UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and the Environment
Aeronautics and Computational Engineering

Autonomous Multi-agent Reconfigurable Control Systems

by

Badril Abu Bakar

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

March 17, 2013

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND THE ENVIRONMENT
Aeronautics and Computational Engineering

Doctor of Philosophy

AUTONOMOUS MULTI-AGENT RECONFIGURABLE CONTROL SYSTEMS

by Badril Abu Bakar

This thesis is an investigation of methods and architectures for autonomous multi-agent reconfigurable controllers. As part of the analysis two components are looked at: the fault detection and diagnosis (FDD) component and the controller reconfiguration (CR) component. The FDD component detects and diagnoses faults. The CR component on the other hand, adapts or changes the control architecture to accommodate the fault. The problem is to synchronize or integrate these two components in the overall structure of a control system. A novel approach is proposed. A multiagent architecture is used to interface between the two components. This method allows the system to be viewed as a modular structure. Three types of agent are defined. A planner agent A_p, a monitor agent A_m and a control agent A_c. The monitor agent takes the role of the FDD component. The planner and control agents on the other hand take the roles of CR component.

The planner decides which controller to use and passes it on to A_c. It also decides on the parameter settings of the system and changes it accordingly. It belongs to the *reactive agent* category. The planner agent's internal architecture maps its sensor data directly to actions using a pre-set rule based conditional logic. It was decided that this architecture would reduce the overall complexity of the system.

The monitor agent A_m belongs to the *learning agent* category. It uses an algorithm called *adaptive resonance theory neural network* or ART-NN to autonomously categorize system faults. A_m then informs the other agents of the fault status. ART-NN was chosen due to the fact that it does not need to be trained with sample data and learns to categorize data patterns on the fly. This allows A_m to detect unmodelled system faults.

The control agent A_c also belongs to the learning agent category. It uses a *multiagent reinforcement learning algorithm* to learn a controller for the system at hand. Once a suitable controller has been learnt, the parameters of the controller are passed to A_p for it to be stored in its memory and learning is terminated. During control execution mode, controller parameters are sent to A_c from A_p.

The novel approach is demonstrated on a case study. Our laboratory-built 4-wheeled skid-steering vehicle complete with sensors is designed as a way of demonstration. Several faults are simulated and the response of the demo system is analyzed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automatic controllers provide the means of manipulating a *plant*'s behaviour in a desired manner. Here, "plant" is used as a generic term for a device or process that is capable of being controlled. A properly designed controller prevents the plant from operating in a dangerous and unstable mode. A poorly designed controller on the other hand, will result in significant damage to the plant and/or injury to people. Growing demand for safety, reliability and survivability in automated systems have prompted for a controller that can absorb these imperfections in design and conditions. Conventional designs have become inadequate to cater for these complex systems.

The design of such controllers can be approached in many ways. Fault tolerant control systems (FTCS), as this field is termed have been subject to intense research. Research done in this field have been motivated by aircraft flight control designs and the process industry. Adaptive control and robust control are just two of the many approaches that have been developed.

Fault tolerant control systems have been a popular topic of research for over 30 years. One of the earliest works in this field was recorded in 1978 (Chizeck and Willsky, 1978),(Chizeck et al., 1983). These early results extended jump linear quadratic gaussian (JLQG) control problems to include among others random jump cost functions. The controller would abruptly and randomly change parameters in the case of predefined faults. Since then, a number of approaches have been introduced such as *robust control* (Hess and Jung, 1989; Qian and Stengel, 2005; Hsieh, 2002), *sliding mode control* (Shin et al., 2005; Demirci and Keresteciogl u, 2004) and *switching control* (Bajpai et al., 2002; Jin and Zhang, 2006). Other forms of approaches include *adaptive control* (Mason et al., 1987; Bolourchi and Hess, 1992; Bodson and Groszkiewicz, 1997) and *intelligent control* (Ferrari and Stengel, 2004; Polycarpou and Helmicki, 1995; Farrell et al., 1993).

1.1 Problem Statement

In the literature, *reconfigurable control systems* fall under the scheme of *active fault tolerant control systems* (AFTCS) (Moerder et al., 1989). They are distinguished from their counterpart, *passive fault tolerant control systems* (PFTCS) through the integration of both the fault detection and diagnosis component (FDD) and the controller reconfiguration component (CR) in the overall structure (Zhang and Jiang, 2008).

Reconfigurable control systems react to system component failures actively. They do so by reconfiguring control actions so that the system's performance and stability can be maintained. There are two key components to reconfigurable control systems; Fault Detection and Diagnosis (FDD) and Controller Reconfiguration (CR). Traditionally, research on these two components have been carried out separately due to the complexity of the problem (Wu, 1997; Zhang and Jiang, 2008; Bodson and Groszkiewicz, 1997; Bajpai et al., 2002). Integration of these two components in the overall structure of the system is an active research topic (Aberkane et al., 2008; Wu, 1997; Balle et al., 1998; Zhang and Jiang, 2003).

In this thesis, the design philosophy is to use a multi-agent architecture that allows for both FDD and CR components or stages to be designed in the same framework.

The problem this work investigates can be summarized as follows. Traditionally the FDD component detects that a fault has occurred and further diagnoses the situation. The CR component adapts or changes the control architecture to accommodate the fault. The problem considered in this thesis is to synchronize or integrate these two components in the overall structure of a control system using innovative agent approach, reinforcement learning, neural networks, game theory and wavelets. The results are then demonstrated on a model of a laboratory built off-road rover.

1.2 Objective

There are two main objectives in this research project:

1. To develop a new theory for autonomous reconfigurable control systems that is practical and robust in applications.
2. To demonstrate the new concepts and results on a simulation of the demonstration rover to reduce experimental time.

In order to achieve the first objective, the investigation is broken down into two stages:

- Investigation into the fault detection and diagnosis.

- Investigation into reconfiguration.

Novel features of possible techniques will arise from FDD/CR being integrated into the multi-agent architecture. The second objective requires that the developed control system be applied on a model of a vehicle. The control demonstration of an autonomous ground vehicle or simply called rover for the rest of this work will be simulated.

This work on reconfigurable control systems will be presented in the following manner. Chapter 2 reviews the state of the art. The definitions of the concepts used in this work are presented in Chapter 3. Our approach to reconfigurable control systems is introduced in Chapter 4. This is followed by a formal representation of the methodology in the software package sEnglish. A case study to demonstrate our approach is given in Chapter 6. Summary and suggestions for future research areas in this field are presented in Chapter 7.

Chapter 2

Literature Review

In this chapter a review of significant contributions is presented on fault tolerant control. These key papers serve as the basis for the theoretical and experimental advances outlined in this work. Several historically important papers are identified and recent papers with direct relevance to our goal of creating an autonomous reconfigurable control algorithm are discussed. Literature on FDD and CR will be described in two separate sections due to the significant separation of these research areas. The chapter ends with a conclusion of the literature review.

2.1 Basic Concepts

Reconfigurable Systems

In a broad sense a system is said to have a reconfigurable control architecture if it is able to recover from a faulty state by altering its operation. This means either reconfiguring its controller architecture or its parameters. It is debatable whether a system that does not change sensor and actuator allocations while keeping its controller structure and adapting its control gains, to accommodate some faulty hardware conditions, can be called reconfigurable. In our research we suggest that systems that do merely control gain adjustments should be called robust or adaptive systems. Another possibility is to call such systems weakly reconfigurable. This could be in the form of changing controller gains. One has to accept however, that the system could be operating in a possibly degraded mode due to physical constraints (Blanke et al., 1997).

Definition of Faults

In our study faults will mean a failure of sensors, actuators or other internal mechanical malfunctions and also the breakdown of communication lines that are part of the control architecture. Software errors due to bugs in the real time controller code will not be

considered faults. A basic assumption is made that the control system is implemented as intended. However, partial breakdown of computing hardware for some subsystem, which causes problems to the rest of the controller software, will be called a fault.

The following sections depict some of the various existing approaches to the problem of reconfigurable control systems.

2.2 Fault Detection and Diagnosis

Fault detection and identification is the process of recognizing anomalies in a plant's behaviour (Chiang et al., 2001). FDD systems implement the following tasks:

1. Fault detection is the discovery that something has gone wrong in the monitored system.
2. Fault isolation is the determination of the exact location of the fault.
3. Fault identification is the determination of the magnitude of the fault.

The FDD process can be better explained on an example. Consider a man driving a car along a straight road, when suddenly a loud sound is heard and the car starts veering off the right. The event is abnormal to the man driving and therefore an alarm is triggered in his head saying that a fault has occurred. This is the detection step of and FDD process. Upon recognizing the fault, the man pulls the car over to the side of the road and checks to see what is wrong. After visual inspection of the car, he realizes that the right front tyre has a flat. The exact location of the fault has been determined. This is the isolation step of the FDD process. Upon closer inspection, the man finds that the tyre has burst creating a big tear on the side of the tyre. The magnitude of the fault is determined. This is the identification step of an FDD process.

The terminology used in the literature to describe the fault detection and diagnosis scheme is not standardized. The term FDI is used to refer to Fault Detection and Isolation in some works of the literature, and Fault Detection and Identification in others. This prompted Zhang and Jiang (2008) to use FDD to indicate that the fault identification function is added to the fault detection and isolation scheme. We use the term fault detection and diagnosis in our work to emphasize the importance of viewing the identification function as an integral part of the overall diagnostic scheme which encompasses fault detection, isolation and identification.

Traditional methods are based on the use of signal processing techniques and/or parallel redundancy. They include frequency spectrum analysis, limit checking, special sensors, parallel redundancy and fault dictionary approach (Patton, 1993). Modern day

approaches on the other hand are generally grouped into three categories (Venkatasubramanian et al., 2003c; Isermann, 2005; Patton, 1993). It is our view however, that FDD can essentially be split into only two categories:

- Model-based approach
- Model-free approach

2.2.1 Model-based approach

Model-based schemes use a mathematical model to provide analytical information about the physical process. These schemes are further broken down into quantitative and qualitative models.

Quantitative model-based approaches detect, isolate and characterise a fault of a system. They compare the system's available measurements with a priori information represented by the system's mathematical model. Faults are detected by setting a (fixed or variable) threshold on a residual quantity generated from the difference between real measurements and estimates of these measurements using a mathematical model. Figure 2.1 depicts a general structure of a quantitative model whereas Figure 2.2 shows the general structure of a residual generator.

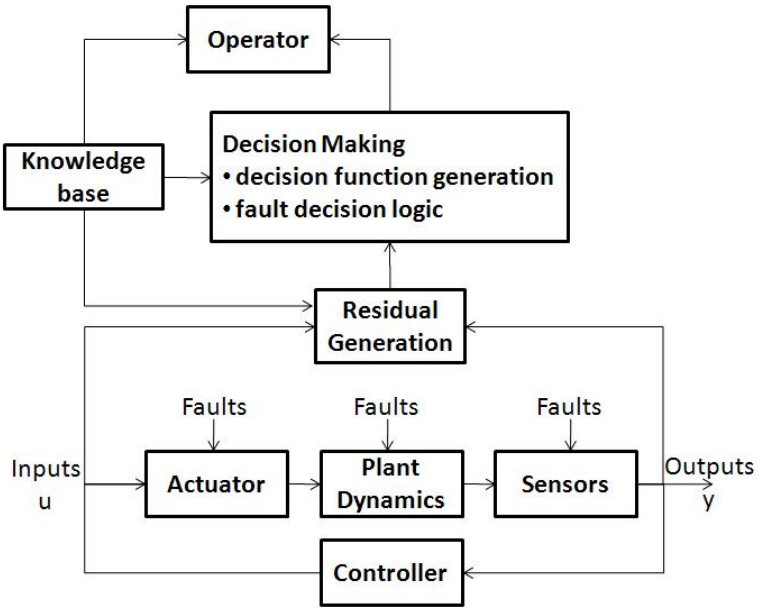


FIGURE 2.1: Quantitative model-based fault diagnosis (Patton, 1993)

In the qualitative model-based approach, the model is expressed in terms of event abstraction/description functions centered around different units in a process (Venkatasubramanian et al., 2003a). Figure 2.3 shows the forms of qualitative models.

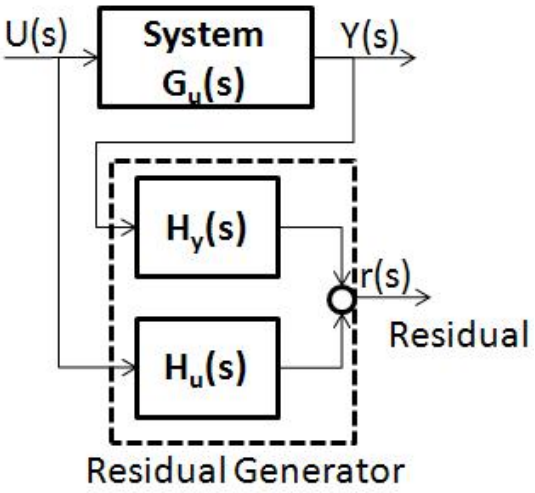


FIGURE 2.2: General structure of a residual generator (Patton, 1993)

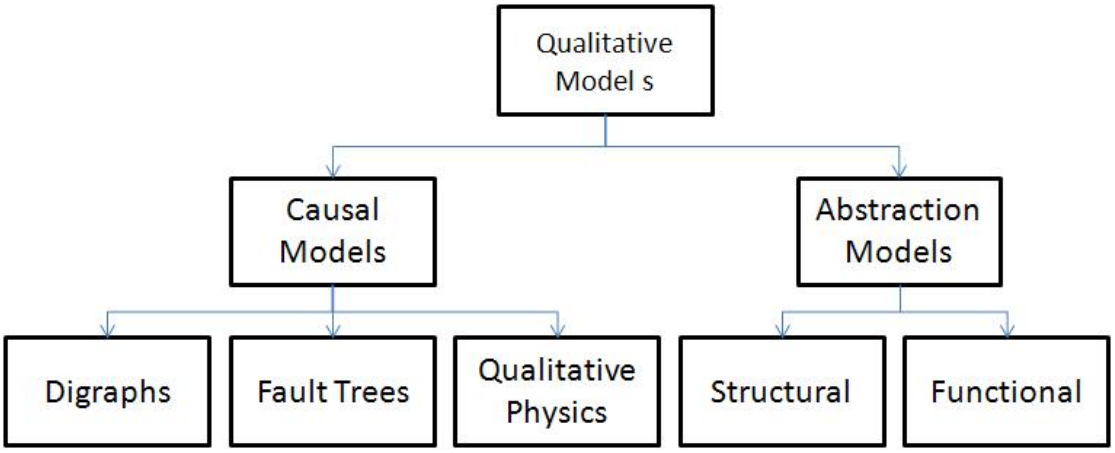


FIGURE 2.3: Forms of qualitative model (Venkatasubramanian et al., 2003a)

The qualitative and quantitative approaches are usually combined to offset their individual weaknesses. Typical methods using this approach are parameter estimation, observers, parity equations, and Kalman filters. Research on FDI methods following the model-based structure has been extensively carried out (Frank, 1995; Yang and Stoustrup, 2000; Yin, 1998; Chang et al., 1994; Koppen-Seliger et al., 1995; Patton et al., 1989, 1995; Gomm et al., 1992; Borutzky, 2009; Samantaray et al., 2005; Dvorak et al., 1991; Trave-Massuyes et al., 2006; Wei et al., 2009; Gentil et al., 2004; Steele and Leitch, 1995; Vinson and Ungar, 1995; Gertler, 2005b,a; Calado et al., 2006; Abdelwahed and Kandasamy, 2007; Deshpande and Patwardhan, 2008).

In Frank (1995), the adaptive threshold logic test based upon fuzzy modelling of the process and fuzzy decision making was the focus of the research. The residual evaluation was formulated as a set of fuzzy rules.

Koppen-Seliger et al. (1995) introduced a neural network based on a residual evaluation

concept for fault diagnosis. The idea was to emphasize the evaluation part of a diagnostic concept in contrast to most existing schemes at the time. A restricted Coulomb Energy Neural Network (RCE) was employed to classify residuals coming from a standard parameter estimation procedure. The classification was aimed at the detection and isolation of different faults in the process under supervision.

