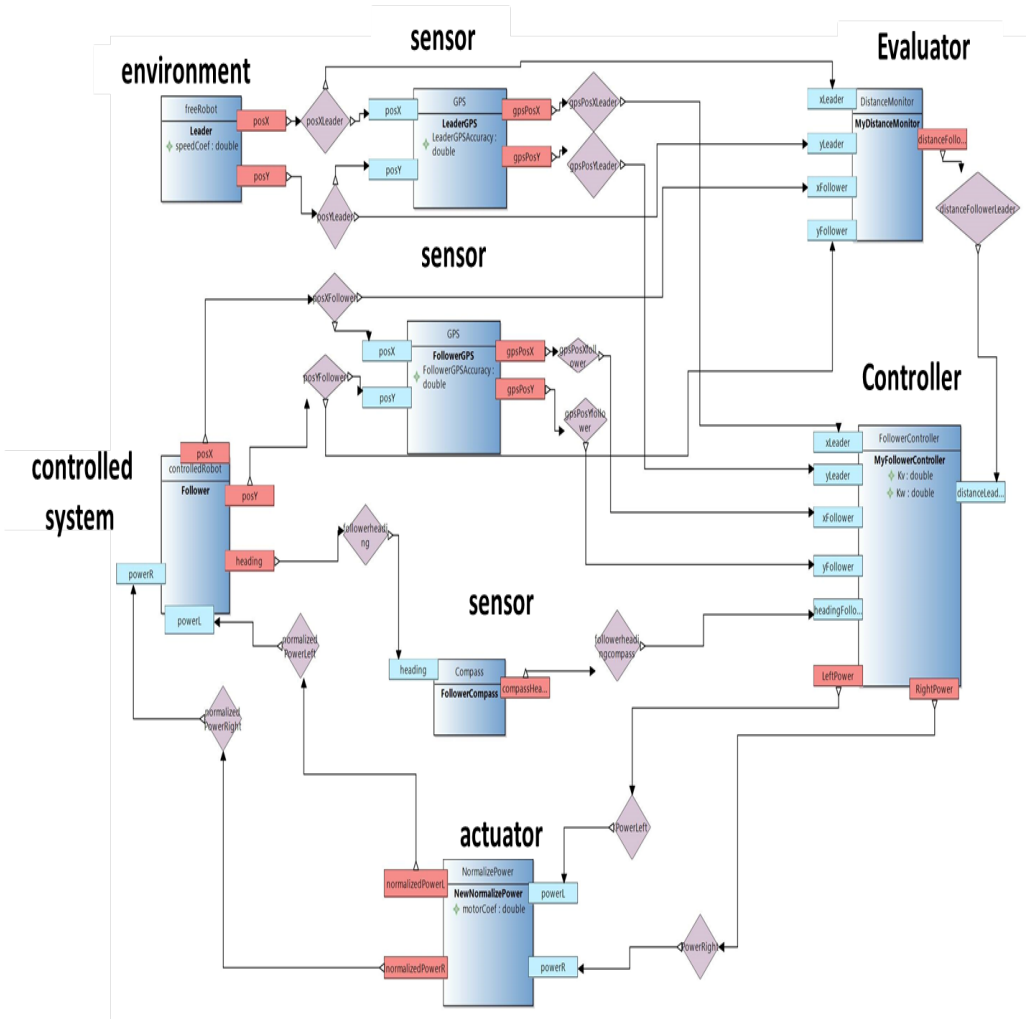

Figure 4.25: Modélisation *AMSA* de l'application *leader follower*

Le *Compass* est un capteur simple qui communique le cap du *follower*. Le *GPS* fournit la position des rovers (*LeaderGPS* fournit la position du *leader* tandis que *FollowerGPS* fournit la position du *follower*). Le *FollowerController* calcule les commandes de puissance (*LeftPower/RightPower*) pour les roues gauche et droite dans le but de corriger l'erreur de distance entre le *follower* et le *leader* en comparant les positions GPS et en tenant compte de la distance et de la vitesse des deux rovers (*leader* et *follower*). Le composant *NormalizedPower* garantit que la puissance émise notée (*normalizedPowerR ou normalizedPowerL*) est comprise entre -100 et 100. Le *DistanceMonitor* calcule la distance réelle entre les deux rovers. Au démarrage de la simulation, par souci de simplicité, nous imposons que le *leader* se déplace en ligne droite. La position et le cap du *follower* dépendent de la puissance reçue de la part du contrôleur (*powerR* et *powerL*).

4.3.2.2 Exploration

Une mauvaise qualité des capteurs et actionneurs (i.e., un capteur peut ne pas fournir des mesures suffisamment précises et les actionneurs peuvent s'user) impactent la qualité de contrôle. De même, la vitesse du *leader* qui a un impact sur le réglage du contrôleur. Nous avons concentré notre exploration sur les paramètres contextuels liés à ces deux parties. Dans le cadre de cette expérimentation, quatre paramètres contextuels $\{P_{Ctxt}\}$ sont considérés :

- le paramètre *LeaderGPSAccuracy* caractérise la précision de la mesure fournie par les capteurs *GPS* liée à la position du *leader*.
- le paramètre *FollowerGPSAccuracy* caractérise la précision de la mesure fournie par les capteurs *GPS* liée à la position du *follower*.
- le paramètre *SpeedCoef*, également appelé coefficient de vitesse, caractérise le rapport entre la vitesse réelle du *Leader* et une vitesse nominale.
- le paramètre *MotorCoef* définit le pourcentage de puissance fournie par l'actionneur. Il caractérise l'efficacité du moteur.

De plus, deux paramètres de contrôle $\{P_{Ctrl}\}$ sont considérés dans le cadre de cette expérimentation pour caractériser le contrôleur qui se base sur une loi proportionnelle :

- le paramètre K_v est utilisé pour corriger la vitesse linéaire proportionnellement à l'erreur de position du *follower*.
- le paramètre K_w est utilisé pour corriger la vitesse angulaire proportionnellement à l'erreur de cap du *follower*.

Notre approche vise à trouver des valeurs optimales pour les deux paramètres de contrôle K_v et K_w tout en tenant compte des divers contextes qui pourraient être rencontrés. Le tableau 4.14 liste les six paramètres utilisés dans nos expériences. Il est à noter que l'espace de recherche englobe l'ensemble des valeurs à virgule flottante dans les ranges spécifiés dans ce ta bleau. Les paramètres contextuels représentent des pourcentages, ce qui explique le choix de l'intervalle

[0.1,1] (i.e., 0.1 est équivalent à 10%, tandis que 1 est équivalent à 100%) pour le *LeaderGPSAccuracy*, le *FollowerGPSAccuracy* et le *MotorCoef*, alors que le *Speed-Coef* est une valeur flottante comprise dans [0.5,5] (i.e., 0,5 correspond à un *leader* qui avance lentement à la moitié de la vitesse théorique tandis que 5 représente un *leader* cinq fois plus rapide).
Pour des raisons de simplicité, nous supposons que le *leader* suit une trajectoire rectiligne. L'intervalle du paramètre K_v (respectivement K_w), a été choisi après avoir effectué des simulations avec diverses valeurs de K_v(respectivement K_w) avec un pas de 10. A l'issue des simulations, l'intervalle de K_v est [0.1,3] avec un pas de 0.5, alors que K_w varie entre [0.01,0.25] avec un pas de 0.04. Durant cette étape, 10000 contextes sont évalués et donc 490000 configurations sont simulées. La simulation est effectuée par le biais d'un code C++ qui a été partiellement généré (structure du code) à partir du modèle *AMSA* et complété à la main (code métier). La simulation des configurations a duré environ une heure et 45 minutes (5083,818209 secondes exactement) sur un PC équipé d'un processeur *Intel(R) Core(TM) i7-7600U CPU* et cadencé à 2.9 *GHz*.

Paramètre	*range*	*Step*
LeaderGPSAccuracy	[0 , 1]	0.1
FollowerGPSAccuracy	[0 , 1]	0.1
SpeedCoef	[0.5 , 5]	0.5
MotorCoef	[0.1 , 1]	0.1
K_v	[0.1 , 3]	0.5
K_w	[0.01 , 0.25]	0.04

Table 4.14: Plage des paramètres pour l'étape d'exploration

4.3.2.3 *Binning*

Puisque l'objectif du contrôleur consiste à garder une distance de référence entre le *leader* et le *follower*, nous avons choisi la distance moyenne pendant la période de simulation comme critère pour évaluer le contrôleur. Nous associons les configurations testées à l'une des trois classes de *QoC* appelées *good*, *average* et *bad*. En ce qui concerne la qualité du contrôle, pour une distance moyenne située dans l'intervalle [11,15], la configuration correspondante est associée à la classe *good*. Pour une distance moyenne entre 15 et 20, la configuration est associée à la classe *average*. Enfin, si la distance moyenne enregistrée dépasse 20 ou est inférieure à 6, les paires (K_v,K_w) sont associées à la classe *bad* (voir figure 4.26).

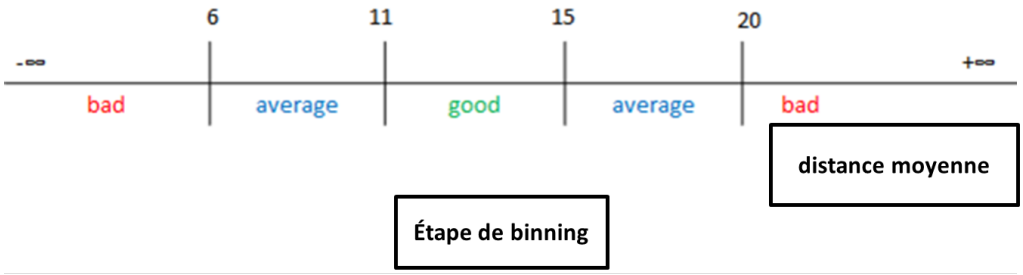

Figure 4.26: Les ranges utilisés pour le binning

Sur les 490 000 configurations testées, 406431 sont *bad*, 57576 sont *good* et 25993 sont *average*.