A similar approach to Frank (1995) was taken by Calado et al. (2006). A computer-assisted fault detection and isolation (FDI) scheme was presented that was coupled with a fuzzy qualitative simulation algorithm used for fault detection purposes and a hierarchical structure of fuzzy neural networks used to perform the fault isolation task. The DAMADICS benchmark actuator system was used as a test bed for the FDI system. Their proposed FDI system was tested for faults associated with the DAMADICS benchmark problem. It was observed that the FDI system was not able to cope with incipient faulty scenarios. The incipient faults were simulated according to the DAMADICS benchmark rule which imposes a very low development speed. In some situations the controller action masked the fault. In other cases the fault effects in the system behaviour were similar to the noise effects, making the faults undistinguishable from the normal system operation. In the case of double simultaneous abrupt faults, the FDI system was able to detect all faults. Some unsuccessful results to isolate the double faults were obtained. Furthermore, it was observed that if the number of measurement variables used as inputs of the FDI system increased, the number of unsuccessful results tended to decrease and eventually be eliminated.

Yin (1998) designed a robust fault isolation method applicable to the directional residual approach by using decision theory. The cost, the loss, and the risk of misclassification were analysed. To achieve a balance between optimality and robustness, a minimax method was proposed which minimized the maximum expected loss from misclassification. An isolation procedure was designed for the diagnostic residuals having normal distribution. A simulated distillation example was given to illustrate the implementation of this technique.

A novel approach for the synthesis of robust reconfigurable control for LTI systems and a class of nonlinear control systems with parametric and additive faults, as well as uncertainties generated by FDI algorithms, has been proposed in a unified framework (Yang and Stoustrup, 2000). The $\mathcal{H}_\infty$ control and μ synthesis techniques could be employed efficiently for this control synthesis by following the model-matching strategy. It did not investigate the integration of controller reconfiguration (CR) with the FDI scheme.

A performance comparison of a nonlinear Radial Basis Function Network-based (RBFN) and linear Auto Regressive (AR) model-based General Parameter (GP) methods in a fault detection application was done in Dote et al. (2001). The fault detection of automatic transmission gears using acoustic data analysis was considered. The proposed

GP-RBFN and GP-AR modelling scheme is computationally efficient and, therefore highly practical for low cost real-time applications. It should be noted, however, that the described GP methods operate only with correlated time series.

Some of the disadvantages of the model-based approach were described in Venkatasubramanian et al. (2003c). One of the drawbacks of this approach was that, for general nonlinear models, the effectiveness was reduced due to poor linear approximations of the model. In practice, severe modelling uncertainties come in the form of multiplicative uncertainties. The disturbance matrix of this approach in most cases only included additive uncertainties. A further disadvantage of this approach was that if a fault was not specifically modelled, there was no guarantee that residuals would be able to detect it.

A causal model-based diagnosis combining FDI and AI was investigated in Gentil et al. (2004). The authors presented a model-based diagnostic method designed in the context of process supervision. It was inspired by both artificial intelligence and control theory. AI contributed tools for qualitative modelling, including causal modelling, whose aim was to split a complex process into elementary submodels. Control theory, within the framework of fault detection and isolation (FDI), provided numerical models for generating and testing residuals, and for taking into account inaccuracies in the model, unknown disturbances and noise. Consistency-based reasoning provided a logical foundation for diagnostic reasoning and clarified fundamental assumptions, such as single fault and exoneration. The diagnostic method presented in their paper benefited from the advantages of all these approaches. Causal modelling enabled the method to focus on sufficient relations for fault isolation, which avoided combinatorial explosion. Moreover, it allowed the model to be modified easily without changing any aspect of the diagnostic algorithm. The numerical submodels that were used to detect inconsistency benefited from the precise quantitative analysis of the FDI approach. This method served more as a diagnostic tool for a human operator than for an autonomous system.

A special issue on model-based approaches to FDI was published in 2004 to address the problem of bridging the community which based its solution approaches on engineering disciplines such as control theory and statistical decision making and the community which based their solution approaches in the fields of computer science and artificial intelligence (Biswas et al., 2004; Cordier et al., 2004; Pulido and Gonzalez, 2004; Hofbaur and Williams, 2004).

A method for finer fault isolation or localization in the model-based fault detection and isolation (FDI) paradigm was developed using parallel computed bond graph models (Samantaray et al., 2005). The model-based fault detection using analytical redundancy (ARR) was extended with a novel multi-tier fault isolation/localization technique, which used a single hypothesized fault parameter estimation from ARR. The proposed solution was to estimate parameters of a subspace from the ARR by assuming a single-fault

hypothesis and then to incorporate the estimated values in separate models to run parallel with the plant during the fault. Thereafter, comparison of model behaviors led to localization of the faulty parameters.

A new approach to FDI for nonlinear systems was presented in Uppala et al. (2006). It only dealt with the FDI and did not attempt to solve the reconfiguration problem. It employed neuro-fuzzy multiple modelling together with robust optimal de-coupling of observers. The author called this new method 'Neuro-Fuzzy and De-coupling Fault Diagnosis Scheme' (NFDFDS). This method exploited the advantages of neural networks and fuzzy logic due to its approximation and reasoning capabilities. It was therefore suitable to be used as a modelling tool of nonlinear systems. Due to the lack of ability for linearisation to represent a good model for processes with strongly nonlinear behaviour coupled with the fact that the nonlinear systems dynamics have to be known with sufficient confidence in order for the nonlinear observer approach to be used, the author's motivation was to explore other methods, i.e. neuro-fuzzy (NF) which is known to overcome some of the problems faced by the model-based techniques. The proposed FDI scheme was made from a matrix of NF based decoupling observers where the rows were the number of fault scenarios considered and the columns were the number of operational points. It generated a residual set which was used by the diagnostics logic unit to determine the location and nature of faults. A linear set of fuzzy fusion sub-observers, each one corresponding to a different operating point of the process was comprised of nonlinear systems, i.e 'Fault Diagnosis Observers'. Through fuzzy-fusion their outputs were summed to calculate the output estimates. The 'NFDFDS' scheme was essentially a combination of a set of fuzzy observers together with the NF multiple model and diagnostic logic. A comparative study was done to evaluate the NFDFDS scheme. Another observer based technique, state-space model developed with generic programming and the so-called extended unknown input observer (EUIO) was chosen. NFDFDS showed superior performance compared to its rival. There were however drawbacks to the system. One disadvantage concerned the availability of the data regarding the fault operating mode. This might not be available for real systems. Another disadvantage concerns the size of memory needed for its large matrices. This would slow down online computation. It was however possible to use effective techniques to calculate the matrices using less computing power.

In Deshpande and Patwardhan (2008), a novel multiple-operating regimes-based technique was proposed for performing online fault diagnosis in nonlinear systems. A Bayesian approach was used to identify a combination of linear perturbation models in different operating regimes that best-represents the plant dynamics at the current operating point. Nonlinear versions of the generalized likelihood ratio (GLR) method that use multiple linear models for fault identification were proposed. To hinder the performance degradation caused by the occurrence of faults, the information provided by the fault diagnosis component was then used for online fault accommodation. Analysis

of the simulation results on benchmark systems revealed that the proposed multimodel Kalman filter-based fault diagnosis schemes outperformed the linear GLR method when a nonlinear process is in a transient state over a wide operating range.

A more recent research related to the Bond graph model-based fault detection was presented in Borutzky (2009). In his paper residual sinks were used in bond graph model-based quantitative fault detection for the coupling of a model of a faultless process engineering system to a bond graph model of the faulty system. This way, integral causality could be used as the preferred computational causality in both models. There was no need for numerical differentiation. Furthermore, unknown variables did not need to be eliminated from power continuity equations in order to obtain analytical redundancy relations (ARRs) in symbolic form. Residuals indicating faults were computed numerically as components of a descriptor vector of a differential algebraic equation system derived from the coupled bond graphs. The presented bond graph approach especially aimed at models with nonlinearities that made it cumbersome or even impossible to derive ARRs from model equations by elimination of unknown variables.

2.2.2 Model-free approaches

In contrast to model-based approaches, no or little a priori knowledge is known about the system in the model-free approaches (Gertler, 1998; Liu, 2004; Zhang et al., 2001; Venkatasubramanian et al., 2003b). In the literature, terminology to describe this approach is not unified. The term *model-free approach* will be used here for it is a good abstraction of the approach. It is important to recognize that the word *model* refers to the model of the system which is used for residual generation or parity equations. In the machine learning literature, a representation of the learning algorithm is also termed a model but is not categorized as such here since it determines a fault through evaluating the process data directly.

A data-based FDI for a nonlinear ship propulsion system was presented in Liu (2004). The work was a novel attempt to study and compare the performance of fuzzy, signal processing and pattern recognition based FDI approaches on a nonlinear ship propulsion. The author asserted that of all three methods, the signal processing approach gave the best performance. It was able to detect and isolate five out of six faults whereas the other two were only capable of detecting three faults. However, only predetermined faults could be analysed.

In Zhang and Ding (2005), the authors proposed an approach of a fault detection system which did not require prior knowledge of the plant model. It was valid for linear discrete-time periodic systems. The proposed approach directly identified parameters of residual generators from input and output data of the periodic system instead of first identifying

parameters of the system model. Periodic parity relation based residual generators as well as periodic observer based residual generators could be obtained this way.

A model-free approach was presented by Previdi and Parisini (2006). The technique was based on the use of a specific spectral analysis tool, namely, the Squared Coherency Functions (SCFs). The detection of a fault was achieved by on-line monitoring an estimate of the squared coherency function, which was sensitive to the occurrence of significant changes in the plant dynamics. The alarm thresholds were based on the estimates of the confidence intervals of the SCF. In their work faults, whose primary nature are abrupt, have been analysed for the DAMADICS benchmark problem. In their experience, the proposed algorithm provided effective fault detection. In particular, for any value of the design parameters, the algorithm presented a high true detection rate. It is worth pointing out that in the simulated case, the signal used to excite the system, i.e. the control variable (CV) was a sinusoidal wave. The authors acknowledged that such a signal was not the best choice for spectral estimation based methods. However, this fact did not seriously limit the power of the proposed algorithm, at least in the discussed application.

A machine learning approach was presented in Castañon et al. (2005). The model consisted of a general statistical inference engine operating on discrete spaces. Their model represented the maximum entropy joint probability mass function (pmf) consistent with arbitrary lower order probabilities. The joint pmf was a rich model that, once learned, allows one to address inference tasks, which can be used for prediction applications. The model allowed the one step-ahead prediction of process variable, given its past values. The relevant past values for the forecast model were selected by learning a causal structure with an algorithm to learn a discrete Bayesian network. The author highlighted the fact that effective diagnosis was hindered due to the noise in data, cascaded effect and the perturbation by neighbouring nodes.

A similar approach was discussed in Yin et al. (2007). This work presented the application of the Support Vector Machine (SVM) algorithm to the flight control fault detection and isolation (FDI) system which was capable of identifying multiple flight control system faults. It introduced the idea of fuzzy-SVM where fuzzy logic was used to determine the optimal kernel function.

A simpler version of machine learning approach was presented in Ozbek and Soffker (2006). A feature-based fault detection approach was proposed. The main idea of this approach was to detect and identify faults in a complex system without any kind of modeling. By extracting features from relevant sensor signals yielded from Hardware-in-the-Loop simulations, and combining them in a matrix, it was possible for human operators to denote subsets of the matrix as fault-free and faulty areas. An advantage of this was the ability to set individual thresholds, giving more robustness towards false alarms and a possibility to denote individual subsets to relevant faults. The raw data

from fault-free and faulty simulations was used in the training of the matrix. The matrix was hand crafted and done offline.

A combined model-free and model-based approach to fault detection and identification (FDI) in a suction foot control system of a wall-climbing robot was presented in Jiang et al. (2009). For the control system, some fault models were derived by kinematics analysis. Moreover, the logic relations of the system states were known in advance. First, a fault tree was used to analyse the system by evaluating the basic events (elementary causes), which could lead to a root event (a particular fault). Then, a multiple-model adaptive estimation algorithm was used to detect and identify the model-known faults. Finally, based on the system states of the robot and the results of the estimation, the model-unknown faults were also identified using logical reasoning. Experiments showed that the proposed approach based on the combination of logical reasoning and model estimating was efficient in the FDI of the robot.

2.3 Reconfigurable Controller

A reconfigurable controller makes up the second part of a reconfigurable control system. This implements the changes to a control structure in order to compensate for faults. The changes that can be applied are not only limited to the parameters of the controller, but can also allow a total reconfiguration of the system. The typical structure of a reconfigurable control system is shown in Figure 2.4.

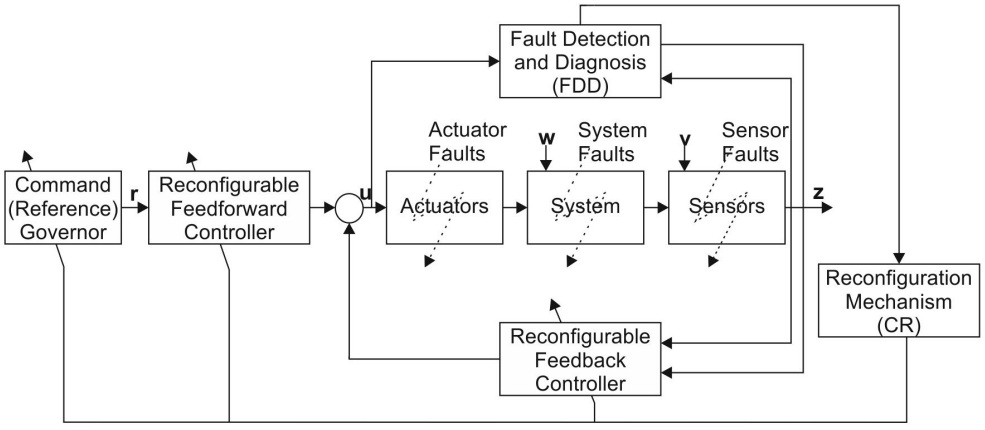


FIGURE 2.4: A general structure of a reconfigurable controller (Zhang and Jiang (2008))

There have been many approaches to designing a reconfigurable controller. They include adaptive control (Bodson and Groszkiewicz, 1997; Bolourchi and Hess, 1992; Brown and Harris, 1994), intelligent control (Ichtev et al., 2002; Ng and Jordan, 2000; Kretchmar, 2000; Diao and Passino, 2002; El-Fakdi et al., 2005; Jongcheol et al., 2006) and multi-agents control (Veres and Luo, 2004; Bojinov et al., 2002; Briot et al., 2007; Britain et al., 2008; Candea et al., 2001). Here, the approaches are divided into two broad

categories; an FDI based controller and a non FDI based controller. A categorization of the available approaches can be found in Zhang and Jiang (2008). In the following, some of these approaches are described.

2.3.1 Robust control

The structure of passive fault tolerant control systems (PFTCS) is generally fixed. They are designed to be robust against a number of presumed faults. It does not need an FDI nor reconfiguration scheme. Therefore, it can have limited fault-tolerant capabilities. Robust control systems generally fall into this category.

Robust control explicitly deals with uncertainty in its approach to controller design. Its methods aim to achieve robust performance and/or stability in the presence of bounded modelling errors. One of the important example is the $\mathcal{H}_\infty$ optimization technique. In 1981, Zames (1981) considered the minimization of the ∞-norm of the sensitivity function of a single-input-single-output linear feedback system. It quickly caught attention and was extended to more general problems after it was recognized that the approach allows dealing with robustness far more directly than other optimization methods.

A good explanation of the technique was given in Kwakernaak (1993). $\mathcal{H}_\infty$-optimization deals with the problem of minimizing the peak value of certain closed-loop frequency response functions.

Considerable amount of research has been done to solve the robust control problem and it is still an active topic. A state space solution to the standard $\mathcal{H}_2$ and $\mathcal{H}_\infty$ control problem was described by Doyle et al. (1989). The state space solution of the $\mathcal{H}_\infty$ problem required more assumptions than the frequency domain solution, such as the requirement that the transfer matrix be proper. On the other hand, the numerical algorithms for solving Riccati equations were better developed than the J-spectral factorization algorithms needed in the frequency domain approach. This approach was later picked up by Allgower et al. (1994) in their discussion of the implementation of $\mathcal{H}_\infty$ optimization technique on practical nonlinear systems.

The numerator-denominator or co-prime factor uncertainty problem was addressed in Glover and McFarlane (1989). It introduced the idea of normalized co-prime factors as a tool for obtaining robust stability using optimal $\mathcal{H}_\infty$ theory. It also showed that the maximum stability margin in the normalized LCF robust stability problem can be simply and directly calculated. Furthermore, it demonstrated a link between robust stabilization using $\mathcal{H}_\infty$ optimization and Nehari extension problems, and showed that the normalized LCF robust stabilization problem can be solved in this way. Finally, it gave an explicit state-space characterization of all suboptimal controllers for the normalized LCF robust stabilization problem. This method was an extension to the work done by Kwakernaak (1983); Vidyasagar et al. (1982) and Vidyasagar (1985).

In Yang and Stoustrup (2000), a novel approach for the synthesis of robust reconfigurable controller of a nonlinear system was discussed. Their controller took into account the parametric and additive faults as well as uncertainties generated by FDI algorithms. In this architecture the nominal and faulty closed-loop systems were combined in a fictitious augmented control system, so that the $\mathcal{H}_\infty$ and μ optimization methods could be used for the control synthesis. The integration of control reconfiguration with the robust FDI method proposed has yet to be tackled.

Shin et al. (2005) presented a reconfigurable flight controller using an adaptive sliding mode control scheme for actuator fault cases. Sliding mode controller, which performed well for systems with various uncertainties, was used to deal with the actuator faults. An actuator fault could be considered as a disturbance or an unexpected parameter change, which degrades the system performance and may destabilize the system. In their study, the adaptive sliding mode control technique was adopted to compensate for the effects of the disturbance generated by actuator faults. Lyapunov stability theory was used to derive the adaptive rule, and closed-loop system stability analysis was performed. To demonstrate the effectiveness of the proposed controller, numerical simulation was performed for aircraft with redundant control surfaces. The proposed controller did not require a FDI process and was considered a robust control scheme.

A similar approach could be seen in Demirci and Keresteciogluu (2004). In their work, a controller design method for underwater vehicles was presented. It was based on reconfiguration of a sliding mode controller in case of disturbances caused by shallow water conditions. The disturbance distribution information could be obtained and used to update the corrective gain vector of the sliding-mode controller. This increased the robustness of the controller and, hence, kept the system performance within acceptable limits. A state observer was used to obtain disturbances information.

More recently, the problem of reconfigurable control for constant gain output feedback controllers with Markovian parameters and state-dependent noise was considered (Aberkane et al., 2008). The authors asserted that although the relationship between the fault detection and identification (FDI) stage and the reconfiguration method stage was inherent, they assumed that the quality of the FDI was described by a known Markov process and so were the plant and actuator faults. They did not address the FDI problem in detail and hence their interactions were not properly investigated. From the FDI point of view, it was satisfactory that a fault was detected early and an alarm was generated. Another requirement was that a low false alarm rate was desirable. This did not take into account the state of the whole system. From the reconfiguration stage point of view, it assumed that a perfect FDI scheme was in place and ready to function. The authors tackles the problem by proposing a mathematical model that includes in the same structure the aspects of FDI and the reconfiguration algorithm. They used the active fault tolerant control systems with a Markovian Parameters (AFTCSMP) approach. This approach involved automatically identifying system faults such as sensor

or actuator failure using an FDI scheme and making reconfiguration decisions based on the detected faults. This was all done online. The main contribution of the author was the formulation of the circumstances needed to design a multi-performance control architecture related to this class of stochastic hybrid systems. The authors designed a mode-independent static output feedback controller using mode-dependent Lyapunov function approach. An $\mathcal{H}_2/\mathcal{H}_\infty$ synthesis controller was used to solve the problem. The group concluded with a numerical example showing that the applied algorithm obtained the desired disturbance attenuation and that the system was stochastically stable.

Other research on robust control include the model matching technique (Yang et al., 2007; Chen et al., 1989; Osa et al., 2003; Gao and Antsaklis, 1992), reliable control design (Hsieh, 2002; Boyd et al., 1994; Paz and Medanic, 1991; Siljak, 1980), probabilistic robust control (Qian and Stengel, 2005; Stengel, 1991; Barmish and Lagoa, 1999; Calafiore and Polyak, 2001), method of inequalities (Satoh et al., 1996; Whidborne et al., 1994), controllers derived via μ analysis and synthesis (Packard and Doyle, 1993) and generalized predictive control (Hess and Jung, 1989).

2.3.2 Multiple model control

An approach to dealing with very large uncertainties arising due to failures and damage was proposed by Boskovic et al. (2001). It was based on the Multiple Model, Switching and Tuning (MMST) methodology. The basic idea was to design an identification model in parallel to the corresponding controller which characterized and identified the failures at different regions in the parameter space. The other main idea was to construct a strategy to switch between controller sets to accomplish a given goal. The identification models were run to diagnose an error while the plant was executing one of the controllers. It found a model which was similar to the current operating algorithm of the plant. The switching mechanism then had a choice to either switch or stay at the corresponding controller.

A novel fault-tolerant control system design technique has been proposed by Jin and Zhang (2006), which blended the multiple-model principle with the unavoidable performance degradation due to faults in actuators, sensors or system dynamics. The number of models employed depended on the characteristics of the system, the nature of the failures considered, and the physical limits of system variables. The achievable performances under various component failures were represented in the form of reference models, known as performance reduced reference models. These models were used to synthesize a set of controllers. Under a specific fault condition, proper controller and revised control system command inputs were selected automatically to achieve the desired performance. A simulation example of an aircraft subject to different types of failure was used to illustrate the design process and to demonstrate the effectiveness of the method.

More recently, Yu and Jiang (2012) proposed a hybrid FTCS which combined passive and active FTCSs to counteract the partial failures in the actuators. The authors offset the slow speed of a detailed fault diagnosis by designing a passive FTCS to stabilize the system with minimal fault information. A reconfigurable controller was then synthesized once comprehensive fault diagnostic information was obtained.

2.3.3 Gain scheduling control

A more simplified approach was taken by a group from NASA. A control design for the X-33 vehicle was developed that took into account a failed control surface and re-distributed the energy to other working surfaces to obtain satisfactory stability and performance (Cotting and Burken, 2001). Here, the design of reconfigurable control methods used an offline nonlinear constrained optimization (ONCO) approach. A simulation was done for the full flight envelope with simulated actuator faults. The FDI scheme was embedded in the actuator controller. Their results showed why reconfiguration was desirable compared to the nominal control mixer. Redundancy plays a major role in reconfigurable control systems. In aircraft flight control systems, redundancy is included in the design to decrease the risk due to failure. Electromagnetic actuators (EMAs) are preferred to hydraulically powered actuators, even though fault analysis has confirmed that EMAs are more likely to fail. The advantages outweigh the disadvantages, such as routing of signal cables. The main reason why reconfigurable control was pursued was to improve the use of functioning actuators in the event of actuator failure. Since EMAs are relatively new and untested compared to hydromechanical actuators, the reconfigurable control design ensured that the vehicle will operate with satisfactory performance. The ONCO method was chosen due to its testability and validity. The vehicle was equipped with smart actuators to tell it which actuator has failed and its last known position. This simplified reconfiguration since the FDI did not need to be part of the primary flight controller. With prior knowledge of what failures could occur, a lookup table was implemented so that system complexity was avoided. Using BFGS formula, offline simulation tests were run and gain sets were modified until the summed error was optimally reduced. The reconfiguration design was only limited to control surface failures. The overall results show that the control design was better suited for certain control failures than others, such as a left body flap during ascent, an inboard elevator during entry, and a rudder during landing.

In Izosimov et al. (2005), an approach to the design optimization of fault tolerant embedded systems for safety-critical applications was presented. Processes were statically scheduled and communications were performed using the time-triggered protocol. The process of re-execution and replication for tolerating transient faults was used. The design optimization approach decided the mapping of processes to processors and the assignment of fault-tolerant policies to processes such that transient faults were tolerated

and the timing constraints of the application were satisfied. Several heuristics able to find fault-tolerant implementations given a limited amount of resources were presented.

2.3.4 Adaptive control

A more direct approach to reconfigurable control is the adaptive systems architecture. In this approach, a diagnosis module is omitted and faults are not detected through explicit FDI. There are two types of adaptive control: model-based and model-free adaptive control.

2.3.4.1 Model-based Approach to adaptive control

Bodson and Groszkiewicz (1997) presented an indirect adaptive control approach that was done in two stages: first the plant parameters were estimated, then the control parameters were derived from them. Direct adaptive control on the other hand estimated the control parameters. By comparison, the author chose direct adaptive control due to the lower identification number of parameters. This was crucial for real time computing. An input error direct algorithm was chosen over output error direct algorithm due to the fact that an output error direct algorithm was more rigid in its implementation. The author stated that the performance of the algorithm needed to be tested in a noisy environment since it was not done in his work. Also, the performance still needed to be evaluated at other flight conditions throughout the whole flight envelope.

A system was designed using this method for satellite formation flying (Thanapalan et al., 2006). The paper proposed the use of model reference adaptive control (MRAS) and quaternion based adaptive attitude control (QAAC) instead of the traditional FDI. Since explicit remodelling, decision making and control redesign could be avoided, the adaptive systems approach was much simpler to implement. Since sensor deficiency was a major issue and should be addressed accordingly, redundancy was introduced to enable solutions to the problem. A model of the system to be controlled was expressed in a state space form and the controller was designed accordingly. The proposed reconfigurable control for actuator degradation was divided into two parts: position control and attitude control. The position control was based on MRAS and the attitude control on QAAC. For the sensors, a voting principle was used to determine the health of the sensors. A numerical simulation was done to test the system. The simulation showed that, for changes in actuator gains at any time, the control system managed to adopt itself to the new actuator gain. Work still had to be done on mapping out the limitations of system reconfiguration due to physical constraints.

A similar MRAS approach was taken by Shore and Bodson (2004). Flight tests were conducted on radio controlled model aircraft with actuator failures. The actuator failures

(i.e. elevators, aileron and engine) were triggered remotely by the operator. The work was limited to demonstrate the identification and reconfiguration algorithms in actual failure cases. A continuously adaptive approach was used where parameters were estimated to compute the gains of the control law. The categorization of the aircraft health or failure state was not attempted. Parameter identification was done by implementing a modified version of the least-squares optimization criterion that was suitable for adaptation. Control reconfiguration was achieved by using a special case of the MRAS. The idea was to apply another set of gains to the pilot command during a faulty state so that the overall gain of the system remained constant, regardless of the mode it was operating in. Overall, results showed that off-line and real time identification of critical aircraft parameters could be obtained reliably. The estimated parameters could be used to observe the actuator failure's effect. The results also showed that the controller designed was successful in compensating for failures and changing flight conditions.

A more recent article which is similar to Bodson and Groszkiewicz (1997) discussed the implementation of direct adaptive model reference control for weapon systems (Wise and Lavretsky, 2006). Direct adaptive increments to extend a fundamental control signal designed using nonlinear or linear robust control methods were incorporated. The control system was tested without a prior knowledge of the modified weapon's aerodynamics. This significantly reduced cost and development time. Again, FDI was not needed in this approach. The work done by the authors had a slightly different approach than their predecessors. Instead of totally replacing the control system, the author proposed its augmentation with and adaptive increment derived from an online learning algorithm. The system design was started off with a baseline or fundamental nominal controller expressed in state-space vectors without considering system uncertainties. In the next step actuator dynamics were removed and system uncertainties were introduced in the closed loop equation. In the presence of uncertainties and inaccuracies, a model reference direct adaptive incremental control signal derived using neural network algorithm was introduced to gracefully degrade the closed loop system. The proposed design also took into account actuator saturation by adaptively modifying the controls. Flight tests of the missiles were conducted and all objectives were met. The author also summarized open problems that still needed to be addressed. Some of the problems were the choice of a suitable reference model; scheduling of models for different flight phases still; a lack of clear guidelines for tuning parameters; transient performance of adaptive systems; settings that needed for adaptive dead zone and learning rates; and refining the methods for the controller. The author concluded by stating that direct adaptive model reference control was suitable to be used for nonlinear systems. Further work was still needed to address open issues.

2.3.4.2 Model-free approach to adaptive control

The concept of an adaptive-predictive controller was first explored by Martin-Sanchez (1976). The method required knowledge of only input and output data. Consequently, no state estimation was necessary. A predictive model based on the desired output and according to the predictive control principle generated the control signal. An adaptive mechanism then adjusted the predictive model parameters to minimize the prediction error. This method differed from the adaptive controllers discussed previously in that model based adaptive control involved a lot of up front work programming a fixed model that could not evolve as the process dynamics did. The issue with this was that if the process variables changed from the initial settings, the model needed to be reworked. With adaptive-predictive controllers it was able to adapt to changes and so the process operation performance did not deteriorate with time.

In Kurnaz et al. (2007), an adaptive neuro-fuzzy inference system (ANFIS) based controller was designed for a UAV. The author implemented fuzzy logic for altitude, speed and roll angle control. A hybrid learning algorithm was used to update the ANFIS parameters. The algorithm was compared to PID based controllers and was found comparable to the latter despite the model free approach. For other flight conditions, unstable performance was observed. This could have occurred due to the lack of optimal learning algorithm.

ANFIS was also the control method preferred in Tahour et al. (2007). Here the author applied ANFIS to control a switched reluctance motor which can be categorized as a nonlinear system. Adaptive neural networks were implemented to learn the membership functions to be implemented in the fuzzy inference system. The Sugeno fuzzy model was implemented for its high interpretability and computational efficiency. It was also chosen for its built-in optimal and adaptive techniques. Simulation results showed that the ANFIS controller was superior to the conventional controller in robustness and tracking accuracy. It also showed that the ANFIS controller had a high performance level in the presence of parameter uncertainties and load disturbance. A fast dynamic response without overshoot and zero steady state error were observed with ANFIS as a speed controller.

In Savanur et al. (2008), a model-free approach was used to determine surface fault. The adaptive neuro-fuzzy inference system (ANFIS) was trained with its input as error between nominal state and faulty state. Its output on the other hand was the parameters for the control distribution matrix. Reconfiguration was carried out by computing new feedback gains using the trained ANFIS. The algorithm was simulated and compared with the model based approach. The filter chosen was the Extended Kalman Filter (EKF). Results showed that the ANFIS scheme approached its reference value faster than EKF. It also showed that the reconfigured system converged to its reference value and ANFIS showed less overall error.

2.3.5 Intelligent control

A less conventional approach to reconfigurable control can be achieved with intelligent control. It is an exciting alternative design methodology which can be applied in developing nonlinear systems.

Fuzzy logic is a popular choice for intelligent control. Due to its fuzzy rules, it is also considered robust which eliminates the need to explicitly handle system robustness issues. An experimental comparative analysis of fuzzy logic based controllers (FLC) and conventional controllers was done in Mrad and Deeb (1999). FLC required expertise knowledge of the process operation for FLC parameter setting, and the controller could be only as good as the expert involved in the design. To make the controller less dependent on the quality of the expert knowledge, different adaptation schemes to compensate for this deficiency and a practical adaptive fuzzy logic controller (AFLC) were proposed. The different techniques were simulated on a DC motor example, and implemented on a hardware station. Conventional control was found to perform better when the mathematical model of the plant was accurate. Fuzzy controllers outperformed their conventional counterparts when the model was inaccurate. It was also the case when large disturbances acted on the DC motor, changing the dynamics of the system and rendering conventional control inefficient. It showed faster settling time and better recovery when perturbed.