4.3.2.4 *Feature Selection*

L'ensemble de données récupéré de l'étape précédente, correspond à une tâche de classification sur laquelle nous devons prédire si une configuration est bonne (*good*) ou non (*bad* or *average*) en se basant sur quatre caractéristiques (features) qui correspondent aux paramètres contextuels $\{P'_{Ctxt}\}$. Le jeu de données (dataset) comporte un total de 490000 observations. La caractéristique cible est la classe de *QoC*. Nous appliquons ensuite différents algorithmes de sélection de caractéristiques afin d'identifier les paramètres contextuels $\{P'_{Ctxt}\}$ qui contribuent le plus à prédire la caractéristique cible (target feature). Pour ce faire, nous procédons à une évaluation de toutes les combinaisons possibles de caractéristiques par rapport à un critère d'évaluation. Pour la classification, le critère d'évaluation adopté est la précision. A la fin de cette phase la paire *MotorCoef* et *SpeedCoef* ont été sélectionnés par les algorithmes de sélection utilisés (i.e., *le Step Forward Feature Selection* ou *le Step Backward Feature Selection*). La paire *LeaderGPSAccuracy* et *FollowerGPSAccuracy* ont été éliminés. Puisque le *leader* avance tout droit, il paraît logique que le contrôleur n'a pas vraiment besoin de connaître la position de ce dernier. Au contraire, les performances du contrôleur dépendent étroitement de la vitesse du *leader* et notamment de la puissance fournie au actionneurs. Ainsi le sous-ensemble de caractéristiques qui produit les meilleures performances est composé de (*MotorCoef*, *SpeedCoef*) comme paramètres contextuels $\{P'_{Ctxt}\}$ (voir figure 4 .27). Il faut noté que cette étape de sélection est relativement rapide. Le temps nécessaire à la sélection de caractéristiques est de l'ordre de 6 min (353,55 secondes exactement) sur un PC équipé d'un processeur *Intel(R) Core(TM) i7-7600U CPU* et cadencé à 2.9 *GHz*.

4.3.2.5 *Exploration bis*

En considérant les paramètres contextuels sélectionnés $\{P'_{Ctxt}\}$ (i.e., *MotorCoef*, *SpeedCoef*), nous avons utilisé un pas plus petit pour les paramètres de contrôle $\{P_{Ctrl}\}$ dans le but d'explorer davantage les limites d'adaptabilité et de trouver des paramètres de contrôle qui conviennent à un grand nombre de contextes.

```
Forward Feature Selection Index(['SpeedCoef', 'MotorCoef'], dtype='object')
Step Backwards Feature Selection Index(['SpeedCoef', 'MotorCoef'], dtype='object')
```

```
Features: 1/6
Features: 2/6
Features: 3/6
Features: 4/6
Features: 5/6
Features: 6/6Best subset (indices): (2, 3)
Best subset (corresponding names): ('SpeedCoef', 'MotorCoef')
selected features: ('SpeedCoef', 'MotorCoef')
```

Figure 4.27: Résultats des algorithmes de feature selection

Le Tableau 4.15 liste les paramètres contextuels sélectionnés, les paramètres de contrôle, leurs nouveaux range et les nouveaux pas utilisés.

Paramètre	*range*	*Step*
SpeedCoef	[0.5 , 5]	0.5
MotorCoef	[0.1 , 1]	0.1
K_v	[0.1 , 10]	0.1
K_w	[0.01 , 1]	0.01

Table 4.15: Nouvelle plage des paramètres pour l'étape d'exploration

Pour cette étape, nous considérons que K_v varie entre 0.1 et 10 avec un pas de 0.1, tandis que K_w se trouve entre 0.01 et 1 avec un pas de 0.01. Pour le clustering, nous commençons par diviser les configurations en trois sous-ensembles, un par classe de *QoC* (*good*, *average* and *bad*).
Nous constatons que pour un *MotorCoef* sous la barre de 0.3 (i.e., l'actionneur est seulement capable de fournir moins de 30% de la puissance nécessaire au roues) et/ou pour un *SpeedCoef* au delà de 2 (i.e., le *leader* va trois fois plus vite), le contrôleur n'était pas en mesure de répondre aux objectifs fixés. Même pour un *SpeedCoef* de 0.5 (i.e., le *leader* avance lentement), le contrôleur ne parvient pas à maintenir la distance de référence souhaitée entre le *leader* et le *follower*. Afin d'étudier précisément les limites du contrôleur, nous avons exploré plus finement les limites d'adaptation du contrôleur. Pour cela, nous lançons de nouvelles simulations en se concentrant sur un *MotorCoef* qui se situe dans l'intervalle [0.2,0.3] avec un pas plus petit de 0.01 au lieu de 0.1 et un *SpeedCoef* qui évolue dans l'intervalle [1.8,2.4] avec un pas de 0.1. Il s'est avéré que pour un *MotorCoef* au dessous de 0.24, peu importe les valeurs des paramètres de contrôle $\{P_{Ctrl}\}$ (K_v and K_w), le contrôleur n'est plus capable de répondre à l'objectif et la distance entre le *leader* et le *follower* continue à s'accroître (voir figure 4.28).

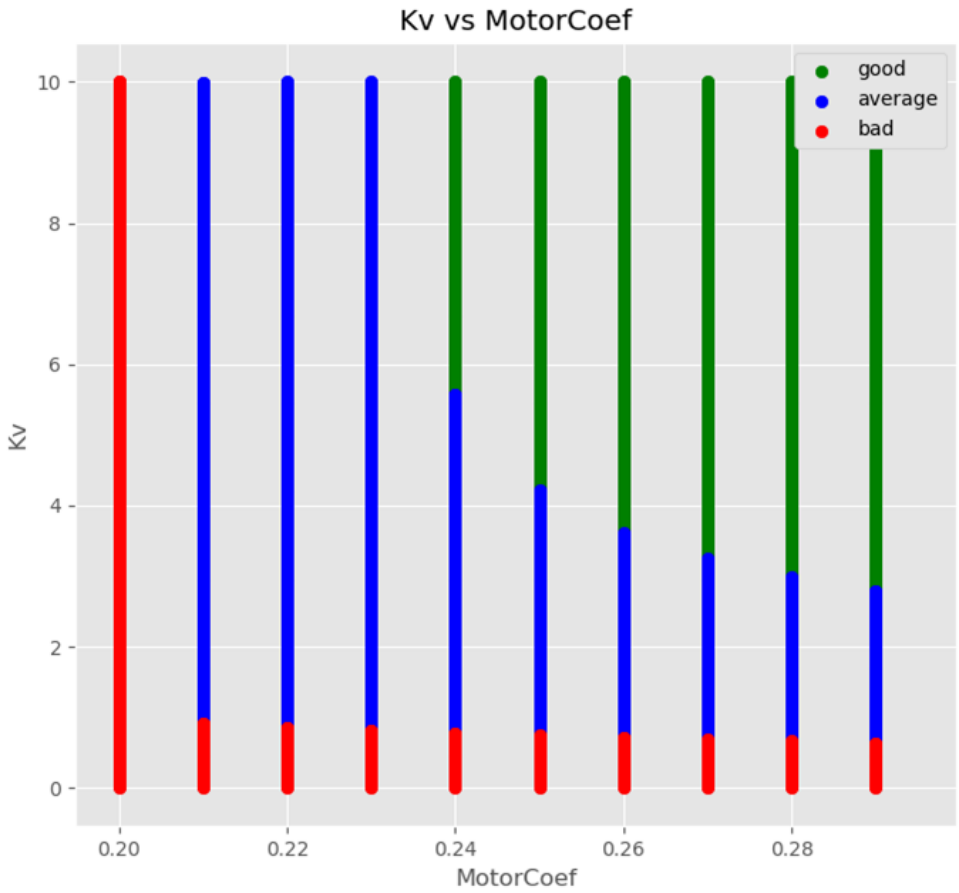

Figure 4.28: Limite d'adaptation du contrôleur

En ce qui concerne le *SpeedCoef*, il s'est avéré que pour un *SpeedCoef* qui dépasse 2.6, le contrôleur devient incapable d'atteindre l'objectif qui lui a été fixé.

4.3.2.6 Clustering

Parmi les 1.010.000 configurations testées, 270.000 configurations appartiennent à la classe de *QoC good*. Ces paramètres de contrôle permettent au contrôleur de répondre avec succès aux objectifs fixés. Ensuite, un ensemble de paramètres de contrôle $\{P_{Ctrl}\}$ composé par les paires (K_v,K_w) appartenant à la classe de *QoC good* est déterminé pour chaque contexte. C'est-à-dire que pour chaque combinaison de (*MotorCoef*,*SpeedCoef*), on associe une liste de doublets formés par K_v et K_w appartenant à la classe notée *good*. Pour cette expérimentation, nous avons pu associer pour seulement 21 contextes (sur 100) (*MotorCoef*,*SpeedCoef*) une liste contenant au moins un doublet (K_v,K_w) appartenant à la classe *good*. Finalement, le chevauchement des contextes est calculé pour différentes valeurs du *percentage_of_confidence* (0.2,0.4,0.6 and 0.8). Le tableau 4.16 présente le nombre de clusters obtenus (i.e., nombre de modes de fonctionnement) ainsi que le temps nécessaire pour les obtenir.

percentage_of_confidence	*Nbr of operational modes*	*Time needed*
0.2	2	144.65 s
0.4	2	130.65 s
0.6	3	211.64 s
0.8	6	156.49 s