A solution to online reconfiguration of control systems using fuzzy logic was proposed in Benítez-Pérez et al. (2005). Reconfiguration was proposed in three stages. First, degradation in time communication within the computer network was used to detect faults. Secondly, based upon this scenario a strategy for online reconfiguration was pursued in order to account for faults where new time delays appear between elements. These delays modify the behaviour of the dynamical response of the system. During the final stage, the control law was modified in terms of current time delays. Online system reconfiguration as multi-variable and multi-stage problem was pursued based upon a quasi-dynamic scheduler that took into account those predetermined time delays and the related control laws. Control reconfiguration was pursued as soon as structural computer network reconfiguration has occurred. The triggering scenarios of reconfiguration were done offline. A planning scheduler selected suitable scenarios and related control laws. When the system was brought online, a simple comparison between a proposed plan and the selected plan allowed online reconfiguration. The related control law was dispatched to the rest of the elements in the computer network once the plan has been validated. This approach presented an ad hoc view of how control performance needed to be considered in order to develop on-line system reconfiguration based upon a quasi-dynamic scheduler algorithm. The author proposed further work in terms of a more precise comparison between proposed and predefined plans. Two different

approaches may be pursued; the use of Neural Networks for pattern classification and genetic algorithms for table optimization.

An alternative to the neuro-fuzzy approach is the reinforcement learning framework (Dayan and Watkins, 2001; Dietterich, 2000; Sutton and Barto, 1998; Mehta and Tadepalli, 2005). In contrast to neural network and fuzzy logic, no training data is given in reinforcement learning. There are also no fixed logic rules. The reinforcement learning *agent*, as it is termed, is given a reward function to optimize. It tries to maximize its total rewards by interacting with its environment. It learns online and discovers optimal actions given a specific state.

Kretchmar (2000) combined robust control theory and a reinforcement learning algorithm to develop a stable neuro-control scheme. Functional uncertainty was used to represent the nonlinear and time-varying components of the neural network. Robust control techniques were applied to guarantee the stability of the neuro-controller. The scheme provided stable control not only for a specific fixed-weight, neural network, but also for a neuro-controller in which the weights were changing during learning. A static and dynamic stability test was developed to determine the stability of the control system. A challenging aspect of the author's second objective was to develop a suitable learning agent architecture. The reinforcement learning algorithm was chosen because it is well suited to the type of information available in the control environment. It performs the trial-and-error approach to discovering better controllers, and naturally optimizes the performance criteria over time. A high-level architecture was designed based upon the actor-critic design in early reinforcement learning. This dual network approach allowed the control agent to operate both like a reinforcement learner and also as a controller. Neuro-dynamic difficulties peculiar to the control situation were addressed. These problems were solved by selecting a low-level architecture with a two-layer feed-forward neural network as the actor, and a discrete, local, table look-up network as the critic.

Similar work exploiting the reinforcement learning framework could also be seen in Ng et al. (2004). Direct policy search (DPS), a class of reinforcement learning algorithm, was used to control an autonomous helicopter. The DPS algorithm was chosen due to the fact that it is effective for low level control tasks. An improvement to the REINFORCE algorithm was made (Williams, 1992). It was argued that the convergence of the REINFORCE algorithm took a very long time. This was due to the random sampling of state-action pairs that had to be done during the learning process in order to update the algorithm. The authors discovered that by generating a sequence of random samples before learning begins and using these fixed samples during the actual learning, the time it took for the algorithm to converge could be significantly reduced.

A relatively new approach is the use of multi agents or beliefs-desires-intentions (BDI) agent architecture. Early work in this field was seen in Shoham (1993). A new computational framework was presented, called agent-oriented programming (AOP). It can be viewed as a specialization of object-oriented programming. The author described an agent as having components such as beliefs, decisions, capabilities, and obligations; for this reason the state of an agent was called its mental state. The mental state of agents was described formally in an extension of standard epistemic logic: besides temporalizing the knowledge and belief operators, AOP introduced operators for obligation, decision, and capability. Agents were controlled by agent programs which included primitives for communicating with other agents. In the spirit of speech act theory, each communication primitive was of a certain type: informing, requesting, offering, etc.

In Bojinov et al. (2002), useful control techniques provided by multi-agent systems for modular self-reconfigurable (metamorphic) robots were demonstrated. Such robots changed their overall shape to suit different tasks. They consisted of many modules that could move relative to each other. This architecture was particularly useful for systems involving uncertain and changing environments. The results showed creation of emergent structures with the desired functionality. Purely local, simple rules and limited sensing were applied. According to the author this approach could be used in conjunction with other self-reconfiguration or control methods, as part of an overall hierarchical control scheme.

An essential aspect of multi-agent architecture is the communication protocol between agents. This was presented in Shen et al. (2002). It addressed two basic problems: how modules in these robots communicated with each other in a dynamic and unexpectedly changing connection between them, and how they collaborated their local actions in a distributed manner to accomplish a common goal such as locomotion or reconfiguration. This problem is valid in any general multi-agent based self reconfigurable system. The author presented a biologically inspired approach to address the problems stated and emulate the concept of hormones used among biological cells for both communication and control. The basic concept was that a single hormone signal could travel through the entire network of modules causing different reactions based on their local receptors, sensors, topology connections, and state information. In the computing world, hormones could be seen as a type of content based message but with no specific destination, having a lifetime and triggering different actions from different receivers. The author introduced the adaptive communication and adaptive distributed control protocol for modules in such architectures. The author argued that this approach could be promising but further work still needed to be done on how to develop an appropriate rule set to generate a specific behaviour.

Veres and Luo (2004) proposed a multi-agent approach on control systems with a high degree of autonomy. This architecture contained agents for various components of the system, for example modelling and controller optimization. This new breed of agents

was called cautiously optimistic control agents or COCA, and applied new modelling results with caution while using current settings until a certain threshold was exceeded. On the implementation side, agent oriented programming (AOP) which allowed actions to be triggered by events was used. COCA is a multilayer architecture with a central unit acting as the coordinator or supervisor of the entire system. Plans and tasks were distributed among multiple agents. Agents such as the physical modeller agent and experimenter agent had specific tasks to complete and must communicate the results to other agents. These results could be used as the inputs for other agents. The key feature was that the central unit did not have full authority over the agents' responses. The cooperative action between the different components made the architecture successful. The authors concluded that further work on this area could be to check by formal methods the details of potential failure modes of operation. Also, the adaptation of the principles for other iterative controller techniques could be explored further.

A mathematical analysis of multi-agent systems was presented in Lerman et al. (2004). A multi-agent system could be described through simple probabilistic equations when observed collectively though they were stochastic individually. The work showed that a class of mathematical models from the details of individual agent controllers could be derived to describe the dynamics of the collective systems. The approach used a stochastic master and rate equation which described how the average macroscopic system properties changed over time. All relevant states and transitions of the multi-agent system had to be accounted for to successfully implement the model. The model was applied to study the collective behaviour of robotic systems. Results showed that even the simplest type of dimensional analysis of the equations yielded important insights into the system. This approach could be extended to heterogeneous agent systems. There were however, limitations to the approach. Systems composed of agents with memory, learning or deliberative capabilities do not, and therefore cannot, be described by the simple models derived in this paper. Another limitation was that this approach was best suited to large systems and could lead to fluctuations when implemented on average systems. It could also be difficult to determine the transition rates between states, especially in the event correlated activities between agents. These equations were suitable to address the probability description, but were not suitable for determining behaviours such as worst-case bounds, behaviour in exceptional situations or extreme values of distribution.

In Duhaut et al. (2007), the authors tackled the problem of deadlock, where a situation is reached when all the robots are unable to move or a set of robots are in oscillation. To avoid this problem, a general order on the environment that guaranteed a hierarchy of behaviours between the robots was introduced.

2.4 Integrated Approach to Reconfigurable Control

Early work in the integrated design approach was seen in Jacobson and Nett (1991). It presented a foundation for an integrated approach to the design of controls and diagnostics in reliable control systems. In this approach the control module and diagnostic module of the control system were designed together instead of independently, thereby accounting for the interactions which occur between the two modules in a functioning control system. This approach was known as the four parameter controller. It was a a generalization of the two parameter controller. It was made up of two vector inputs (rather than one) and two vector outputs; correspondingly, this controller comprised of four matrix parameters (rather than two). The additional controller output may be viewed as a diagnostic output which was monitored to detect and isolate sensor and actuator faults. This work only dealt with linear systems and left nonlinear systems for further research.

An extended approach to the above was discussed by Wu (1997). The design of a reconfigurable control algorithm that helped in acquiring information for diagnostic purposes in fault tolerant control system was presented. Two indices, showing the performance and diagnostics respectively, were chosen to indicate overall performance. The relationship between the two indices was established that clarified the control/diagnostics tradeoffs. A set of controllers that achieved a specified performance level was designed first and the remaining degree of freedom of the controller set was used for optimizing the diagnostic performance. The author presented the typical modules of a reconfigurable control system: a control module, a diagnostics module and a reconfiguration module which linked the other two. The basis of the reconfiguration module was that it could only be done with suitable redundancy relations between the outputs and inputs. A major problem area linked to reconfigurable systems was controller robustness and failure sensitivity. The normal solution would be to sacrifice control performance for diagnostics. To maximize failure sensitivity measure and sustain the control performance, the paper emphasized the importance of an integrated approach to reconfigurable control. It gave an example of placing a threshold that indicated whether a certain failure should cause a control reconfiguration to take place. Through integration, it was possible to place the reconfiguration threshold at a higher level than the failure detection threshold. This resulted in the enhancement of the overall reliability of a reconfigurable control system. The solution to the integrated control/diagnostic design problem was the main contribution of the author. The interaction was described as an inequality involving respective performance indices. An important assumption by the author was that the plant to be controlled could be represented as a linear model. The objective of the control was to follow a command signal and reject a noise signal. The author used the $\mathcal{H}_\infty$ controller due to its property of rejecting the worst exogenous signals. The architecture was suitable for control reconfiguration. This type of controller could be parameterized, hence some room for achieving additional objectives remained. The

$\mathcal{H}_2$-norm was then used as the diagnostics performance measure. The paper concluded by giving a numerical example showing the improvements brought in by optimizing a diagnostic performance and the acquisition of diagnostic signals from appropriate locations in the control loop. The results showed that 1) the sensor bias could be seen more evidently with the optimized diagnostic signal and 2) that diagnostic signal's noise was better suppressed relative to other measurable signals.

Balle et al. (1998) introduced a model based on a fuzzy functions of the process. This model was meant for a heat exchanger. It integrated a model-based adaptive controller with a multi-model fault detection scheme. Four fuzzy models of different sub-processes were used to detect faults. The author left replacing fault measurements with an estimation for further research.

A method which allowed one to explicitly incorporate allowable system performance degradation in the event of partial actuator fault in the design process was discussed in Zhang and Jiang (2003). The method was based on model-following and command input management techniques. The degradation in dynamic performance was accounted for through a degraded reference model. The degradation in steady-state performance was dealt with using a command input adjustment technique. An eigenstructure assignment algorithm was used in an explicit model-following framework so that the dynamics of the closed-loop system followed that of the degraded reference model. The command input was also adjusted automatically in parallel to prevent the actuators from saturation.

Wang and Wang (2011) considered the FDD and control of a four-wheel independently driven electric ground vehicle using vehicle dynamics and motion signals. A hybrid approach to fault tolerant controllers was taken. An adaptive control-based passive fault-tolerant controller was first designed to maintain vehicle stability when a fault occurred. Then, an active fault diagnosis method which involved introducing an additional control gain multiplier when a fault occured was proposed to isolate and evaluate the fault under the designed passive FTC. The resources were finally reallocated based on the diagnosis result to relieve the torque demand on the faulty actuator avoiding further damages.

Yetendje et al. (2012) proposed a robust multi-sensor fault tolerant model-following model predictive control design for constrained systems. A sensor FDI strategy which employed a bank of sensors-estimator combinations was considered. The strategy verified that, for each of these combinations, the updated estimation tracking errors lay inside pre-computed "healthy" sets. For the reconfiguration component, an active fault-tolerant control scheme based on the output feedback problem for constrained linear discrete-time systems subject to state and measurement disturbances was proposed. The author's work was concerned with how resources (sensor) were reallocated into groups when a fault occurred. The author only considered sensor faults and linear systems.

2.5 Chapter Conclusion

A number of approaches to designing reconfigurable controllers were presented. Due to the complexity of the problem, research done in this area was divided into the research on FDD and research on CR. Recently, effort has been made to unify them in a framework as discussed in Section 2.4. The merger of different subsystems should be straight forward. In practice however, this is not the case. The difficulty lies in the fact that each subsystem struggles to provide instantaneous information for other subsystems despite working perfectly on its own. An effective integration of the two components still remains an open issue. *To this end, the multi-agent approach described in Section 2.3.5 is a potential solution.* It offers the advantage of viewing the reconfigurable control system in a modular manner. It allows us to break down the system into subcomponents so as to simplify the design and implementation.

The different methods of fault tolerant controllers were discussed. There were many controllers from which one could choose to implement. They were divided into two broad classes. Passive fault tolerant control systems have limited capability in reconfiguring the system. They were designed to cope with only a fixed number of faults. On the other hand, active fault tolerant control systems change or adapt the control architecture to accommodate a fault. The adaptive-predictive controller (Martin-Sanchez, 1976) for example, was able to learn a controller without a priori knowledge of the system model. However, this controller is generally suitable for linear systems. In fact most of the research done in this field concentrated on linear systems. In contrast, a machine learning approach would be able to deal with nonlinear systems.

It was seen in Section 2.2.1 and Section 2.3.5 that the machine learning approach to reconfigurable control systems was mostly implemented for neural networks and fuzzy logic or a combination of the two. *The implementation of other learning algorithms in this framework were relatively unexplored.* An alternative method is the use of reinforcement learning. Due to its "trial and error" based learning, and its ability to find new solutions to a control problem, it truly is a self reconfiguring system as we will prove that in chapters to follow. Reinforcement learning algorithms are especially good at finding solutions where the control strategy is not so straight forward. For example, most conventional control systems will fail in trying to drive an under powered vehicle up a steep hill (Sutton and Barto, 1998). Reinforcement learning algorithms would be able to handle such a problem because it takes into account that each action will affect future reward of the system. Therefore, it will sacrifice smaller immediate reward for a long term bigger reward. Another reason to explore the reinforcement learning approach stems from the fact that the theory of reinforcement learning fits naturally in the overall multi-agent architecture.

Two general approaches in FDD were presented. The model-based approach has the advantage of being highly effective when a plant is linear in nature and can be modelled

correctly. However, in practical systems which behaves in a nonlinear manner, the effectiveness of this approach is reduced due to poor linear approximations of the model. Another drawback to this approach is that for many practical systems, an exact model of their dynamics is not obtainable. The model-free approach requires no or little a priori knowledge of the system. It uses data gathered from the system to detect, isolate and identify a fault. From the literature, model-free approaches using machine learning have been very popular in solving the FDD problem. Machine learning has the advantage of being able to recognize patterns in high dimensions where conventional methods fail to do so. The power of machine learning algorithms enables us to deal with nonlinear systems without linearising at a point. Due to this reason, our work will look into the possibilities of using the model-free approach with machine learning techniques in realizing our monitor and control agents A_m and A_c in Chapter 4.

Redundancy of system components plays an important role in a reconfigurable controller. This was emphasized in Section 2.2. In particular, the concept of analytical redundancy relations (ARR) paved the way to extensive research in the field of reconfigurable controllers. However, the management of these redundancies in the overall architecture still remains an open issue. The mechanism of transferring resource allocations in the event of faults is an active research topic. In our work, we will explore the idea of using redundant sensors and actuators. It is our view that without redundancy there would be no case for reconfigurability. Our approach will enable the system to choose between working sensors and actuators in order to ensure stable operation. This is possible due to the modular view of the system.

Chapter 3

Definition of Fundamentals

We have seen in Chapter 2 some of the possible approaches to solving the reconfigurable control problem. As it appeared to be promising, it was decided that this work will investigate a multiagent approach. In this chapter we will define the concepts that were relevant to our work.

3.1 Definition of Agents

The notion of agents is applied to describe a system or a group of systems which act autonomously. Literature in this field suggests that there is no consensus on how an agent is defined (Franklin and Graesser, 1997; Wooldridge, 2002; Veres and Luo, 2004; Briot et al., 2007; Shoham and Leyton-Brown, 2008). However, all researchers agree that an agent has to at least have three capabilities:

- *autonomy*: agents work without direct influence of outside elements and have a certain degree of control over their actions.
- *sensing ability*: agents are able to sense their environment.
- *acting ability*: agents are able to act upon their environment.