Table 4.16: Résultats du clustering

4.3.2.7 Configuration

Dans le but d'avoir un nombre minimal de modes de fonctionnement, nous devons fixer le *percentage_of_confidence* à 0.2 ou 0.4. Dans ce cas, le doublet (K_v= 9.0, K_w= 0.01) est associé au premier mode de fonctionnement relié au contexte (*MotorCoef* = 1.0, *SpeedCoef* =2.5). Un K_v égal à 3.8 et un K_w égal à 0.07 est choisi pour l'ensemble des 20 contextes restants et ce paramétrage va caractériser ainsi le deuxième mode de fonctionnement.
Pour un *percentage_of_confidence* de 0.6, nous avons réussi à obtenir trois modes de fonctionnement : le premier mode est composé de quatorze contextes et est associé au doublet (K_v= 3.8, K_w= 0.07). Le deuxième mode de fonctionnement englobe six autres contextes et le paramétrage de contrôle optimal associé est composé d'un K_v qui vaut 1.8 et d'un K_w égal à 0.01. Le troisième mode est associé à un seul contexte et sera représenté par le doublet (K_v= 9.0, K_w= 0.01) (voir tableau 4.17).

mode	*Nbr de contextes*	K_v	K_w
1	14	3.8	0.07
2	6	1.8	0.01
3	1	9	0.01

Table 4.17: Paramétrages trouvés pour un *percentage_of_confidence* égal à 0.6

En utilisant un *percentage_of_confidence* qui vaut 0.8, six modes de fonctionnement ont été obtenus.

mode	*Nbr de contextes*	K_v	K_w
1	9	3.1	0.01
2	1	0.9	0.01
3	3	2.1	0.01
4	6	1.8	0.01
5	1	3.8	0.07
6	1	9	0.01

Table 4.18: Paramétrages trouvés pour un *percentage_of_confidence* égal à 0.8

Le premier mode contient neuf contextes et est caractérisé par un K_v et un K_w respectivement égaux à 3.1 et à 0.01. Le deuxième mode est associé au contexte formé par un *MotorCoef* égal à 0.8 et un *SpeedCoef* égal à 1.0. Les paramètres

de contrôle $\{P_{Ctrl}\}$ proposés pour ce mode est le doublet (K_v= 0.9, K_w= 0.01). Le troisième mode inclut trois contextes et est associé à un K_v égal à 2.1 et à un K_w égal à 0.01. Le quatrième mode englobe six contextes et est représenté par le doublet (K_v= 1.8,K_w= 0.01). Le cinquième mode est associé au contexte caractérisé par un *MotorCoef* égal à 0.7 et à un *SpeedCoef* qui vaut 2.5. Le paramétrage optimal proposé $\{P_{Ctrl}\}$ pour ce mode est un K_v qui vaut 3.8 et un K_w de 0.07. Le sixième et dernier mode est relié au contexte formé par un *MotorCoef* égal à 1.0 et à un *SpeedCoef* qui vaut 2.5 et est représenté par un K_v égal à 9.0 et un K_w de l'ordre de 0.01.

4.3.2.8 Régression

Dans cette partie, nous procédons à une évaluation du modèle de réseau de neurones pour la régression construit à l'aide de *Keras* et qui a été mis en place pour prédire des "bons" paramètres de contrôle ($P_{Ctrl_{opt}}$) en termes de *QoC* pour des contextes inconnus. Le jeu de données utilisé dans ce travail contient 270.000 lignes et 4 variables. Les variables dépendantes, appelées aussi prédicteurs incluent les paramètres contextuels sélectionnés, c'est à dire un *MotorCoef* appartenant à l'intervalle [0.3,1] et un *SpeedCoef* qui varie entre 1 et 2. Les variables cibles sont composées des paramètres de contrôle K_v et K_w. Toutes les configurations utilisées pour cette partie font partie de la classe de *QoC good*. Ensuite, nous construisons un modèle de régression moyennant *Keras*. L'objectif est de construire un réseau de neurones qui représente la meilleure adéquation entre les variables dépendantes et les variables cibles contenues dans le *training set* conformément à un critère d'évaluation. Le modèle est composé de quatre couches. La première couche spécifie la fonction d'activation et le nombre de paramètres d'entrées (*MotorCoef* et *SpeedCoef*). Le modèle comprend également deux couches cachées *hidden layers* et une couche de sortie qui donne K_v et K_w . La fonction d'activation utilisée dans les couches cachées est *rectified linear unit ReLU* en raison de sa capacité à considérer la non-linéarité. L'algorithme d'optimisation *ADAM* est utilisé et une fonction d'erreur quadratique moyenne est optimisée. La performance du modèle est évaluée en calculant la racine carrée de l'erreur quadratique moyenne entre les valeurs prévues et réelle (cette dernière étant notée *RMSE*). Nous avons également fixé l'argument *epochs*) à 20. Il convient de noter qu'un *epoch* est équivalent à un cycle d'entraînement complet sur le *training set*.
Nous avons ensuite divisé le jeu de données en *training* (70%) et *testing sets* (30%). Le modèle de régression conçu via *Keras* est ensuite estimé et ajusté (*fitted*) sur le *training set* afin de faire des prédictions par la suite sur le *testing set* et de calculer les métriques d'évaluation. Les résultats indiquent une petite erreur quadratique moyenne *RMSE* (0.0086). Les valeurs de K_v et K_w prédites ont été testées en refaisant la simulation afin de s'assurer que les paramètres de contrôle proposés $\{P_{Ctrl}\}$ répondent à l'objectif fixé du contrôleur. Les résultats démontrent que seulement 1,23% des paramètres de contrôle prédits (K_v et K_w) n'appartiennent pas à la classe de *QoC good*. Les différentes métriques utilisées pour évaluer le modèle de régression *Keras* montrent des performances élevées de ce modèle avec une bonne précision de prédiction (Voir tableau 4.7).

Training_size	*Test_size*	*RMSE*	*Nbr_of_bad_prediction*
189000	81000	0.0086	996 (1.23%)

Table 4.19: Mesures de performance du modèle de régression

4.3.3 Validation sur une plateforme réaliste

Pour démontrer l'applicabilité de notre approche, nous avons effectué des tests sur un code C++ ROS (Robot Operating System) (généré via *AMSA*). Ce code pourrait être embarqué sur un robot compatible ROS [Qui+09] .

4.3.3.1 Génération de code *ROS*

Nous présentons dans ce qui suit une description de la procédure de génération de code *ROS* (nœuds, topics, launch files et scénario) en utilisant le framework *AMSA*. Le générateur de code est basé sur la technologie *Acceleo* [Del11]. La génération de code *ROS* respecte les principes suivants :

- les ressources de ROS sont organisées dans une structure appelée package. Un package est un répertoire qui contient les nœuds, les librairies externes, des données, des fichiers de configuration et un fichier de configuration xml nommé manifest.xml. En utilisant *AMSA*, un seul package *ROS* est créé avec le nom du composant racine (root-component).
- nous adoptons la communication par échange de *topic* entre les différents noeuds *ROS*. Un *topic* est un système de transport de l'information basé sur le système de l'abonnement/publication (subscribe/ publish). Un *topic* est ensuite défini pour chaque donnée et également défini pour chaque paramètre. Un *topic* est identifié par son nom et est fortement typé par le type de message *ROS* utilisé pour publier.
- un scénario est généré comme un nœud spécifique mettant en œuvre une séquence de modifications de paramètres dans une période de temps fixe.
- un noeud *ROS* est crée pour chaque composant basique.
- le code généré est écrit en langage C ++.
- le code métier spécifique au domaine tel que la loi de contrôle est ajouté au code généré afin de le compléter.

Un fichier C++ est généré pour chaque noeud *ROS* et donc pour chaque composant *Leaf*. La figure 4.29 montre un aperçu du squelette de code généré pour chaque nœud.

```
//User declarations
//Start of user code Declarations for Regulator
float precision=0.5;
//End of user code
//Infinite loop
while (ros::ok()) {
 // User body
 //Start of user code Body for Regulator
 if (abs(set_point.data-heading_sensor.data)>precision) {
  power.data=Kv.data*(set_point.data-heading_sensor.data);
 } else {
  power.data=0;
 }
 //End of user code
 //Publish output
 my_publisher_object.publish(power);
 ROS_INFO("force command =%f",power.data);
 ros::spinOnce();//allow data update from callback
 //Wait for the next period
 naptime.sleep();//wait for remainder of specified period
}
return 0;//should never get here, unless roscore dies
```

Figure 4.29: Exemple de Noeud *ROS* généré

Dans notre exemple, la génération donne lieu à 9 fichiers C ++. Un fichier pour chaque nœud *ROS* est généré pour chacun des 8 composants *Leaf* et un fichier de plus pour le scénario. Le code généré est complété à la main pour inclure le code métier de la loi de contrôle, les *user includes* et la déclaration des variables locales. Un fichier de lancement (*launch file*) est également généré. L'architecture *ROS* suit l'architecture *AMSA* comme le montre la figure 4.30.

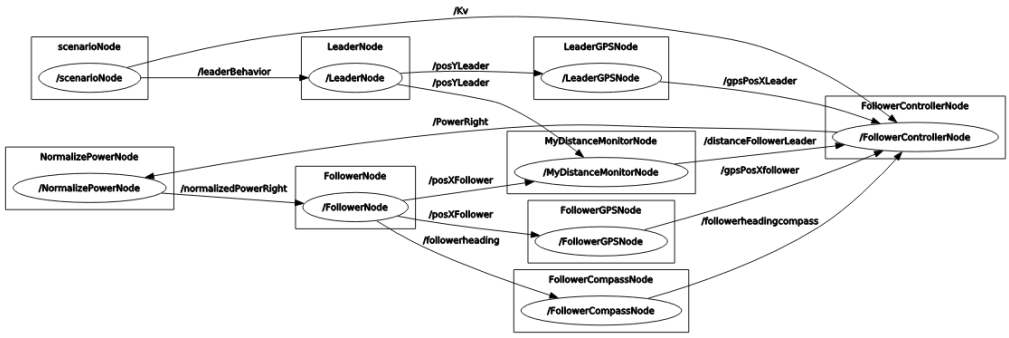

Figure 4.30: Graphe ROS : nœuds sous forme d'ovales, topics sous forme de carrés

4.3.3.2 Simulation ROS

Le but de ce chapitre est d'illustrer l'impact du paramétrage de la loi de contrôle proposé par notre approche sous *ROS*. *MyFollowerController* est le nœud principal qui implémente l'algorithme de contrôle visant à préserver la distance de référence entre le *leader* et le *follower*. L'interface utilisateur *Qt* dédiée à *ROS* est utilisée afin d'aider l'utilisateur à évaluer la qualité du contrôle, qui correspond dans cette expérimentation à évaluer la distance entre le *leader* et le *follower*. Dans un premier scénario, la configuration est la suivante : *LeaderGPSAccuracy* = 0.6, *FollowerGPSAccuracy* = 0.5, *MotorCoef* = 0.6 et *SpeedCoef* = 2. En ce qui concerne les paramètres de contrôle $\{P_{Ctrl}\}$, nous avons choisi un K_v qui vaut 3.1 et un K_w égal à 0.01 (paramétrage proposé par notre approche dans le tableau 3.17). Dans ce premier scénario et malgré la perte de précision des données *GPS* du *leader* et l'inefficacité du moteur, après un temps de stabilisation, la distance entre le *leader* et le *follower* est pratiquement stable à 15 (voir figure 4.31).

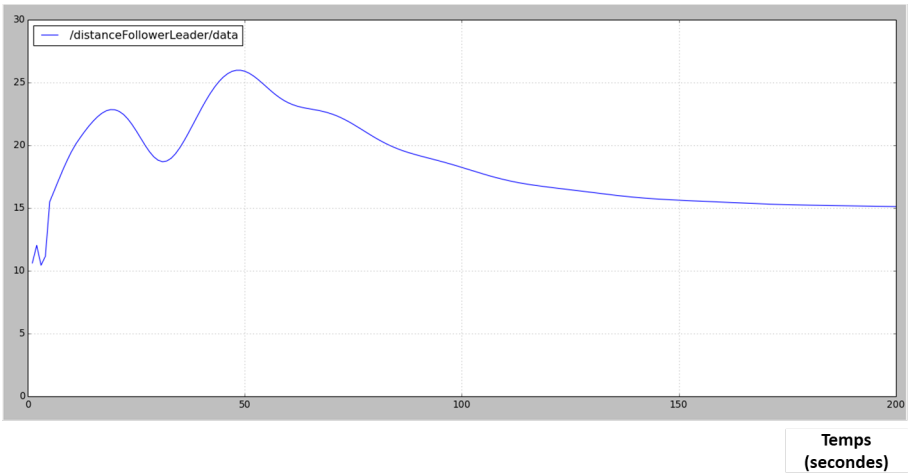

Figure 4.31: Scénario 1 : distance entre le *leader-follower* en fonction du temps (en secondes)

Nous notons que la distance maximale entre les deux rovers est de 27 et la durée (le temps) pour la stabiliser est d'environ 155 secondes. Le paramétrage proposé par notre approche prend en compte le contexte dans lequel évolue le contrôleur et est capable de maintenir une distance proche de la distance souhaitée entre le *leader* et le *follower* (distance fixée à 13).
Pour le scénario 2, nous avons utilisé un paramétrage de contrôle $\{P_{Ctrl}\}$ constitué d'un K_v qui vaut 1.8 et un K_w égal à 0.01 pour un contexte caractérisé par un *MotorCoef* égal à 0.8, un *SpeedCoef* égal à 1.5, un *LeaderGPSAccuracy* = 0.9, et un *FollowerGPSAccuracy* = 0.8. La figure 4.32 illustre la distance entre le *leader* et le *follower* pour le scénario 2.

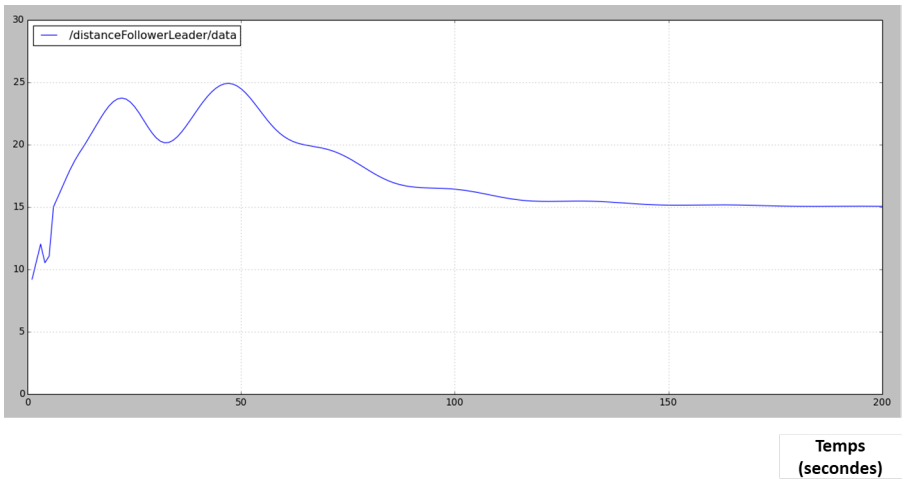

Figure 4.32: Scénario 2 : distance entre le *leader-follower* en fonction du temps (en secondes)

Dans ce cas, nous constatons que la distance entre les deux rovers se stabilise autour de 15 à partir d'un temps égal à 140 secondes. A noter que la distance maximale n'a pas dépassée 25 m.
Pour le troisième scénario, et pour illustrer l'intérêt du paramétrage obtenu par prédiction, nous utilisons les valeurs prédites (K_v = 3.1, K_w = 0.01) pour un contexte non simulé caractérisé par (un *LeaderGPSAccuracy* = 0.6, un *FollowerGPSAccuracy* = 0.5, un *MotorCoef* = 0.6, et un *SpeedCoef* = 2). Les résultats obtenus (voir figure 4.33) montrent que le contrôleur est capable de s'adapter et de maintenir la distance entre le *leader* et le *follower* à 15. En utilisant ce paramétrage, nous notons que la distance maximale entre les deux rovers était de 32 m et qu'il faut près de 150 secondes pour atteindre la distance de référence. On note également la présence d'oscillations plus fortes.

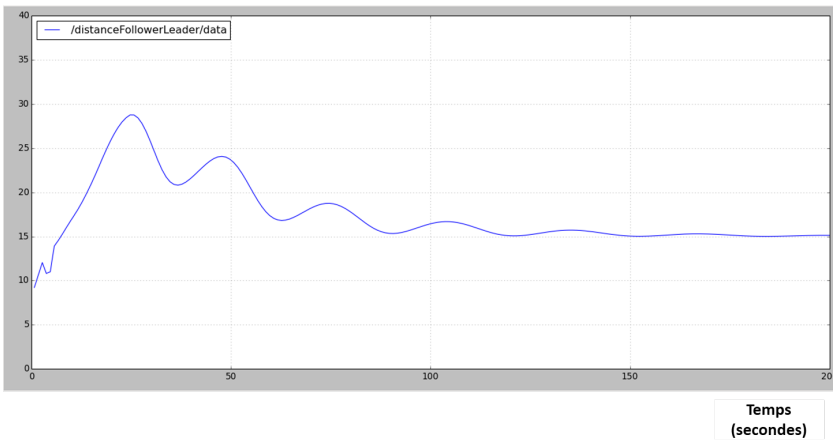

Figure 4.33: Scénario 3 : distance entre le *leader-follower* en fonction du temps (en secondes)

4.4 Bilan

L'étude menée illustre l'intérêt de notre approche pour automatiser et faciliter la mise au point des contrôleurs associés aux systèmes cyber-physiques évoluant dans un environnement incertain. L'approche permet d'explorer divers contextes auxquels le système contrôlé peut être confronté grâce à la simulation. L'utilisation d'algorithmes de réduction de dimensionnalité, plus spécifiquement de sélection de caractéristiques permet de réduire le nombre de contextes à explorer. Il est alors possible de focaliser l'exploration sur les contextes qui ont plus d'impact sur la qualité de contrôle *QoC*. En ne considérant que les paramètres contextuels sélectionnés, nous procédons à une étape de *clustering* afin d'identifier pour un groupe de contextes un réglage adéquat (paramètres de contrôle). Cette procédure permet d'avoir un contrôleur avec moins de modes de fonctionnement et surtout avec un paramétrage de contrôle adapté pour différents contextes. Finalement, l'application d'un modèle de prédiction qui se base sur les données historiques garantit de trouver des réglages adaptés pour des situations imprévues. Les résultats obtenus par notre approche montrent qu'il est possible d'utiliser l'approche pour produire un contrôleur doté de capacités d'adaptation.