Taking these capabilities into account, we will adhere to a minimally restrictive definition of an agent for the rest of our work.

Definition 3.1. An agent is represented by an internal architecture $\sigma \in \Sigma$, a communication set $\chi \in \mathcal{X}$ and a resource set $\kappa \in \mathcal{K}$:

$$A = \langle \Sigma, \mathcal{X}, \mathcal{K} \rangle .$$

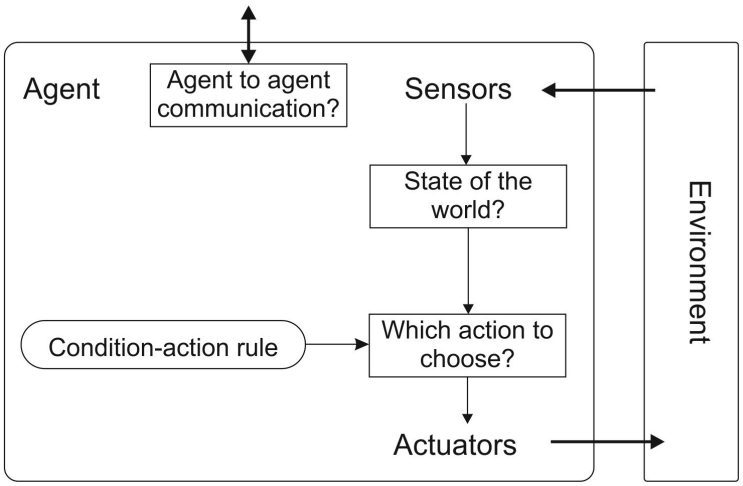


FIGURE 3.1: Reactive agent block diagram. Adapted from (Wooldridge, 2002).

The internal architecture σ of the agent determines how the agent is built. This includes the algorithm that the agent uses to makes decisions or how the internal memory (if any) is laid out. The communication set χ describes how an agent communicates with other agents in the environment. The resource set κ includes all necessary sensors and actuators needed to perceive or act upon the environment.

Agents are classified according to their properties such as *reactive, learning, utility oriented, goal oriented, adaptive*, and others (Wooldridge, 2002). In our work, we will only deal with two kinds of agents, reactive agents and learning agents.

A reactive agent is illustrated in Figure 3.1. A reactive agent's sensors map directly to actions. Its internal architecture is made up of conditional action rules which defines the mapping. We will not go deeper into the internal architecture of a reactive agent as it is straight forward to implement.

Figure 5.1 shows an illustration of a learning agent. The internal architecture is equipped with a learning mechanism. It continously evaluates its actions and improves its action selection by learning as the agent interacts with its environment.

The internal architecture of a learning agent deserves more attention. In the next few sections, several machine learning algorithms will be presented which serve as the foundation to understanding how a learning agents' internal architecture are realized.

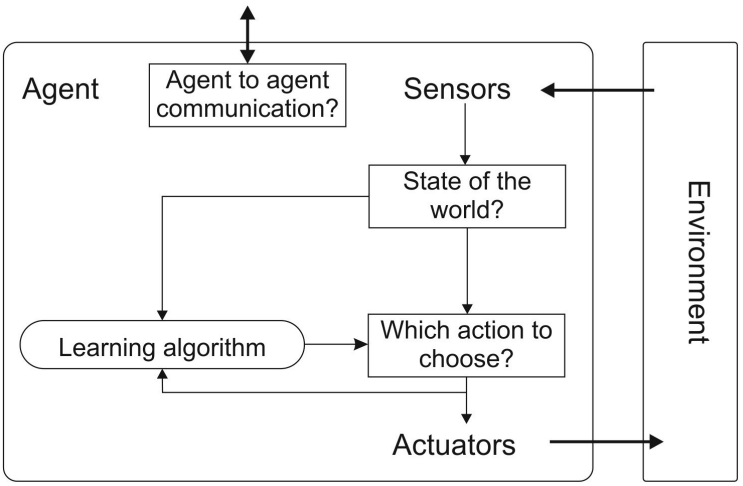


FIGURE 3.2: Learning agent block diagram. Adapted from (Wooldridge, 2002).

3.2 Artificial Neural Networks

Machine learning is a field of study that gives computers the ability to learn certain relationships or actions without explicitly being programmed (Samuel, 1959). For instance, given a dataset, the machine should be able to produce a model which accurately represents the data. The reason why we would want the machine to model the data is that the machine can then predict an outcome in the world based on the model. The prediction task could be for classification of data, trying to find a functional relationship between input and output data or for probability estimation where the variable distributions are modelled. There are three types of machine learning approaches. They are supervised learning, unsupervised learning and semi-supervised learning. In our work all three types of machine learning approaches are implemented. We will start by presenting the artificial neural network.

The idea of neural networks came about when machine learning researchers tried to develop an algorithm which mimics how the human brain works. A simple artificial neural network is shown in Figure 3.3. The basic element of a neural network is a *node*. They are arranged in layers. Each node in a layer is connected to a preceding layer through a weighted link. The input signal to the network is shown at the input layer. This signal is propogated forward through the network until it reaches the final layer which is the network output. The intermediate layer is called a hidden layer since the nodes are "hidden" inside the network. The *bias* is a way of adding a constant term to a neuron or layer output. These types of network are also termed *feedforward* networks since no cyclic connections exist. This means that no arrow in Figure 3.3 is pointed to a preceding layer. The weights, which are used as "gains" applied to signals between neurons, are denoted as θ_{ij}. We say that θ_{ij} connects node i to node j. This is illustrated in Figure 3.4.

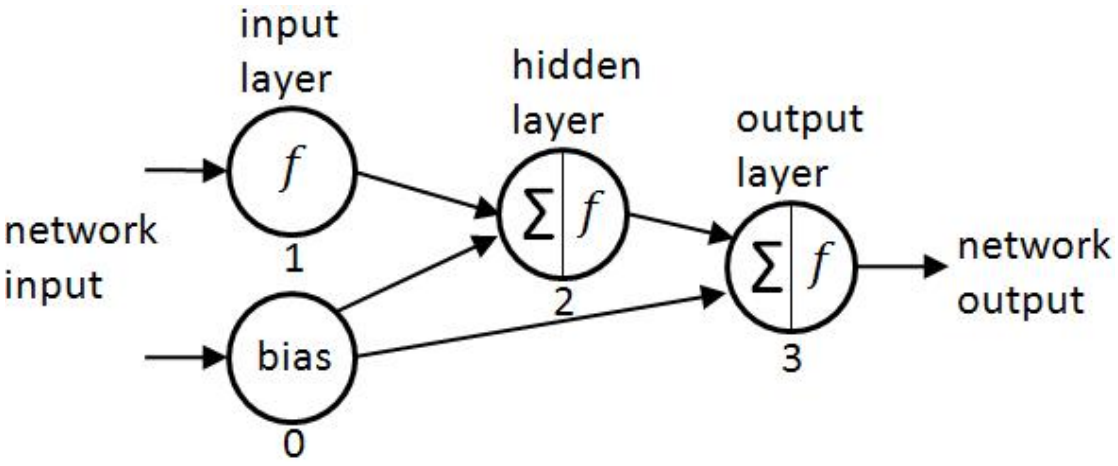


FIGURE 3.3: A feedforward multi-layer neural network where a node corresponds to a circle and each node is connected through a weighed link. Adapted from Brown and Harris (1994).

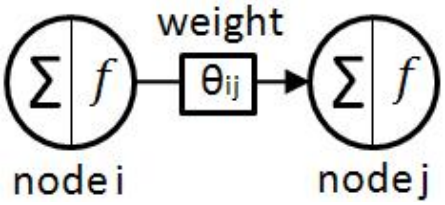


FIGURE 3.4: Connection between nodes through weights θ_{ij}.

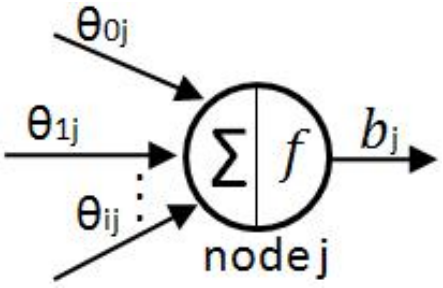


FIGURE 3.5: Input and output of a single node.

The function f is the node's *activation function*. In the simplest case, f is an identity function. The input and output of a single node is shown in Figure 3.5. The output of the node b_j can be written as

$$b_j = f(\sum_i \theta_{ij} b_i).$$

The weighted sum $\sum_i \theta_{ij} b_i$ is normally called the *net input* to node j. It is often written as net_j.

If we defined $x_0 = 1$ and x_1 as the inputs to our network, the network could be represented as in Figure 3.6.

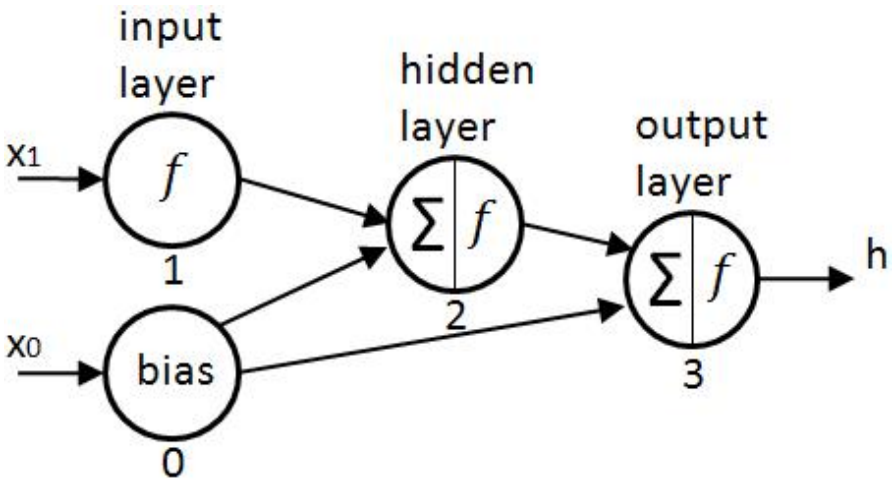


FIGURE 3.6: Network to represent function h.

The output for the entire network in Figure 3.3 can be written as follows:

$$
\begin{aligned}
b_0(x, \boldsymbol{\theta}) &= x_0 \\
b_1(x, \boldsymbol{\theta}) &= f(x_1) \\
b_2(\mathbf{x}, \boldsymbol{\theta}) &= f(\theta_{02} b_0(x, \boldsymbol{\theta}) + \theta_{12} b_1(x, \boldsymbol{\theta})) \\
b_3(\mathbf{x}, \boldsymbol{\theta}) &= f(\theta_{03} b_0(x, \boldsymbol{\theta}) + \theta_{23} b_2(\mathbf{x}, \boldsymbol{\theta})) \\
h(\mathbf{x}, \boldsymbol{\theta}) &= b_3(\mathbf{x}, \boldsymbol{\theta}).
\end{aligned}
$$

In practice the activation function for the input and output layer is set to be the identity function. A common activation function used in a hidden layer is the hyperbolic tangent function $f = \tanh(x)$. The advantage of using hyperbolic tangent function is that it normalizes incoming data to be in the range of ± 1. This results in the data set having a normal distribution which is very tractable analytically. Figure 3.7 shows a graph of a hyperbolic tangent.

Due to the non-linear nature of a hyperbolic tangent function, theoretically a combination of several hidden nodes with a hyperbolic tangent function allow the network to model any non-linear relationship (Swingler, 1996). The function $h(\mathbf{x}, \boldsymbol{\theta})$ is known as the hypothesis or predictor function and can be rewritten as

$$h(\mathbf{x}, \boldsymbol{\theta}) = \theta_{03} x_0 + \theta_{23} \tanh(\theta_{02} + \theta_{12} x_1).$$

The hypothesis $h(\mathbf{x}, \boldsymbol{\theta})$ is equal to the output node of the network. If we define the index "o" to indicate the output layer, the hypothesis h can be formulated as follows:

$$h = b_o = f(\sum_i \theta_{io} b_i).$$

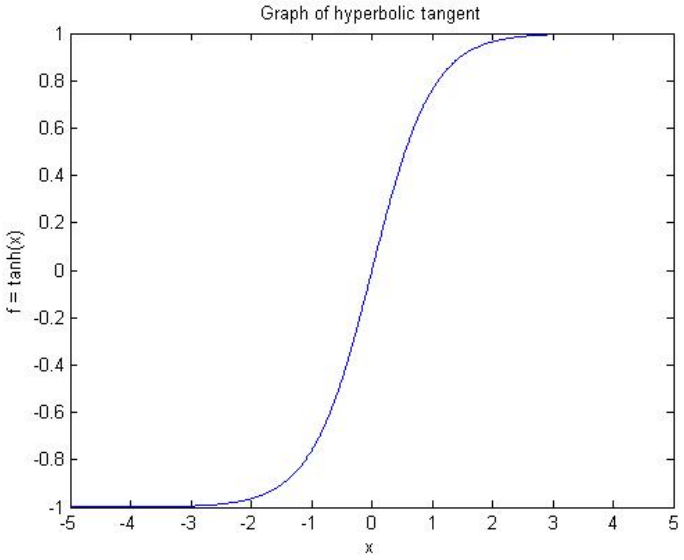


FIGURE 3.7: Graph of a hyperbolic tangent function.

Note that index "i" refers to the nodes of the preceding layer.

In general there could be multiple inputs and outputs to a network. Usually a training set is presented to the network in the form of $\{(\mathbf{x}^{(l)}, \mathbf{y}^{(l)}); l = 1, ..., m\}$. The hypothesis or predictor function would be represented in its vector form as $\mathbf{h}(\mathbf{x}, \boldsymbol{\theta})$.

Given a training set $\{(\mathbf{x}^{(l)}, \mathbf{y}^{(l)}); l = 1, ..., m\}$, a neural network needs to be trained in order to optimize the parameters θ and get a good predictor $h(\mathbf{x}, \boldsymbol{\theta})$ for the value of $\mathbf{y}$. One of the most popular techniques is to use the *gradient descent* algorithm.

The gradient descent algorithm tries to make the predictor $h(\mathbf{x}, \boldsymbol{\theta})$ close to $\mathbf{y}$. This is done by minimizing the *least-squares cost function*:

$$J(\theta) = \frac{1}{2} \sum_{l=1}^{m} (h(\mathbf{x}^{(l)}, \boldsymbol{\theta}) - y^{(l)})^2. \tag{3.1}$$

In order to choose the weights θ so as to minimize $J(\boldsymbol{\theta})$, the gradient descent algorithm starts off with some initial value and repeatedly performs the update:

$$\theta_{ij} := \theta_{ij} - \alpha \frac{\partial}{\partial \theta_{ij}} J(\boldsymbol{\theta}). \tag{3.2}$$

This update is done for each weight θ_{ij}. In the above equation, α is called the *learning rate*. It determines by how much the weights are changed each time we perform the update.

We have presented the artificial neural network which can be subjected to a supervised machine learning algorithm. In the next section we introduce the Markov Decision

Process (MDP), a mathematical structure that is different from neural networks, which can be subjected to an un-supervised or semi-supervised learning algorithm.

3.3 Reinforcement Learning

The term reinforcement learning is a procedure of teaching a learner to do a task by rewarding or penalizing its actions. It belongs to the semi-supervised family of machine learning as rewards do not provide direct reference to what to achieve and can occur later than the action taken. In the semi-supervised setting, the learner is not immediately told whether its current actions is wrong or right. Instead, the learner is only told how much its accumulated past actions are rewarded by the environment. In order to understand how it works, a short excursion into *Markov decision processes* is necessary.

3.3.1 Markov Decision Process

A *Markov decision process* (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ with the individual elements defined as follows:

- $\mathcal{S}$: Finite set of states $s \in \mathcal{S}$ of the environment
- $\mathcal{A}$: A finite set of actions $a \in \mathcal{A}$
- P: Probability of being in state s' after taking action a in state s. It is denoted as $P(s'|s, a)$.
- r: A reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with $r(s, a)$ being the reward of taking action a in state s. This general way of defining a reward function is also known as the *state-action reward*. The reward can also be written as a function of just the state with $r(s)$ being the reward in state s. This is known as the *state reward*.