Chapitre 5

Conclusion et perspectives

5.1 Conclusion

Ce travail de thèse porte sur la mise au point des contrôleurs de systèmes cyber-physiques évoluant en environnement incertain. L'idée était de proposer une approche pour automatiser le réglage des contrôleurs dans le but d'obtenir un système doté de capacité d'adaptation face aux variations de l'état du système et aux perturbations de son environnement. Nous dressons ici un bilan des différentes contributions qui ont été proposées.
L'approche proposée permet d'explorer divers contextes auxquels le système contrôlé peut être confronté grâce à la simulation guidée par un modèle. Elle ne nécessite pas une connaissance au préalable du modèle du système contrairement aux méthodes classiques (traditionnelles) de la théorie du contrôle. L'approche établit une séparation claire des préoccupations entre la spécification du contrôleur et de son environnement, la description de la qualité de contrôle *QoC* et l'identification des modes de fonctionnement. Avec les travaux réalisés et présentés dans cette thèse, nous avons atteint principalement trois objectifs.
Le premier est de réduire le nombre de situations (contextes) à explorer en appliquant des algorithmes de réduction de dimensionnalité, plus spécifiquement de sélection de caractéristiques. Ceci nous permet de concentrer l'exploration sur les contextes qui contribuent le plus à qualifier un réglage de contrôle en termes de qualité de contrôle *QoC*. Le deuxième objectif est ici d'avoir un contrôleur avec moins de modes de fonctionnement via un paramétrage de contrôle adapté pour différents contextes.
En ne considérant que ces situations (contextes), nous procédons à un *clustering* dans le but de regrouper les contextes possédant un réglage de contrôle similaire (i.e., ayant des valeurs de paramètres de contrôle conduisant à la même qualité de contrôle) au sein d'un même groupe. Le regroupement proposé permet d'éviter de changer de réglage de contrôle suite à chaque changement de contexte. Le troisième objectif est d'assurer la prédiction d'un réglage à partir des données historiques issues de la simulation. Ce module garantit de trouver des réglages (valeurs pour les paramètres de contrôles) adaptés pour des situations (contextes) imprévues et non explorées. Le système cyber-physique dispose donc des capacités d'adaptation à des contextes inconnus a priori, et

il est capable de proposer des capacités de production de réglages adaptés en fonction de mesures effectuées.
L'approche proposée peut être utilisée de manière itérative afin de préciser la limite d'adaptabilité du contrôleur (i.e., les contextes pour lesquels aucun paramétrage de contrôle ne permet de répondre aux exigences souhaitées en termes de *QoC*).
L'approche proposée a été validée en la comparant avec les résultats des techniques les plus utilisées de la théorie du contrôle à savoir *Ziegler-Nichols* et *Root Locus* pour le réglage de systèmes dont la fonction de transfert est connue. L'approche a aussi été appliquée pour aider à mettre au point un contrôleur proportionnel dans le cadre d'une application de type *leader follower*. Les résultats obtenus par notre approche montrent qu'il est possible de l'utiliser pour produire un contrôleur doté de capacités d'adaptation face aux changements de contextes.

5.2 Perspectives

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

1. Au niveau de la modélisation, des améliorations sont possibles sur les points suivants : dans cette thèse, nous avons choisi de modéliser la variabilité du comportement du système cyber-physique et de son environnement à travers la définition de paramètres variables numériques sous forme d'attributs. Une première piste d'amélioration consiste à étendre l'approche à d'autres types de paramètres à savoir les énumérations, les booléens... Au moment de formuler notre approche, nous avons émis l'hypothèse que les paramètres considérés pour l'exploration doivent être indépendants, il serait judicieux d'examiner et de mettre en place une stratégie pour intégrer les cas où les paramètres sont dépendants entre eux.

2. En ce qui concerne la politique de balayage, nous avons essentiellement utilisé un balayage exhaustif des paramètres en fixant un intervalle (*Max* et *Min*) et un pas (*Step*) pour chacun d'eux. Il est nécessaire d'examiner les possibilités d'utiliser d'autres techniques de balayage plus intelligentes telles que les approches probabilistes [Gor+14] et notamment les *réseaux Bayésiens* [KF09] ou encore les méthodes à intervalles [Dăn07] (analyse par intervalle) pour essayer de diriger la simulation directement vers les zones contenant des bons réglages (paramétrage) de contrôle en termes de qualité de contrôle *QoC* au lieu de considérer toutes les combinaisons possibles de paramètres.

3. Pour la phase de *binning*, nous avons exigé qu'un expert en automatique intervienne pour fixer les intervalles des différentes classes de qualité de contrôle *QoC*. Il serait nécessaire de développer une interface homme-machine pour guider l'utilisateur dans la définition des classes de *QoC*,

soit leur nombre, leurs noms ainsi que les intervalles associés à chaque classe. Une telle interface pourrait également inclure des outils de visualisation des critères de performances.

4. Bien que l'algorithme de *clustering* proposé réponde aux objectifs visés par cette étape, l'algorithme présente un défaut de sensibilité vis-à-vis de l'ordre dans lequel nous traitons les contextes. Il serait intéressant d'évaluer d'autres stratégies pour définir les modes de fonctionnement.

5. On pourrait réfléchir à étendre l'application de l'approche proposée au contrôle de plusieurs systèmes cyber-physiques à la fois (par exemple pour un essaim de drones). Concrètement, on étendrait l'approche proposée pour tenir compte du contexte et de la situation des voisins.

6. Afin de rendre les simulateurs réalistes, il serait utile d'améliorer le générateur de code *ROS* pour faciliter la tache d'intégration du code métier et générer un comportement similaire au simulateur. Le but est de permettre à l'expert de produire du code métier sans avoir à connaître les détails de l'implémentation.

7. Il serait très intéressant d'intégrer la mise au point des paramètres dans la version embarquée afin de réaliser l'apprentissage des paramètres qui a été fait à base de simulation à partir de données mesurées.

Bibliography

[AB95] David W. Aha and Richard L. Bankert. "A Comparative Evaluation of Sequential Feature Selection Algorithms". In: *In Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*. Springer-Verlag, 1995, pp. 1–7.

[AG17] Igor Anikin and Rinat Gazimov. "Privacy preserving DBSCAN clustering algorithm for vertically partitioned data in distributed systems". In: June 2017, pp. 1–4. DOI: 10.1109/SIBCON.2017.7998473.

[AGS02] K. Altisen, Gregor Gößler, and J. Sifakis. "Scheduler Modeling Based on the Controller Synthesis Paradigm". In: *Real-Time Systems* 23 (2002), pp. 55–84.

[And+05] Noriaki Ando et al. "RT-middleware: Distributed component middleware for RT (Robot technology)". In: Sept. 2005, pp. 3933–3938. DOI: 10.1109/IROS.2005.1545521.

[AS11] Seyed-Hosein Attarzadeh-Niaki and Ingo Sander. "Co-simulation of embedded systems in a heterogeneous MoC-based modeling framework". In: July 2011, pp. 238–247. DOI: 10.1109/SIES.2011.5953667.

[ASK08] Noriaki Ando, Takashi Suehiro, and Tetsuo Kotoku. "A Software Platform for Component Based RT-System Development: OpenRTM-Aist". In: *Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. SIMPAR '08. Venice, Italy: Springer-Verlag, 2008, pp. 87–98. ISBN: 9783540890751. DOI: 10.1007/978-3-540-89076-8_12. URL: https://doi.org/10.1007/978-3-540-89076-8_12.

[Åst06] Karl Johan Åström. "Feedback systems : an introduction for scientists and engineers". In: *Feedback systems : an introduction for scientists and engineers*. 2006. URL: http://people.duke.edu/~hpgavin/SystemID/References/Astrom-Feedback-2006.pdf.

[Bav17] Yazdan Bavafa-Toosi. "5 - Root locus". In: *Introduction to Linear Control Systems*. Ed. by Yazdan Bavafa-Toosi. Academic Press, 2017, pp. 377–467. ISBN: 978-0-12-812748-3. DOI: https://doi.org/10.1016/B978-0-12-812748-3.00005-7. URL: http://www.sciencedirect.com/science/article/pii/B9780128127483000057.

[BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. "Modeling Heterogeneous Real-Time Components in BIP". In: *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*. SEFM '06. USA: IEEE Computer Society, 2006, pp. 3–12. ISBN: 0769526780. DOI: 10.1109/SEFM.2006.27. URL: https://doi.org/10.1109/SEFM.2006.27.

[BD04] A. Banerjee and R.N. Dave. "Validating clusters using the Hopkins statistic". In: vol. 1. Aug. 2004, 149–153 vol.1. ISBN: 0-7803-8353-2. DOI: 10.1109/FUZZY.2004.1375706.

[BG11] Radhakisan Baheti and Helen Gill. "Cyber-physical Systems". In: 2011.

[BGB18] Hamza El Baccouri, Goulven Guillou, and Jean-Philippe Babau. "Robotic system testing with AMSA framework". In: *MODELS Workshops*. 2018.

[BH02] James Bezdek and R.J. Hathaway. "VAT: A tool for visual assessment of (cluster) tendency". In: vol. 3. Feb. 2002, pp. 2225–2230. ISBN: 0-7803-7278-6. DOI: 10.1109/IJCNN.2002.1007487.

[BK18] Štefan Bucz and Alena Kozakova. "Advanced Methods of PID Controller Tuning for Specified Performance". In: Sept. 2018. ISBN: 978-1-78923-700-9. DOI: 10.5772/intechopen.76069.

[BKS66] R. Bellman, R. Kalaba, and R. Sridhar. "SENSITIVITY ANALYSIS AND INVARIANT IMBEDDING". In: *Sensitivity Methods in Control Theory*. Ed. by L. RADANOVIC. Pergamon, 1966, pp. 36–45. ISBN: 978-1-4831-9822-4. DOI: https://doi.org/10.1016/B978-1-4831-9822-4.50006-6. URL: http://www.sciencedirect.com/science/article/pii/B9781483198224500066.

[Bod40] H. W. Bode. "Relations between attenuation and phase in feedback amplifier design". In: *The Bell System Technical Journal* 19.3 (1940), pp. 421–454.

[BR18] Nesrine Badache and Pascal Roques. "Capella to SysML Bridge: A Tooled-up Methodology for MBSE Interoperability". In: *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*. TOULOUSE, France, Jan. 2018. URL: https://hal.archives-ouvertes.fr/hal-01710529.

[Bre+12] D. Brezak et al. "A comparison of feed-forward and recurrent neural networks in time series forecasting". In: *2012 IEEE Conference on Computational Intelligence for Financial Engineering Economics (CIFEr)*. 2012, pp. 1–6.

[Bre01] Leo Breiman. "Random Forests". In: *Machine Learning* 45 (2001), pp. 5–32.

[Bro+05] A. Brooks et al. "Towards component-based robotics". In: Sept. 2005, pp. 163–168. DOI: 10.1109/IROS.2005.1545523.

[Bru+13] Herman Bruyninckx et al. "The BRICS Component Model: A Model-Based Development Paradigm for Complex Robotics Software Systems". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. SAC '13. Coimbra, Portugal: Association for Computing Machinery, 2013, pp. 1758–1764. ISBN: 9781450316569. DOI: 10.1145/2480362.2480693. URL: https://doi.org/10.1145/2480362.2480693.

[Bru01] H. Bruyninckx. "Open robot control software: the OROCOS project". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 3. 2001, 2523–2528 vol.3.

[BS05] Bruno Bouyssounouse and Joseph Sifakis. *Embedded Systems Design: The ARTIST Roadmap for Research and Development*. Jan. 2005. ISBN: 3-540-25107-3. DOI: 10.1007/b106761.

[BS10] Davide Brugali and Azamat Shakhimardanov. "Component-Based Robotic Engineering (Part II)". In: *Robotics & Automation Magazine, IEEE* 17 (Apr. 2010), pp. 100–112. DOI: 10.1109/MRA.2010.935798.

[BSK03] H. Bruyninckx, Peter Soetens, and B. Koninckx. "The real-time motion control core of the OROCOS project". In: vol. 2. Oct. 2003, 2766–2771 vol.2. DOI: 10.1109/ROBOT.2003.1242011.

[BSP12] Hari Bansal, Rajamayyoor Sharma, and Shreeraman Ponpathirkoottam. "PID Controller Tuning Techniques: A Review". In: 2 (Nov. 2012), pp. 168–176.

[Bui+13] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[CAR16] Olivier CARDIN. "Contribution to design, evaluation and implementation of cyber-physical production systems". Habilitation à diriger des recherches. Université de nantes, Dec. 2016. URL: https://tel.archives-ouvertes.fr/tel-01443318.

[CC12] In-Sung Choi and Jong-Suk Choi. "Leader-Follower formation control using PID controller". In: vol. 7507. Oct. 2012, pp. 625–634. DOI: 10.1007/978-3-642-33515-0_62.

[Cen13] María Victoria Cengarle. "Characteristics, capabilities, potential applications of Cyber-Physical Systems: a preliminary analysis". In: 2013.

[Cho+02] Pai Chou et al. "Control Generation for Embedded Systems Based on Composition of Modal Processes**This work was supported by PYI MIP-8858782, DARPA DAAH04-94-G-0272, and a Mentor fellowship." In: *Readings in Hardware/Software Co-Design*. Ed. by Giovanni [De Micheli], Rolf Ernst, and Wayne Wolf. Systems on Silicon. San Francisco: Morgan Kaufmann, 2002, pp. 350–357. DOI: https://doi.org/10.1016/B978-155860702-6/50031-4.

URL: http://www.sciencedirect.com/science/article/pii/B9781558607026500314.

[Cho+15] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[Cla+16] Anthony J. Clark et al. "An Evolutionary Approach to Discovering Execution Mode Boundaries for Adaptive Controllers". In: *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI)*. Athens, Greece, Dec. 2016, pp. 1–8. URL: http://www.cse.msu.edu/6.952708e-310mckinley/Pubs/files/Clark.Modes.ICES.2016.pdf.

[CSV96] Alberto Cavallo, Roberto Setola, and Francesco Vasca. *Using MATLAB, SIMULINK and Control System Toolbox: A Practice Approach*. USA: Prentice-Hall, Inc., 1996. ISBN: 0132610582.

[Dăn07] Nicolae Dăneţ. "Interval Analysis - A Powerfull Trend in Numerical Analysis". In: June 2007.

[DB00] Jennifer Dy and Carla Brodley. "Feature Subset Selection and Order Identification for Unsupervised Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning* (Oct. 2000).

[DCL08] Zuohua Ding, Zhenbang Chen, and Jing Liu. "A Rigorous Model of Service Component Architecture". In: *Electronic Notes in Theoretical Computer Science* 207 (Apr. 2008), pp. 33–48. DOI: 10.1016/j.entcs.2008.03.084.

[Del11] Laurent Delaigue. *Acceleo*. 2011. URL: http://www.eclipse.org/acceleo/.

[DHS00a] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. USA: Wiley-Interscience, 2000. ISBN: 0471056693.

[DHS00b] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. USA: Wiley-Interscience, 2000. ISBN: 0471056693.

[DL97] M. Dash and Huan Liu. "Feature selection for classification". In: *ntelligent Data Analysis* (1997), pp. 131–156.

[Doy78] J. Doyle. "Guaranteed margins for LQG regulators". In: *IEEE Transactions on Automatic Control* 23.4 (1978), pp. 756–757.

[Dre+02] Gérard Dreyfus et al. *Réseaux de neurones - Méthodologie et applications*. Ed. by Eyrolles. Jan. 2002. URL: https://hal.archives-ouvertes.fr/hal-01125016.

[Ebe67] L. L. Eberhardt. "Some Developments in 'Distance Sampling'". In: *Biometrics* 23.2 (1967), pp. 207–216. ISSN: 0006341X, 15410420. URL: http://www.jstor.org/stable/2528156.

[Ern03] Erik Ernst. "Separation of concerns". In: (Jan. 2003).

[Esh16] Sulaymon Eshkabilov. *MATLAB®/Simulink® Essentials: MATLAB®/Simulink® for Engineering Problem Solving and Numerical Analysis*. Nov. 2016. ISBN: 978-1-4834-5806-9.

[Est+96] Martin Ester et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[Eva48] W. R. Evans. "Graphical Analysis of Control Systems". In: *Transactions of the American Institute of Electrical Engineers* 67.1 (1948), pp. 547–551.

[Eva50] W. R. Evans. "Control System Synthesis by Root Locus Method". In: *Transactions of the American Institute of Electrical Engineers* 69.1 (1950), pp. 66–69.

[FG12] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. 1st. Addison-Wesley Professional, 2012. ISBN: 0321888944.

[Fie00] Roy Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. Jan. 2000.

[FM17] Michael Fop and Thomas Murphy. "Variable Selection Methods for Model-based Clustering". In: *Statistics Surveys* 12 (July 2017). DOI: 10.1214/18-SS119.

[FMS08] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: Systems Modeling Language*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN: 9780080558363.

[Fri19] Nejm Eddine Frigui. "Autonomic maintenance of high programmable optical access network". Theses. Ecole nationale supérieure Mines-Télécom Atlantique, Jan. 2019. URL: https://tel.archives-ouvertes.fr/tel-02176863.

[FS86] Andrew M. Fraser and Harry L. Swinney. "Independent coordinates for strange attractors from mutual information". In: *Phys. Rev. A* 33 (2 Feb. 1986), pp. 1134–1140. DOI: 10.1103/PhysRevA.33.1134. URL: https://link.aps.org/doi/10.1103/PhysRevA.33.1134.

[GB06] Sebastien Guerif and Younès Bennani. "Selection of Clusters Number and Features Subset during a Two-Levels Clustering Task". In: Jan. 2006, pp. 28–33.

[GE03a] Isabelle Guyon and André Elisseeff. "An Introduction of Variable and Feature Selection". In: *J. Machine Learning Research Special Issue on Variable and Feature Selection* 3 (Jan. 2003), pp. 1157–1182. DOI: 10.1162/153244303322753616.

[GE03b] Isabelle Guyon and André Elisseeff. "An Introduction to Variable and Feature Selection". In: *J. Mach. Learn. Res.* 3.null (Mar. 2003), pp. 1157–1182. ISSN: 1532-4435.

[Gei75] Seymour Geisser. "The Predictive Sample Reuse Method with Applications". In: *Journal of the American Statistical Association* 70.350 (1975), pp. 320–328. ISSN: 01621459. URL: http://www.jstor.org/stable/2285815.

[GHM10] M. K. Goldberg, M. Hayvanovych, and M. Magdon-Ismail. "Measuring Similarity between Sets of Overlapping Clusters". In: *2010 IEEE Second International Conference on Social Computing*. 2010, pp. 303–308.

[Gol+02] Frank Golatowski et al. "Framework for Validation, Test and Analysis of Real-time Scheduling Algorithms and Scheduler Implementations". In: (Aug. 2002).

[Gor+14] Andrew D. Gordon et al. "Probabilistic Programming". In: *Future of Software Engineering Proceedings*. FOSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 167–181. ISBN: 9781450328654. DOI: 10.1145/2593882.2593900. URL: https://doi.org/10.1145/2593882.2593900.

[Guy+06] Isabelle Guyon et al. "Feature Extraction - Foundations and Applications". In: *Feature Extraction*. 2006.

[HDR19] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. "On the Impact of the Activation function on Deep Neural Networks Training". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, Sept. 2019, pp. 2672–2680. URL: http://proceedings.mlr.press/v97/hayou19a.html.

[HG15] Zena M. Hira and Duncan Fyfe Gillies. "A Review of Feature Selection and Feature Extraction Methods Applied on Microarray Data". In: *Advances in Bioinformatics* 2015 (2015).

[HK14] Joachim Hartung and Guido Knapp. "Multivariate Multiple Regression". In: 2014.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. 3rd ed. Morgan Kaufmann Series in Data Management Systems. Amsterdam: Morgan Kaufmann, 2011. ISBN: 978-0-12-381479-1. URL: http://www.sciencedirect.com/science/book/9780123814791.

[HL89] David W. Hosmer and Stanley Lemeshow. "Applied Logistic Regression". In: 1989.

[HL95] Walter Hursch and Cristina Lopes. "Separation of Concerns". In: (Mar. 1995).

[HN09] Richard M. Heiberger and Erich Neuwirth. "Polynomial Regression". In: *R Through Excel: A Spreadsheet Interface for Statistics, Data Analysis, and Graphics*. New York, NY: Springer New York, 2009, pp. 269–284. ISBN: 978-1-4419-0052-4. DOI: 10.1007/978-1-4419-0052-4_11. URL: https://doi.org/10.1007/978-1-4419-0052-4_11.

[Hol65] P. Holgate. "Some New Tests of Randomness". In: *Journal of Ecology* 53.2 (1965), pp. 261–266. ISSN: 00220477, 13652745. URL: http://www.jstor.org/stable/2257973.