A *stochastic policy* is a function $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ mapping from states to probabilities of selecting an action. The notation $\pi(s, a)$ is understood as the probability of taking action a in state s. A special case of the stochastic policy is where the probability of taking action a in state s is 1. This is known as the *deterministic policy* which is a function $\pi : \mathcal{S} \to \mathcal{A}$ mapping states to actions. The notation simplifies to $\pi(s)$ and we say $a = \pi(s)$.

Given a policy π, the state transition probabilities define a *Markov chain* that can also be classified by its accessibility of states under some policy (Puterman, 1994). State transition probabilities after k transitions will be denoted by $P_\pi^k(s'|s),\ s, s' \in S$.

3.3.2 MDP Application in Reinforcement Learning

In *Reinforcement Learning* (RL), the terms *learner* and *environment* are used to describe a learning process (Sutton and Barto, 1998; Ng et al., 2004; Watkins, 1989; Dayan and Watkins, 2001) . It is useful to define these terms for our purposes as we will use them in subsequent sections.

Definition 3.2. A *learner* is represented by a set of actions $a \in \mathcal{A}$ and its policy $\pi(s, a) \in P$ for an MDP=$\langle \mathcal{S}, \mathcal{A}, P, r \rangle$ that defines the environment and has state set S compatible with the learner's decisions and permits the use of the action set $\mathcal{A}$.

Sutton and Barto (1998) use the term agent instead of learner. We use the latter in our work to emphasize that it makes up a component of an agent, namely a component within the internal structure $\sigma \in \Sigma$. Figure 3.8 shows how a learner interacts with its environment. Let $s_t \in \mathcal{S}$ be the state that the learner is in at time t. The learner chooses an action $a_t \in \mathcal{A}$ from its policy $\pi(s_t, a_t)$ which causes it to be in state $s_{t+1} \in \mathcal{S}$ and receives a reward $r(s_{t+1}) \in \mathcal{R}$. The process is repeated until an *end state* is reached.

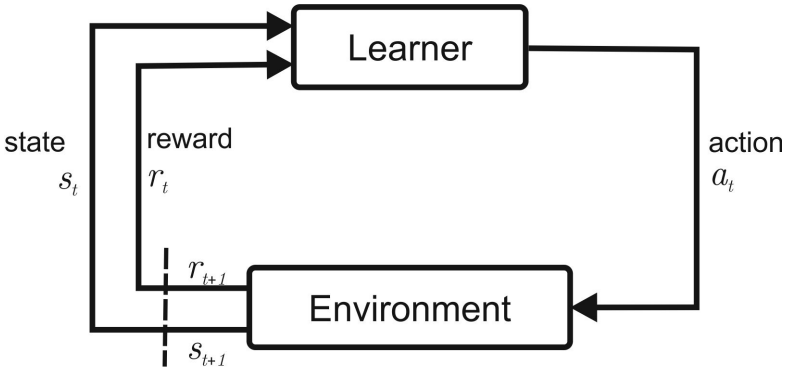


FIGURE 3.8: Learner environment interaction. Sutton and Barto (1998)

The goal in reinforcement learning is to choose actions over time so as to maximize the expected value of the *discounted reward*.

Definition 3.3. The discounted reward R_t at time t is the sum of all rewards starting from time t and weighted by a *discount factor* γ^t:

$$R_t = r(s_t) + \gamma r(s_{t+1}) + \cdots = \sum_{t=0}^{\infty} \gamma^t r(s_t). \tag{3.3}$$

The expected value of the discounted reward is given by

$$\mathrm{E}\{R_t\} = \mathrm{E}\{r(s_t) + \gamma r(s_{t+1}) + \cdots | s_t\}. \tag{3.4}$$

The value of the discount factor is usually between 0 and 1, $\gamma \in [0, 1]$. Note that the simpler state rewards form was used in the equations above. The same could be done for

state-action rewards. The discount factor γ is a way of stating that the value of a reward received in the future means less to us than if we were to receive the reward immediately. For $\gamma < 1$, acquiring positive rewards in the least amount of time is desired in order to make $\mathrm{E}\{R_t\}$ as large as possible. The discount factor is important in a reinforcement learning setting with *infinite horizon*. This means that we view the task as if it will go on forever. The opposite of this is the finite horizon setting. Here, we break the learning task into episodes and terminate at a horizon time t_T. In this work we will mainly use episodic reinforcement learning.

So far we have defined what an MDP is and how it is applied to reinforcement learning. We mentioned that the learner takes an action a through its policy $\pi(s, a)$, but it was not explained how the learner comes up with such a policy. This will be treated in the next section.

3.3.3 Direct Policy Search

The *direct policy search* reinforcement learning searches directly in the policy space $\mathcal{P}$. The learner's task is to find a good policy $\pi(s_t, a_t) \in \mathcal{P}$.

$$\sum_a \pi(s, a) = 1, \pi(s, a) \geq 0.$$

The concept of direct policy search can be explained with an example. Consider Figure 3.9 where a cart is shown with an inverted pendulum attached to it. The cart is allowed to take two actions, a_1 and a_2 of accelerating the cart right or left. Depending on the action chosen, the cart will accelerate right or left. The task is to move the cart in a direction while keeping the angle of the pole φ as much verticle as possible. In other words, we would not like the pole to fall over as the cart moves along in a direction. The cart receives a reward of -1 if the pole falls over. It receives a reward of 0 otherwise.

Let x and $\dot{x}$ be the linear diplacement and velocity of the cart. Let φ and $\dot{\varphi}$ be the angular displacement and velocity of the pole that we assume are all measurable by the learner. Let $\boldsymbol{\theta}$ be a set of weights that will be used as a parametrization of a policy. If we took $x, \dot{x}, \varphi, \dot{\varphi}$ as the features to represent the state $\mathbf{s}$, we have

$$\mathbf{s} = \begin{bmatrix} 1 \\ x \\ \dot{x} \\ \varphi \\ \dot{\varphi} \end{bmatrix}.$$

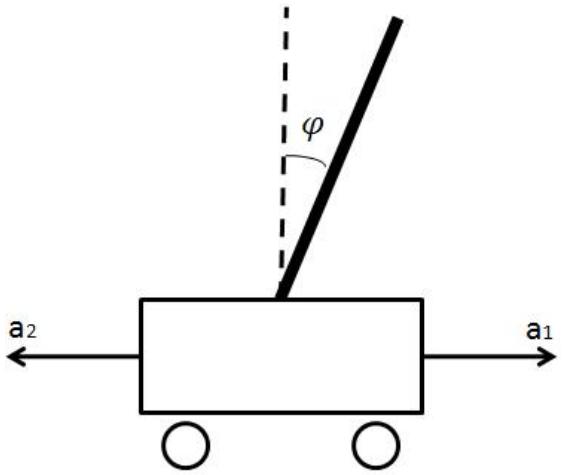


FIGURE 3.9: Pole balancing cart problem. Example adapted from Schaefer and Udluft (2005).

Note that the first feature of state $\mathbf{s}$ is a *bias* term 1. A stochastic policy $\pi_\theta : S \times \mathcal{A} \to \mathbb{R}$, mapping from states to probabilities of selecting an action, could then be defined using a *Bernoulli logistic* functions as follows:

$$\pi_\theta(\mathbf{s}, a_1) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{s})},$$
$$\pi_\theta(\mathbf{s}, a_2) = 1 - \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{s})},$$

with

$$\sum \pi_\theta(\mathbf{s}, a) = \pi_\theta(\mathbf{s}, a_1) + \pi_\theta(\mathbf{s}, a_2) = 1.$$

The goal is to maximize the expected reward $\mathrm{E}\{R_t\}$ with respect to the parameters $\boldsymbol{\theta}$:

$$\max_{\boldsymbol{\theta}} \mathrm{E}\{R_t\} = \max_{\boldsymbol{\theta}} \mathrm{E}\left[r(\mathbf{s}_t, a_t) + r(\mathbf{s}_{t+1}, a_{t+1}) \cdots | \pi_\theta, \mathbf{s}_t\right]. \tag{3.5}$$

The maximization with respect to $\boldsymbol{\theta}$ can be done using any supervised learning methods using global optimization of experimental sample averages of R_t for θ. Finding a control policy using a physical model in simulations is a kind of direct *policy search*.

Direct policy search algorithms are preferred when a simple policy class P that maps the features of the state to actions can be found. An example of which is a linear function or a logistic function. Suitable scenarios for this type of reinforcement learning include low level tasks such as steering a car or flying a plane (Ng and Jordan, 2000; Kohl and Stone, 2004).

In the next section, an algorithm from the unsupervised machine learning family will be presented as this is the approach needed in our main results.

3.4 Adaptive Resonance Theory

Adaptive resonance theory (ART) was developed to address the *stability - plasticity* dilemma (Grossberg, 1976; Tanaka and Weitzenfeld, 2002). The dilemma deals with the question of how a learning system can learn new events (plastic) and yet not effect its structure so as to preserve what it has already learned (stability).

The ART architecture is a *self organizing competitive* neural network which stems from the *competitive learning* model. It is a clustering algorithm to recognize patterns in data. In contrast to other clustering algorithms, the ART architecture is able create new clusters when a different pattern is encountered online. This is done without having to retrain the whole system. Since its introduction, different variations of ART have been developed, the most notable are ART1, ART2, ARTmap and FuzzyART (Grossberg, 1976; Carpenter and Grossberg, 1987; Carpenter et al., 1991) .

Our work utilizes the FuzzyART algorithm. We will first describe ART1 in order to show how a basic ART system works. We will then elaborate on FuzzyART which is a slight modification to the ART1 algorithm but overcomes its disadvantages. A brief introduction to wavelet transform is then given to complete the discussion.

3.4.1 ART1

An ART network is made up of two subsystems, an orienting subsystem and an attentional subsystem. The ART1 architecture is shown in figure 3.10.

The attentional subsystem is made up of four blocks: comparison block $F1$; recognition block $F2$; and the control gains blocks Gain1 and Gain2. The nodes in blocks $F1$ and $F2$ are called *short term memory* (STM) where the current input and outputs are stored. The two blocks are fully connected by *long term memory* (LTM) traces or weights which store the knowledge already learned. The weights θ_{ij} going from $F1$ to $F2$ are termed bottom-up weights (in fact "gains" applied to signals) whereas the weights θ_{ji} going from $F2$ to $F1$ are termed top-down weights. The orienting subsystem contains the reset block which controls the attentional subsystem overall dynamics based on the *vigilance parameter* ρ. This vigilance parameter determines the degree of mismatch that would be tolerated between the input pattern vectors and the weights connecting $F1$ and $F2$. The nodes at $F2$ represent the clusters formed. The top-down weights corresponding to each node in $F2$ represent the prototype vector for that node.

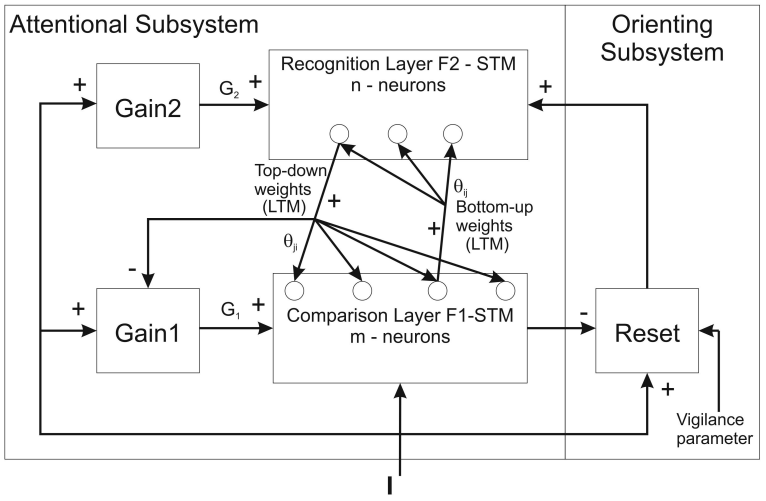
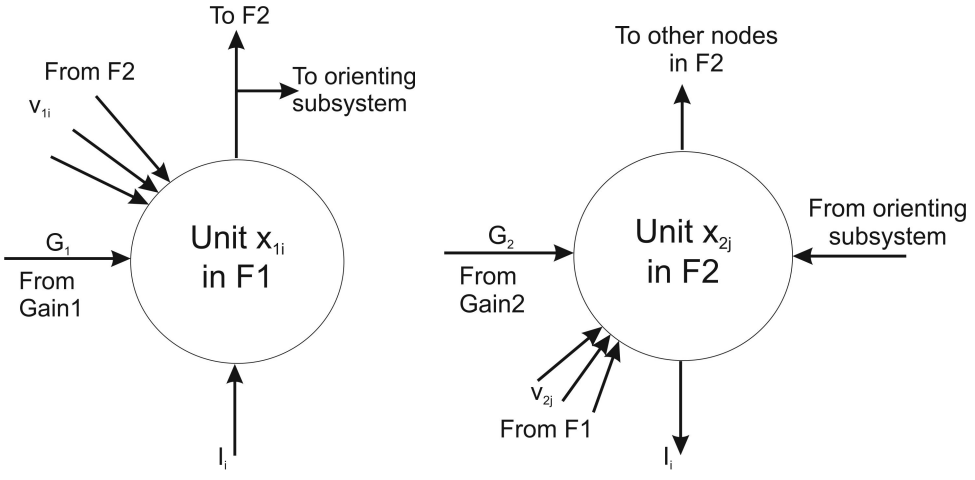


FIGURE 3.10: ART1 architecture consisting of attentional and orienting subsystem. Adapted from Tanaka and Weitzenfeld (2002).

An external binary input vector **I** is fed to $F1$. This input is passed to F2 where it is classified to a matching cluster. The result of this match is fed back to F1 where a comparison is made with the original input vector. If the output prototype vector from F2 and the input from F1 are relatively similar, the weights of the system are modified so that the prototype vector resembles the input vector. A new input vector **I** is fed to F1 and the cycle begins again. On the other hand, if there was a mismatch between F1 and F2, a reset signal is produced in the orienting subsystem which supresses the previously chosen category in F2. A new classification is carried out and the result is fed back to F1 for comparison. The cycle is repeated until a match is found between F1 and F2. If no match could be found between the two levels, this implies that the input vector **I** presented at F1 is significantly different from previous patterns. A new cluster is then created in F2. The gains G_1 and G_2 control the activity of the two layers respectively. A processing node in $F1$ and $F2$ are shown in figure 3.11.



(a) F1 processing node (b) F2 processing node

FIGURE 3.11: Processing nodes of F1 and F2

A processing node in $F1$ receives excitatory inputs from three sources: external binary input I_i; G_1 control signal; and prototype vector output v_{1i} from $F2$ multiplied by the corresponding connection weights θ_{ji}. The output of the node in $F1$ is fed to $F2$ as an excitatory output and to the orienting subsystem as a inhibitory output.

A processing node in $F2$ also receives excitatory inputs from three sources: Orienting subsystem; G_2 control signal;and output v_{2j} from node $F1$ multiplied by the corresponding connection weights θ_{ij}. The output of the node in $F2$ is fed to $F1$ as an excitatory output and to Gain1 block as an inhibitory output.

At any given time, only two out of the three inputs in $F1$ and $F2$ are used. The nodes in $F1$ and $F2$ are activated only if the two inputs are active. This is called the 2/3 rule. The sequence of processing an input is described below.

3.4.1.1 Recognition

When the input vector $\mathbf{I}$ is absent, the two gains G_1 and G_2 are equal to zero. This also disables all recognition nodes in $F2$ prior to the next recognition competition. When $\mathbf{I}$ is presented at the input, any component of $\mathbf{I}$ that is set to one will trigger both $G1$ and $G2$ and set their values to one. The control gains are formulated as follows:

$$G_1 = \begin{cases} 1 & \text{if } \mathbf{I} \neq 0 \text{ and } \mathbf{x}_2 = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

and

$$G_2 = \begin{cases} 1 & \text{if } \mathbf{I} \neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

Here, $\mathbf{x}_2 = [x_{21}, \ldots, x_{2j}, \ldots, x_{2n}]$ are the processing nodes in $F2$. The control gain G_1 depends on the vectors $\mathbf{I}$ and $\mathbf{x}_2$ of $F2$. The control gain $G2$ on the other hand only depends on the input vector $\mathbf{I}$.