[HS54] BRIAN HOPKINS and J. G. SKELLAM. "A New Method for determining the Type of Distribution of Plant Individuals". In: *Annals of Botany* 18.70 (1954), pp. 213–227. ISSN: 03057364, 10958290. URL: http://www.jstor.org/stable/42907238.

[HYC08] Chenn-Jung Huang, Dian-Xiu Yang, and Yi-Ta Chuang. "Application of Wrapper Approach and Composite Classifier to the Stock Trend Prediction". In: *Expert Syst. Appl.* 34.4 (May 2008), pp. 2870–2878. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2007.05.035. URL: https://doi.org/10.1016/j.eswa.2007.05.035.

[Jac11] F Smuts Jacques. *Process control for practitioners : how to tune PID controllers and optimize control loops*. OptiControls Inc, 2011. ISBN: 0983843813.

[Jah18] Malte Jahn. "Artificial neural network regression models: Predicting GDP growth". In: 2018.

[Jai10] Anil Jain. "Data Clustering: 50 Years Beyond K-Means". In: *Pattern Recognition Letters* 31 (June 2010), pp. 651–666. DOI: 10.1016/j.patrec.2009.09.011.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. USA: Prentice-Hall, Inc., 1988. ISBN: 013022278X.

[Jea07] Julien Deantoni Jean-Philippe Babau. "Architectures logicielles pour les systèmes embarquéstemps réel". In: *Ecole d'été temps réel*, (Sept. 2007).

[Jel13] Mohieddine Jelali. "Assessment, Diagnosis and Improvement of Control Loop Performance". In: *Control Performance Management in Industrial Automation*. 2013. URL: http://dx.doi.org/10.1007/978-1-4471-4546-2.

[JKP94] George H. John, Ron Kohavi, and Karl Pfleger. "Irrelevant Features and the Subset Selection Problem". In: *ICML*. Morgan Kaufmann, 1994, pp. 121–129.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. "Data Clustering: A Review". In: *ACM Comput. Surv.* 31.3 (Sept. 1999), pp. 264–323. ISSN: 0360-0300. DOI: 10.1145/331499.331504. URL: https://doi.org/10.1145/331499.331504.

[Kas+18] Florian Kastner et al. “Exploring Deep Neural Networks for Regression Analysis”. In: 2018.

[KD19] Utkarsh Mahadeo Khaire and R. Dhanalakshmi. “Stability of feature selection algorithm: A review”. In: *Journal of King Saud University - Computer and Information Sciences* (2019). ISSN: 1319-1578.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN: 0262013193.

[KH04] N. Koenig and A. Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3.

[Kit78] Josef Kittler. “Feature set search algorithms " in Pattern Recognition and Signal Processing”. In: 1978.

[KJ97] Ron Kohavi and George H. John. “Wrappers for Feature Subset Selection”. In: *Artif. Intell.* 97.1–2 (Dec. 1997), pp. 273–324. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(97)00043-X. URL: https://doi.org/10.1016/S0004-3702(97)00043-X.

[KM14] Vipin Kumar and Sonajharia Minz. “Feature selection: A literature review”. In: *Smart Computing Review* 4 (Jan. 2014), pp. 211–229. DOI: 10.1145/2740070.2626320.

[KM17] Eamonn Keogh and Abdullah Mueen. “Curse of Dimensionality”. In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2017, pp. 314–315. ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_192. URL: https://doi.org/10.1007/978-1-4899-7687-1_192.

[Kow97] Adam Kowalczyk. “Estimates of Storage Capacity of Multilayer Perceptron with Threshold Logic Hidden Units”. In: *Neural Networks* 10.8 (1997), pp. 1417–1433. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00009-9. URL: http://www.sciencedirect.com/science/article/pii/S0893608097000099.

[Kur07] T. Kurfess. “Getting in tune with Ziegler-Nichols”. In: *Control Engineering* 54 (Feb. 2007).

[Lar+01] Edouard Laroche et al. “Web winding system robustness analysis via µ-analysis”. In: Feb. 2001, pp. 948–953. DOI: 10.1109/CCA.2001.973992.

[Lav+16] Erin Lavigne et al. “A Process for Evaluating Parametric Models for Mechanical Systems Simulation : the Case of a Sailboat”. In: 2016.

[Lav19] Emilien Lavigne. “AMSA, un framework dédié à la simulation des lois de contrôle pour des voiliers de compétition”. PhD thesis. UBO, 2019.

[Lee06] Edward Lee. "Cyber-Physical Systems - Are Computing Foundations Adequate?" In: (Jan. 2006).

[LGB18] Emilien Lavigne, Goulven Guillou, and Jean-Philippe Babau. "AVS, a model-based racing sailboat simulator: application to wind integration". In: *IFAC-PapersOnLine* 51.10 (2018). 3rd IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control CESCIT 2018, pp. 88–94. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.06.242. URL: http://www.sciencedirect.com/science/article/pii/S2405896318305627.

[LL02] Jie Liu and Edward A. Lee. "A Component-Based Approach to Modeling and Simulating Mixed-Signal and Hybrid Systems". In: *ACM Trans. Model. Comput. Simul.* 12.4 (Oct. 2002), pp. 343–368. ISSN: 1049-3301. DOI: 10.1145/643120.643125. URL: https://doi.org/10.1145/643120.643125.

[Lu+16] Chang Lu et al. "Data Mining Applied to Oil Well Using K-Means and DBSCAN". In: Nov. 2016, pp. 37–40. DOI: 10.1109/CCBD.2016.018.

[LWG11] Ulrike Luxburg, Robert Williamson, and Isabelle Guyon. "Clustering: Science or Art?" In: *J Mach Learn Res* 27 (July 2011).

[LY05] Huan Liu and Lei Yu. "Yu, L.: Toward Integrating Feature Selection Algorithm for Classification and Clustering. IEEE Transaction on Knowledge and Data Engineering 17(4), 491-502". In: *IEEE Transactions on Knowledge and Data Engineering - TKDE* 17 (Apr. 2005), pp. 491–502. DOI: 10.1109/TKDE.2005.66.

[Mac67] James B. MacQueen. "Some methods for classification and analysis of multivariate observations". In: 1967.

[Mar97] Anu Maria. "Introduction to Modeling and Simulation". In: *Proceedings of the 29th Conference on Winter Simulation*. WSC '97. Atlanta, Georgia, USA: IEEE Computer Society, 1997, pp. 7–13. ISBN: 078034278X. DOI: 10.1145/268437.268440. URL: https://doi.org/10.1145/268437.268440.

[Mat+19] R. Matinnejad et al. "Test Generation and Test Prioritization for Simulink Models with Dynamic Behavior". In: *IEEE Transactions on Software Engineering* 45.9 (2019), pp. 919–944.

[McK10] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[MG63] T. Marill and D. Green. "On the effectiveness of receptors in recognition systems". In: *IEEE Transactions on Information Theory* 9.1 (Jan. 1963), pp. 11–17. ISSN: 1557-9654. DOI: 10.1109/TIT.1963.1057810.

[ML10] Jelena Milojkovic and Vanco Litovski. "ANN versus Grey theory based forecasting methods implemented on short time series". In: *10th Symposium on Neural Network Applications in Electrical Engineering, NEUREL-2010 - Proceedings* (Sept. 2010). DOI: 10.1109/NEUREL.2010.5644094.

[MMR96] Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka. *Elements of Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1996. ISBN: 0262133288.

[MNB17] Reza Matinnejad, Shiva Nejati, and Lionel C. Briand. "Automated Testing of Hybrid Simulink/Stateflow Controllers: Industrial Case Studies". In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2017. Paderborn, Germany: Association for Computing Machinery, 2017, pp. 938–943. ISBN: 9781450351058. DOI: 10.1145/3106237.3117770. URL: https://doi.org/10.1145/3106237.3117770.

[Mon14] László Monostori. "Cyber-physical production systems: Roots, expectations and R&D challenges". In: *Procedia CIRP* 17 (2014), pp. 9–13. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2014.03.115. URL: http://www.sciencedirect.com/science/article/pii/S2212827114003497.

[MP17] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 2017. ISBN: 0262534770.

[MPH09] Laurens van der Maaten, Eric O. Postma, and Jaap van den Herik. "Dimensionality Reduction: A Comparative Review". In: 2009.

[Mud06] F. Mudry. *Ajustage des Paramètres d'un Régulateur PID*. 2006. URL: http://fltsi.fr/tsi/archives_16_17/tsi/tsi2/TP_TSI2/TP4%5C%202015/Asservissement%5C%20reglages%5C%20PID/Ajustage%5C%20PID.pdf.

[Mur03] Fionn Murtagh. "Multilayer perceptrons and regression for classification". In: 2003.

[NBP13] Juan F. Navas, Jean-Philippe Babau, and Jacques Pulou. "Reconciling Run-Time Evolution and Resource-Constrained Embedded Systems through a Component-Based Development Framework". In: *Sci. Comput. Program.* 78.8 (Aug. 2013), pp. 1073–1098. ISSN: 0167-6423. DOI: 10.1016/j.scico.2012.08.004. URL: https://doi.org/10.1016/j.scico.2012.08.004.

[Nej+19] Shiva Nejati et al. "Evaluating Model Testing and Model Checking for Finding Requirements Violations in Simulink Models". In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Tallinn, Estonia: Association for Computing Machinery, 2019, pp. 1015–1025. ISBN: 9781450355728. DOI: 10.1145/3338906.3340444. URL: https://doi.org/10.1145/3338906.3340444.

[Nis00] Norman S. Nise. "Control Systems Engineering". In: *Control Systems Engineering*. 2000, p. 992.

[Nyq32] H. Nyquist. "Regeneration theory". In: *The Bell System Technical Journal* 11.1 (1932), pp. 126–147.

[Oga01] Katsuhiko Ogata. *Modern Control Engineering*. 4th. USA: Prentice Hall PTR, 2001. ISBN: 0130609072.

[Oli17] David J. Olive. "Multivariate Linear Regression". In: 2017.