An STM pattern activity is generated at each node in $F1$ that receives a nonzero input. Using the 2/3 rule, this results in an exact duplicate of the input at the output node of $F1$,

$$\mathbf{x}_1 = \mathbf{I} \text{ if } G_1 = 1. \tag{3.8}$$

The $F1$ output pattern $\mathbf{x}_1 = [x_{11}, \ldots, x_{1i}, \ldots, x_{1m}]$ is multiplied by the connecting LTM weights θ_{12}. At $F2$, each node calculates the value

$$v_{2j} = \frac{\sum_i x_{1i}\theta_{ij}}{\sum_i \theta_{ij}}. \tag{3.9}$$

The output of $F2$ is the node x_{2j} which best matches the input $\mathbf{I}$. This is determined by v_{2j} which receives the biggest (i.e. most matching) $F1$ output,

$$x_{2j} = \begin{cases} 1 & \text{if } G_2 = 1 \cap v_{2j} = \max_k v_{2k}, \forall k \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

When the winner of $F2$ node is activated, its value is set to one and inhibits all other nodes in the $F2$ block. The value of all other nodes in $F2$ is set to zero.

3.4.1.2 Comparison

The STM activity on $F2$ generates a pattern output vector $\mathbf{x}_2$. If at least one component in $\mathbf{x}_2$ is active , it inhibits the gain G_1 and forces its value to zero. The pattern $\mathbf{x}_2$ is multiplied by the connecting LTM weights θ_{21}. This product is sent to $F1$ to be compared with the original input $\mathbf{I}$. Each node in $F1$ calculates the value

$$v_{1i} = \frac{\sum_j x_{2j}\theta_{ji}}{\sum_j \theta_{ji}}. \tag{3.11}$$

Using the 2/3 rule, The top-down vector $\mathbf{v}_1$ is compared with $\mathbf{I}$ for each component. The node output x_{1i} on $F1$ is determined by

$$x_{1i} = \begin{cases} 1 & \text{if } I_i = 1 \cap v_{1i} = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

If the match between the two vectors is significant (in terms of the vigilance parameter defined below), the system stabilizes and learning can be done. If there is a mismatch between the two, this implies that the recognition class v_{2j} chosen was not correct and therefore should be inhibited.

3.4.1.3 Search

The reset block in the orienting subsystem measures how similar $\mathbf{I}$ and $\mathbf{x}_1$ are. If the degree of similarity between the two is less than ρ, a reset signal is triggered that

inhibits the previously chosen recognition class v_{2j} in $F2$ for the remainder of the current classification cycle. Here $0 < \rho \leq 1$ is a vigilance parameter.

An input pattern mismatch is triggered when

$$\frac{\|\mathbf{x}_1\|}{\|\mathbf{I}\|} < \rho. \tag{3.13}$$

The vigilance parameter determines the degree of mismatch that would be tolerated before a reset signal is triggered. A vigilance value which is close to one would result in the system having a large number of classes since it forces $\mathbf{I}$ and $\mathbf{x}_1$ to have very similar values. A value closer to zero would result in fewer number of classes since the system tolerates a bigger difference in similarity.

When the active $F2$ is inhibited, the top-down vector v_1 is removed and the original pattern x_1 on $F1$ is regenerated. This causes the orienting subsystem to cancel the reset signal and the bottom-up activation starts again. A different $F2$ node is chosen and a different pattern is passed back to $F1$ for comparison. This process is repeated until no reset is generated and a match is found. If no match is found, a new node in $F2$ is generated which becomes a learned new class.

3.4.1.4 Learning

The learning process occurs only after a match has been found and the search process has ended. This is due to the speed at which the recognition, comparison and search process is executed. The LTM traces θ_{21} and θ_{12} between $F1$ and $F2$ are too slow to execute an update during this process. The LTM weights from $F1$ to $F2$ are updated as follows:

$$\tau_1 \frac{d\theta_{ij}}{dt} = \begin{cases} (1-\theta_{ij})L - \theta_{ij}(\|\mathbf{x}_1\| - 1) & \text{if } v_{1i} \text{ and } v_{2j} \text{ are active} \\ -\|\mathbf{x}_1\|\,\theta_{ij} & \text{if only } v_{2j} \text{ is active} \\ 0 & \text{if only } v_{2j} \text{ is inactive.} \end{cases} \tag{3.14}$$

In the equation above, L is a parameter which has a value greater than one and τ_1 is a time constant. The above equation is called a slow learning equation. A simpler version, called the fast learning equation, is given as:

$$\theta_{ij} = \begin{cases} \dfrac{L}{L - 1 + \|\mathbf{x}_1\|} & \text{if } v_{1i} \text{ and } v_{2j} \text{ are active} \\ 0 & \text{if only } v_{2j} \text{ is active.} \end{cases} \tag{3.15}$$

The initial value of θ_{ij} is chosen randomly that satisfies the following:

$$0 < \theta_{ij} < \frac{L}{L - 1 + \|m\|}. \tag{3.16}$$

The variable m is equal to the number of nodes in $F1$. The top-down LTM weights from $F2$ to $F1$ are updated with

$$\tau_2 \frac{d\theta_{ji}}{dt} = x_{2j}(-\theta_{ji} + x_{1i}). \tag{3.17}$$

The variable τ_2 is once again a time constant. As with the bottom-up equation, the fast learning version of the above is given as

$$\theta_{ji} = \begin{cases} 1 & \text{if } v_{1i} \text{ and } v_{2j} \text{ are active} \\ 0 & \text{if only } v_{2j} \text{ is active.} \end{cases} \tag{3.18}$$

The value of θ_{ji} is initially set randomly according to the inequality

$$1 \geq \theta_{ji} > c \tag{3.19}$$

where c is decided by the slow learning equation parameters.

The disadvantage of ART1 is that it is only able to process binary inputs. A version called FuzzyART developed by the same group of researchers rectifies this problem. The details of FuzzyART is explained next.

3.4.2 FuzzyART

The FuzzyART system integrates computations from fuzzy set theory into ART1. In fuzzy set theory, a membership function assigns an object a grade of membership to a certain class (Zadeh, 1965). We will make use of the fuzzy operators $\wedge$ and $\vee$ in ART1. They are defined as

$$(x \wedge y)_i \equiv \min(x_i, y_i), \tag{3.20}$$

and

$$(x \vee y)_i \equiv \max(x_i, y_i). \tag{3.21}$$

By making a slight modification to the ART1 equations, analog input patterns can be processed. Instead of taking the inner product of two variables, we insert the min and max operator into the ART1 equations as shown in Table 3.1.

ART1 (Binary)	FuzzyART (Analog)
Category Choice	
$v_{2j} = \frac{\sum_i x_{1i}\theta_{ij}}{\sum_i \theta_{ij}}$	$v_{2j} = \frac{\sum_i \min(x_{1i}, \theta_{ij})}{\sum_i \theta_{ij}}$
Match Criterion	
$\frac{\sum_i I_i v_{1i}}{\sum_i I_i} < \rho$	$\frac{\sum_i \min(I_i, v_{1i})}{\sum_i I_i} < \rho$
Weight Update	
$\theta_{ij} = \begin{cases} \frac{L}{L-1+\|\mathbf{x}_1\|} \\ 0. \end{cases}$	$\theta_{ij} = \min(\theta_{ij}, I_i)$
$\theta_{ji} = \begin{cases} 1 & \text{if } v_{1i} \text{ and } v_{2j} \text{ are active} \\ 0 & \text{if only } v_{2j} \text{ is active.} \end{cases}$	$\theta_{ji} = \min(\theta_{ji}, I_i)$

TABLE 3.1: FuzzyART equations.

The choice function v_{2j} tells us to what extent the weight vector θ_{ij} is a fuzzy subset of the input vector $\mathbf{I}$. If

$$\frac{\sum_i \min(I_i, \theta_{ij})}{\sum_i \theta_{ij}} = 1, \tag{3.22}$$

then category j is said to be a *fuzzy subset choice* for $\mathbf{I}$.

FuzzyART requires that the input $\mathbf{I}$ be normalized. This can be done by preprocessing the incoming vector. Other than normalizing the input, the preprocessing stage extracts features from a dataset. A method of extracting features can be done using the discrete wavelet transform. In the next section, the discrete wavelet transform is introduced.

3.5 Discrete Wavelet Transform

The wavelet transform addresses the issue of capturing information of both the frequency components of a non-stationary signal and the time that it occurs (Kaiser, 1994). A wavelet transform "cuts up data of functions or operators into different frequency

components, and then studies each component with a resolution matched to its scale" (Daubechies, 1992). There are two types of wavelet transforms; the *continuous wavelet transform* (CWT) and the *discrete wavelet transform* (DWT). In our work, the latter is of interest.

The CWT of a signal results in an infinite number of coefficients on the time-scale domain. This would lead to a very high computation time and would be impossible to implement. The discretized continuous wavelet transform provides a sampled version of the CWT. However, it is not a true discrete transform. The discrete wavelet transform or DWT on the other hand, is easier to implement and provides a significant reduction in computation time. It generates sufficient information for the analysis of the original signal. In DWT, the scale and position parameters are discretized as

$$s = s_0^m, \tag{3.23}$$

and

$$\tau = k\tau_0 a_0^m. \tag{3.24}$$

Here, m and k are integers. These parameters are normally *dyadic* (powers of two), so $s_0 = 2$ and $\tau_0 = 1$. Therefore, $s = 2^m$ and $\tau = k2^m$. The discrete wavelet transform is written mathematically as

$$\tilde{x}(m,k) = \langle \psi_{m,k}, x \rangle = \frac{1}{\sqrt{|s_0^m|}} \int_{-\infty}^{\infty} x(t)\psi^* \left(\frac{t - k\tau_0 a_0^k}{a_0^m} \right) dt. \tag{3.25}$$

In practical applications, the above equation is implemented using lowpass and highpass filters. These filters are called scaling and wavelet function respectively. The DWT decomposes the signal or sequence $x[n]$ into a coarse and a detailed approximation by passing it successively through a scaling function (lowpass filter) $h[n]$ and a wavelet function (highpass filter) $g[n]$. It is analyzed at different frequency bands with different resolutions. The scaling and wavelet functions are half band digital filters where frequencies above/below half the maximum frequency are removed. For example, if we had 20 radians as the maximum frequency, a half band lowpass filter would remove all frequencies above 10 radians (for discrete signals, frequency is expressed in terms of radians).

In DWT, we use the word resolution instead of translation. The resolution of a signal tells us what detail information is in the signal. The resolution of the signal is changed by the filtering operation.

The scale of a DWT is changed by upsampling and downsampling the data. When the signal is downsampled, the sampling rate is reduced and samples are dropped from the signal. On the other hand, upsampling a signal increases the sampling rate and adds new samples to the signal.

After passing through $h[n]$ and $g[n]$, the signal is split into a high frequency component, y_h and low frequency component y_l. The high frequency component y_h is better known as the *detail* of the signal whereas y_l is known as the *approximation* of the signal. Every other sample from these components are discarded. This is possible due to the Nyquist sampling criteria,

$$f_s > 2f_c. \tag{3.26}$$

Here, f_s is the sampling frequency and f_c is the highest frequency component of the signal. After going through the lowpass filter, the highest frequency of y_l is $f_c/2$. This makes some of the samples redundant and therefore can be discarded. The same can be argued for y_h. Using the bandpass sampling theorem,

$$f_s > 2(f_c - \frac{f_c}{2}). \tag{3.27}$$

The high and low frequency components of signal x can be written mathematically as

$$y_h[k] = \sum_n x[n] \cdot g[2k - n], \tag{3.28}$$

$$y_l[k] = \sum_n x[n] \cdot h[2k - n]. \tag{3.29}$$

One important thing to note is that $g[n]$ and $h[n]$ are not independent and are related by

$$g[L - 1 - n] = (-1)^n \cdot h[n]. \tag{3.30}$$

Here, L is the filter length. These types of filters are known as the *Quadrature Mirror Filters* (QMF). There are many filters that one can choose to implement. One such filter is the Daubechies' scale and wavelet functions (Kaiser, 1994).

The *subband coding* procedure above can be repeated for further decomposition. Each time a filtering and a subsampling process is done, it results in half the number of samples and half the frequency band spanned. Decomposing the original signal into frequency components as above will decrease the time resolution by half, since only half

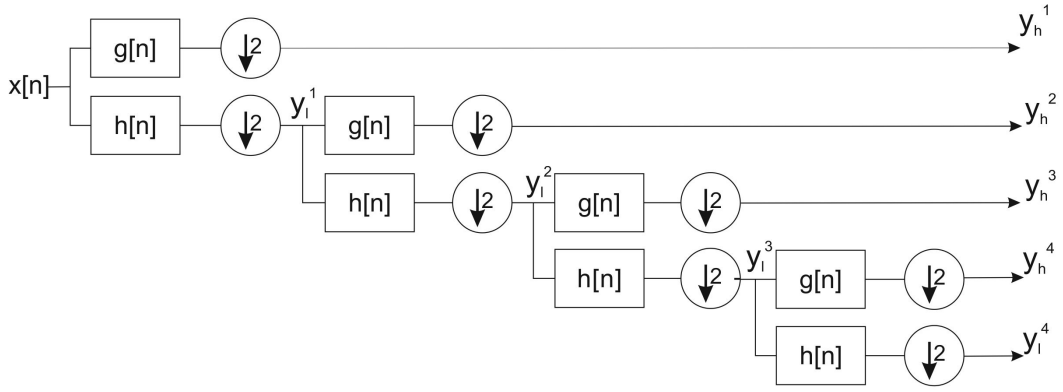


FIGURE 3.12: Decomposing signal $x[n]$ through subband coding. After every filtering level, output is subsampled (downsampled) by 2. The details y_h^i, where i is the level, give the DWT coefficients for that level.

the number of samples can be used to characterize the original signal. Subsampling the components increases the scale since the frequency band spanned is now halved. We say it increases the frequency resolution. Figure 3.12 shows how the signal is decomposed.

After every filtering level i, y_h^i or the detail gives the DWT coefficient for that level. The low frequency component y_l^i or approximation is further decomposed to get y_h^{i+1} and y_l^{i+1}. For many signals, the approximation or low frequency component is of interest since it holds the identity of the signal. The detail or high frequency component contains the nuance or flavour of the signal.

3.6 Chapter Conclusion

The elements that are used in this work were defined in this chapter. An agent is defined as a tuple $A = \langle \Sigma, \mathcal{X}, \mathcal{K} \rangle$ where $\sigma \in \Sigma$ is the internal architecture, $\chi \in \mathcal{X}$ is the communication set and $\kappa \in \mathcal{K}$ is the resource set. Agents are calssified according to their properties. In our work, we deal exclusively with reactive and learning agents.

A reactive agent's internal architecture is made up of conditional logic rules which maps sensors directly to actions. The internal architecture of a learning agent on the other hand involves implementing machine learning algorithms and slow changing of reactive behaviour.

Several algorithms from the supervised, semi-supervised and unsupervised machine learning family were presented in this chapter. A popular computational component capable of supervised learning, called artificial neural network, is made up of nodes. They are arranged in layers. Each node in a layer is connected to a preceding layer through weighted links. The input signal to the network appears at the input layer. This signal is propagated forward through the network until it reaches the final layer which is the network output. The intermediate layers are called the hidden layer.

The reinforcement learning algorithm belongs to the semi-supervised machine learning family. Reinforcement learning has its roots in Markov decision processes. A *Markov decision process* (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, P, r \rangle$.

A *stochastic policy* is a function $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ mapping from states to probabilities of selecting an action. The notation $\pi(s, a)$ is understood as the probability of taking action a in state s.