[Pas18] Michal Pasternak. *MDE tools challenge*. 2018. URL: https://mdetools.github.io/mdetools18/committees.html.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Qui+09] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: vol. 3. Jan. 2009.

[Raj+10] Ragunathan Rajkumar et al. "44.1 Cyber-Physical Systems: The Next Computing Revolution". In: Jan. 2010, pp. 731–736. DOI: 10.1145/1837274.1837461.

[Ras18] Sebastian Raschka. "MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack". In: *The Journal of Open Source Software* 3.24 (Apr. 2018). DOI: 10.21105/joss.00638. URL: http://joss.theoj.org/papers/10.21105/joss.00638.

[RF90] Maurice Rivoire and Jean-Louis Ferrier. *Cours d'Automatique - TOME 3 - Commande par calculateur, Identification*. EYROLLES, 1990, p. 223. URL: https://hal.archives-ouvertes.fr/hal-00839070.

[Rom+10] Daniel Romero et al. "Integration of Heterogeneous Context Resources in Ubiquitous Environments". In: Sept. 2010, pp. 123–126. DOI: 10.1109/SEAA.2010.27.

[Ros58] Frank F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6 (1958), pp. 386–408.

[Rub20] Daniel Rubinfeld. "Reference Guide on Multiple Regression". In: (June 2020).

[San+17] Jose Sanchez Lopez et al. "A Multi Layered Component Based Approach for the Development of Aerial Robotic Systems: The Aerostack Framework". In: *Journal of Intelligent & Robotic Systems* 88 (May 2017). DOI: 10.1007/s10846-017-0551-4.

[SC08] Ingo Steinwart and Andreas Christmann. "Support Vector Machines". In: *Information science and statistics*. 2008.

[Sen+14] Rajat Sen et al. "Comparison Between Three Tuning Methods of PID Control for High Precision Positioning Stage". In: *MAPAN* 30 (Mar. 2014), pp. 65–70. DOI: 10.1007/s12647-014-0123-z.

[SIL07] Yvan Saeys, Iñaki Inza, and Pedro Larranaga. "A review of feature selection techniques in bioinformatics". In: *Bioinformatics (Oxford, England)* 23 (Nov. 2007), pp. 2507–17. DOI: 10.1093/bioinformatics/btm344.

[Sin+04a] F. Singhoff et al. "Cheddar: A Flexible Real Time Scheduling Framework". In: *Proceedings of the 2004 Annual ACM SIGAda International Conference on Ada: The Engineering of Correct and Reliable Software for Real-Time & Distributed Systems Using Ada and Related Technologies*. SIGAda '04. Atlanta, Georgia, USA: Association for Computing Machinery, 2004, pp. 1–8. ISBN: 1581139063. DOI: 10.1145/1032297.1032298. URL: https://doi.org/10.1145/1032297.1032298.

[Sin+04b] F. Singhoff et al. "Cheddar: A Flexible Real Time Scheduling Framework". In: *Ada Lett.* XXIV.4 (Nov. 2004), pp. 1–8. ISSN: 1094-3641. DOI: 10.1145/1046191.1032298. URL: https://doi.org/10.1145/1046191.1032298.

[SSH07] P. Seidel, A. Seidel, and O. Herbarth. "Multilayer perceptron tumour diagnosis based on chromatography analysis of urinary nucleosides". In: *Neural Networks* 20.5 (2007), pp. 646–651. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2006.12.004. URL: http://www.sciencedirect.com/science/article/pii/S0893608006002802.

[STO77] M. STONE. "Asymptotics for and against cross-validation". In: *Biometrika* 64.1 (Apr. 1977), pp. 29–35. ISSN: 0006-3444. DOI: 10.1093/biomet/64.1.29. eprint: https://academic.oup.com/biomet/article-pdf/64/1/29/1382990/64-1-29.pdf. URL: https://doi.org/10.1093/biomet/64.1.29.

[SVK97] David B. Stewart, Richard A. Volpe, and Pradeep K. Khosla. "Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects". In: *IEEE Trans. Softw. Eng.* 23.12 (Dec. 1997), pp. 759–776. ISSN: 0098-5589. DOI: 10.1109/32.637390. URL: https://doi.org/10.1109/32.637390.

[SW99] C. Schlegel and R. Worz. "The software framework SMARTSOFT for implementing sensorimotor systems". In: *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289)*. Vol. 3. 1999, 1610–1616 vol.3.

[Szt07] J. Sztipanovits. "Composition of Cyber-Physical Systems". In: *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*. 2007, pp. 3–6.

[Szy02] Clemens Szyperski. *Component Software : Beyond Object-Oriented Programming*. Jan. 2002. ISBN: 0-201-745572-0.

[TAL14] Jiliang Tang, Salem Alelyani, and Huan Liu. "Feature Selection for Classification: A Review". In: *Data Classification: Algorithms and Applications*. 2014.

[WH99] Kok Kiong T.Qing-Guo W. and Chang Chieh H. "Classical Designs". In: *Advances in PID Control*. 1999, pp. 19–34. URL: https://doi.org/10.1007/978-1-4471-0861-0_2.

[Whi71] A. W. Whitney. "A Direct Method of Nonparametric Measurement Selection". In: *IEEE Trans. Comput.* 20.9 (Sept. 1971), pp. 1100–1103. ISSN: 0018-9340. DOI: 10.1109/T-C.1971.223410. URL: https://doi.org/10.1109/T-C.1971.223410.

[WL98] Bernard Widrow and Michael A. Lehr. "Perceptrons, Adalines, and Backpropagation". In: *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, pp. 719–724. ISBN: 0262511029.

[Yu06] Cheng-Ching Yu. "Introduction". In: *Autotuning of PID Controllers: A Relay Feedback Approach*. 2006, pp. 1–8. URL: https://doi.org/10.1007/1-84628-037-0_1.

[ZF81] G. Zames and B. A. Francis. "A new approach to classical frequency methods: Feedback and minimax sensitivity". In: *1981 20th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes*. 1981, pp. 867–874.

[Zhe+17] X. Zheng et al. "Perceptions on the State of the Art in Verification and Validation in Cyber-Physical Systems". In: *IEEE Systems Journal* 11.4 (2017), pp. 2614–2627.

[ZM97] Mohamed Zaït and Hammou Messatfa. "A comparative study of clustering methods". In: *Future Generation Computer Systems* 13.2 (1997). Data Mining, pp. 149–159. ISSN: 0167-739X. DOI: https://doi.org/10.1016/S0167-739X(97)00018-6. URL: http://www.sciencedirect.com/science/article/pii/S0167739X97000186.

[ZMK19] Bernard P. Zeigler, Alexandre Muzy, and Ernesto Kofman. "Chapter 2 - Framework for Modeling and Simulation". In: *Theory of Modeling and Simulation (Third Edition)*. Ed. by Bernard P. Zeigler, Alexandre Muzy, and Ernesto Kofman. Third Edition. Academic Press, 2019, pp. 27–41. ISBN: 978-0-12-813370-5. DOI: https://doi.org/10.1016/B978-0-12-813370-5.00010-9. URL: http://www.sciencedirect.com/science/article/pii/B9780128133705000109.

[ZN93a] J. G. Ziegler and N. B. Nichols. "Optimum Settings for Automatic Controllers". In: *Journal of Dynamic Systems, Measurement, and Control* 115.2B (June 1993), pp. 220–222. ISSN: 0022-0434. DOI: 10.1115/1.2899060. eprint: https://asmedigitalcollection.asme.org/dynamicsystems/article-pdf/115/2B/220/5546571/220_1.pdf. URL: https://doi.org/10.1115/1.2899060.

[ZN93b] J. G. Ziegler and N. B. Nichols. "Optimum Settings for Automatic Controllers". In: *Journal of Dynamic Systems, Measurement, and Control* 115.2B (June 1993), pp. 220–222. URL: https://doi.org/10.1115/1.2899060.

[ZPH98] Peter Zhang, Eddy Patuwo, and Michael Hu. "Forecasting With Artificial Neural Networks: The State of the Art". In: *International Journal of Forecasting* 14 (Mar. 1998), pp. 35–62. DOI: 10.1016/S0169-2070(97)00044-7.

[ZPK00] Bernard Zeigler, Herbert PrÃhofer, and Tag Gon Kim. "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems". In: 2 (Jan. 2000).

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UBO Université de Bretagne Occidentale

Titre : Automatisation de la mise au point de lois de contrôle de systèmes cyber-physiques évoluant en environnement incertain

Mots clés : réglage des contrôleurs, simulation, réduction de dimensionnalité, clustering, régression

Résumé : Le comportement des systèmes cyber-physiques évoluant dans un environnement incertain dépend étroitement des évolutions de ce dernier. Dans ce contexte, la mise au point du contrôleur associé à de tels systèmes est un processus complexe et exigeant en raison de la multiplicité des contextes à considérer et du fait qu'un réglage de contrôle correct (en termes de qualité de contrôle) pour un contexte spécifique peut ne pas être adapté à un autre contexte. Il est alors nécessaire de trouver des valeurs adéquates des paramètres de la loi de contrôle pour chaque contexte. Néanmoins, il n'est pas approprié de changer le réglage suite à chaque changement du contexte (changement de l'environnement, dans l'état des capteurs ou des actionneurs). Il est donc impératif de trouver des réglages qui conviennent pour plusieurs contextes.

Title : Automation of the development of control laws for cyber-physical systems evolving in an uncertain environment

Keywords : controller tuning, simulation, dimensionality reduction, clustering, regression

Abstract : The evolution of cyber-physical systems evolving in an uncertain environment depends closely on the evolution of the latter. In this context, the development of the controller associated with such systems is a complex and demanding process due to the multiplicity of contexts to be considered and the fact that a correct control setting (in terms of control quality) for a specific context may not be adapted to another context. It is then necessary to find adequate values of the parameters of the control law for each context. However, it is not appropriate to change the setting after each change of the context (change of the environment, in the state of the sensors or actuators). It is therefore imperative to find settings that are suitable for several contexts.