It was detailed how MDP's are applied in reinforcement learning. A learner is represented by a set of actions $a \in \mathcal{A}$ and its policy $\pi(s, a) \in P$ for an MDP=$\langle \mathcal{S}, \mathcal{A}, P, r \rangle$ that defines the environment and has state set S compatible with the learner's decisions and permits the use of the action set $\mathcal{A}$. Direct policy search reinforcement learning was discussed as a method of finding the optimal policy $\pi^*(s, a)$. It searches directly in the policy space P.

The ART network is a self organizing competitive neural network. It belongs to the unsupervised category of machined learning. It is a clustering algorithm to recognize patterns in data. In contrast to other clustering algorithms, the ART architecture is able to create new clusters when a different pattern is encountered online.

The FuzzyART algorithm was introduced which overcomes the disadvantage of ART1 in that it can only analyze binary data. The min and max operator from fuzzy set theory replaces the inner product operator in the ART1 equations. What results is a stable learning algorithm that is able to process both binary and analog inputs.

The wavelet transform was presented as a foundation to understanding the process of extracting features from a dataset. A signal is analyzed at different frequencies with different resolutions. The terms time resolution and frequency resolution are often used to describe how well the information is captured in the time and frequency domains respectively.

The discrete wavelet tranform (DWT) is implemented using lowpass and highpass filters. These filters are called scaling and wavelet function respectively. The DWT decomposes the signal or sequence $x[n]$ into a coarse and a detail approximation by passing it successively through a scaling function (lowpass filter) $h[n]$ and a wavelet function (highpass filter) $g[n]$. It is analyzed at different frequency bands with different resolutions. The scaling and wavelet functions are half band digital filters where frequencies above/below half the maximum frequency are removed.

Chapter 4

Multiagent Reconfigurable Control Theory

This chapter presents the approach that was taken in our work. This research considers the design of an autonomous reconfigurable control system for ground vehicles that will hopefully generalize to other applications. It will try to address the problem of integrating the FDD and reconfiguration components of a reconfigurable control system. A multiagent architecture will be the basis of our approach.

4.1 Novelty of proposed approach

In this work, a novel approach is proposed. A multiagent architecture will be used to interface between the two components. It is a relatively unexplored area in reconfigurable control systems for vehicles. Research done in agent based control has been towards modular robots and industrial applications such as electric power distribution systems and process plant production. This method that we follow will allow the system to be viewed as a modular structure.

Six agents from the reactive and learning agents categories are defined in our work. They are labelled *planner agent* A_p, *monitor agent* A_m, and *control agent* A_c. Table 4.1 describes which category they belong to.

Name	Symbol	Number	Category
planner	A_p	1	reactive
monitor	A_m	1	learning
control	A_c	4	learning

TABLE 4.1: Agent category

The agents defined in this work are used to solve the integration of the FDD and CR components in a reconfigurable control system. The planner agent A_p and the control

agents $A_c^k, k = 1, \ldots, 4$ take the role of the reconfiguration component. The monitor agent A_m on the other hand takes the role of the FDD component.

The following auxiliary definitions will be needed to complete our assumptions on agents.

We define $\mathcal{B} = \{b_1, \ldots, b_j\}$, where j is an integer, to be the set of parameter settings of a system. These settings could be the parameters for the sensors, or it could define resource combinations for a certain operating mode.

We define $\mathcal{C} = \{c_1, \ldots, c_i\}$, where i is an integer, to be the set of controllers that are available to a system. These are either the controllers learned by the control agent or a predefined controller set by the designer.

We define $\mathcal{D} = \{d_{s1}, \ldots, d_{sl}, d_{a1}, \ldots, d_{an}\}$, where l and n are integers, to be the set of low level sensors and actuators such as wheel encoders and motors.

We define $\mathcal{F} = \{f_1, \ldots, f_m\}$, where m is an integer, to be the set of fault states of a system.

We define $\mathcal{G} = \{g_1, \ldots, g_o\}$ where o is an integer to be the set of messages that can be passed throughout the system.

We define the following message set $\mathcal{G}$ to facilitate synchronization between agents. We let $g_1 =$ "*learning_status*", $g_2 =$ "*task_enable_command*" and $g_3 =$ "*task_disable_command*".

4.2 Planner Agent

The planner agent $A_p = \langle \Sigma_p, \mathcal{X}_p, \mathcal{K}_p \rangle$ belongs to the reactive agent category. Its members are defined as follows:

A_p must have the ability to disable and enable other agent's task in order for it to control the overall system. The communication set $\chi_p \in \mathcal{X}_p$ contains a subset of a simple message set $\mathcal{G}$. We let $\chi_p = \{g_2, g_3\}$.

Furthermore, the planner agent must know the fault status in order for it to choose a different strategy. Choosing a different strategy involves searching through a storage of system parameter settings and stored controllers. For the resource set $\kappa_p \in \mathcal{K}_p$, we let $\kappa_p = \langle \mathcal{F}, \mathcal{B}, \mathcal{C}, \mathcal{D} \rangle$.

The internal architecture $\sigma_p \in \Sigma_p$ maps its sensor data directly to actions using a rule based conditional logic. The planner agent has to have the capability to choose a strategy or a solution depending on its perception of the state of its environment. It communicates this strategy with the control agents for it to be implemented. It needs to be able to store predefined and new solutions in its memory bank. The planner agent

must also be able to change the parameters of a control sytem to adapt to faults. Figure 4.1 shows the internal structure of a control agent A_c^k.

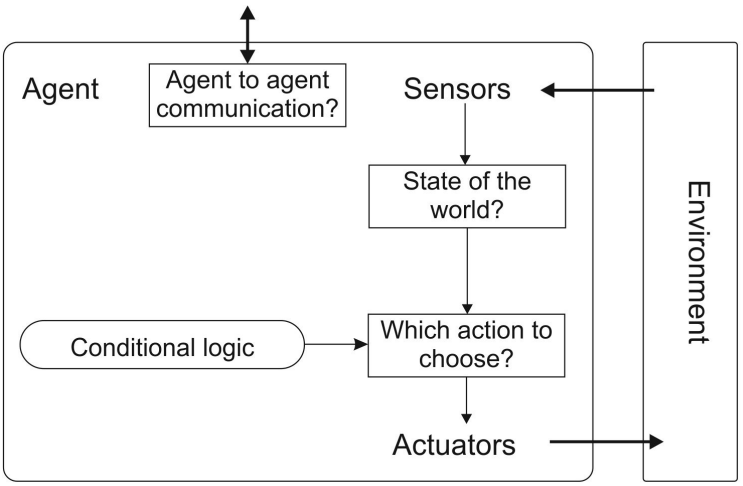


FIGURE 4.1: Internal architecture of planner agent A_p. Conditional logic is used to decide on which startegy to use.

We define two types of memory for the planner agent. The first is called a *resource memory* $\mathcal{M}_r$ and the second is called a solution memory $\mathcal{M}_s$. The memory of the planner agent is assigned as follows:

$$\mathcal{M}_r = \mathcal{B}, \tag{4.1}$$

and

$$\mathcal{M}_s = \mathcal{C}. \tag{4.2}$$

The planner agent A_p perceives the actions of the monitoring agent A_m as its states. It then decides what to do depending on the state. The planner agent's actions are the choice of controllers and system settings to use. The controller c_i is passed to the control agents A_c^k (where k is the agent index) to be implemented. The system is parameterized according to b_j.

4.3 Control Agent

The control agent $A_c = \langle \Sigma_c, \mathcal{X}_c, \mathcal{K}_c \rangle$ belongs to the learning agent category. Its members are defined as follows.

A_c must be able to communicate whether it is learning a new controller so that the process is not interrupted by other agents. Following this criteria, the communication set $\chi_c \in \mathcal{X}_c$ is assigned as $\chi_c = \{g_1\}$.

The control agents also need to know which sensors and actuators are available to them in learning and executing a control task. For the resource set $\kappa_c \in \mathcal{K}_c$, we let $\kappa_c = \langle \mathcal{D}, \mathcal{F} \rangle$.

In our reconfiguration problem, four control agents A_c from the learning agent category were defined. A method to allow the agents to communicate and achieve a common goal is therefore needed. In Section 3.3.3, the direct policy search algorithm was introduced. In our work, we extend the algorithm to the multiagent setting. The internal architecture $\sigma_c \in \Sigma_c$ of a A_c implements a *multiagent direct policy search* algorithm. The details of which are explained in the following.Figure 4.2 shows the internal structure of a control agent A_c^k.

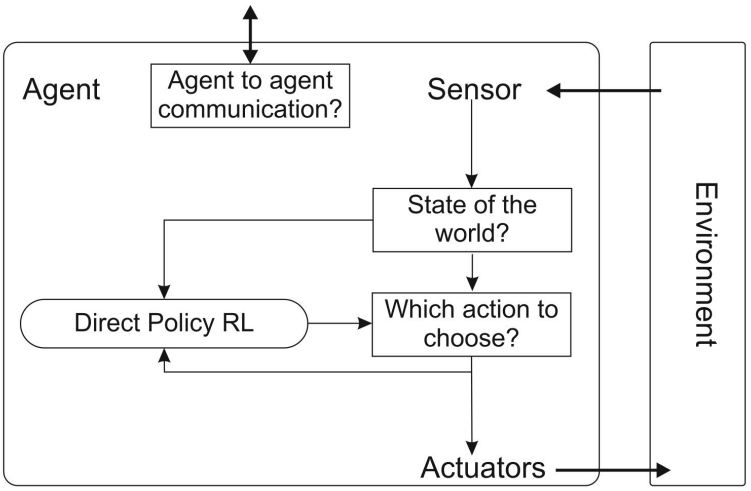


FIGURE 4.2: Internal architecture of one control agent A_c^k. The diagram shows direct policy search being used to decide which actions to take.

Some researchers have extended direct policy search reinforcement learning to the multiagent setting (Ma and Cameron, 2009; Wu et al., 2011). However, they have all assumed that the states and actions are discrete and finite. This is seldom the case in practical applications. For example, the reinforcement learning states of the autonomous vehicle presented in the example to follow are represented as a vector in $\mathbb{R}^5$ comprising its linear and angular velocities (v_x, ω), the error between the reference and actual value of the velocities (e_{v_x}, e_ω). The action set is represented as a subset of $\mathbb{R}^4$ since the vehicle has four independent PWM input currents for each motor driving a wheel.

In our work we extend the multiagent direct policy search reinforcement learning to allow for continuous states and actions. We have defined four control agents A_c^k, $k = 1, \ldots, 4$. These agents need to take into account the actions of other agents in order to complete the control task. In the following we use *game theory* to explain how the control agents interact (Bowling and Veloso, 2002).

		Column Player	
		LEFT	RIGHT
Row Player	UP	3 , 3	0 , 5
	DOWN	5 , 0	1 , 1

FIGURE 4.3: Two player strategic game. The first number in each box is the reward for *row player*. Second number in each box is the reward for *column player*. Action set $\mathcal{A}_r$ for *row player* is $\mathcal{A}_r = \{\text{'UP'}, \text{'DOWN'}\}$. Action set $\mathcal{A}_c$ for *column player* is $\mathcal{A}_c = \{\text{'LEFT'}, \text{'RIGHT'}\}$. If *row player* chooses the action $a_r =$ 'UP' and *column player* chooses action $a_r =$ 'RIGHT', then the reward for *row player* $r_r = 0$ whereas reward for *column player* $r_c = 5$

4.3.1 Game theoretical formulation

A *matrix game* or *strategic game* for our control agents is a multi player, single state framework. The agents and their reward functions form a tuple $\langle \mathcal{A}_1, \ldots, \mathcal{A}_n, r_1, \ldots, r_n \rangle$ where n=4 in our case study.

$\mathcal{A}_k$: A set of actions available to a control agent k (here $k = 1, \ldots, n$). The *joint action* set is $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$

r: A reward function for control agent k. $r_k : \boldsymbol{\mathcal{A}} \to \mathbb{R}$.

Each control agent chooses an action from its own action set and receives a reward depending on all other control agents' actions. Figure 4.3 illustrates a simple matrix game for two agents or players.

The control agents choose actions according to the strategic game they are playing. The types of strategic games that they can "play" are as follows.

1. Fully cooperative: A game where all players have the same reward function ($r_1 = r_2 = \cdots = r_n$). All players have the same goal. It is also known as a *general-sum* game.
2. Fully competitive: The sum of rewards is zero in a two player game, they have opposite goals ($r_1 = -r_2$). It is also known as a *zero-sum* game.
3. Mixed: Games where players are neither fully cooperative nor fully competitive.

An important concept used in game theory is the *Nash equilirium*. A basic definition of the Nash equilibrium is given below for our control agents. We will formulate it mathematically in the next section when we discuss stochastic games.

Definition 4.1. If there is a set of strategies for a game where no player can benefit by changing its strategy while other players' strategies remain unchanged, then that set and its corresponding rewards is called a nash equilibrium (Tuyls,2005).

The Nash equilibrium for the example given in figure 4.3 is the joint action set $\mathbf{a} = \{a_r = \text{'DOWN'}, a_c = \text{'RIGHT'}\}$ since the strategy chosen for each player guarantees that it will receive a reward of at least 1 point regardless of what the other player does. This is better then taking a chance of playing a different strategy and getting 0 points.

4.3.2 Stochastic game

A *stochastic game* is an extension of the strategic game to Markov decision processes where there are multiple states. We can view each state of the MDP to be a strategic game where all agents chooses an action, receives a reward and proceeds to the next state. It is defined as a tuple $\langle \mathcal{S}, \mathcal{A}_1, \ldots, \mathcal{A}_n, P, r_1, \ldots, \mathcal{R}_n \rangle$ where the agents can be our control agents in which case n=4. The index n is the number of agents. The elements in the tuple have the following meaning:

$\mathcal{S}$: Finite set of states of the environment $s \in \mathcal{S}$

$\mathcal{A}_k$: A set of actions available to an agent k (here $k = 1, \ldots, n$). The *joint action* set is $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$

P: Probability of being in state s' after taking the joint action $\mathbf{a} \in \boldsymbol{\mathcal{A}}$ in state s. $P : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \to [0, 1]$.

r: A reward function for agent A^k. $r_k : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \to \mathbb{R}$.

The policy, $\pi^k : \mathcal{S} \to \mathcal{A}_k$, maps states to agent A^k's actions. A *joint policy* $\boldsymbol{\pi}$ is a collection of all agents' policies π^k for $k = 1, \ldots, n$. We would like to find a joint policy $\boldsymbol{\pi}^*$ that maximizes all agents' rewards.

To analyze the joint work of control agents we will concentrate on fully cooperative tasks. The rewards are equal for each agent, $r_1 = r_2 = \cdots = r_n$. We will make use of Nash equilibrium to find the optimal joint policy $\boldsymbol{\pi}^*$. The following defines Nash equilibrium mathematically for the agents.

Definition 4.2. A Nash equilibrium is a joint policy $\boldsymbol{\pi}^*(s, \mathbf{a}) = \left\{\pi^{1*}, \ldots, \pi^{n*}\right\}$ such that an agent A^k's policy π^{k*} is the best response to other agents' $A^i(\forall k \neq i)$ policies (i.e. no agent can take better action for itself given the policies of the other agents).

$$E\left\{r_k|\pi^1,..,\pi^{k*},..,\pi^n\right\} \geq E\left\{r_k|\pi^1,..,\pi^k,..,\pi^n\right\} \forall \pi_k \tag{4.3}$$

The definition of Nash equilibrium requires that each agent's strategy is a best response to the other agents' strategy. The following theorem by Filar and Vrieze (1996) shows that there has to be at least one Nash equilibrium in stationary strategies for any stochastic game.

Theorem 4.3. *Every general-sum discounted stochastic game possesses at least one equilibrium point in stationary strategies.*

A multiagent direct policy search reinforcement learning algorithm (MADPRL) will be implemented for each control agent A_c to learn the optimal policy π^{k*}.

It was assumed in our work that the state-action sets could be continous and infinite. This would require an infinite amount of memory and time to account for each state-action pair. We will need a way to represent the states and actions without exhausting computer resources. A solution to this problem is by using *function approximation* techniques to generalize over the states and actions.

We first define $\boldsymbol{\pi}(\mathbf{s}_t, a_t)$ to be a vector of policies at time t as follows:

$$\boldsymbol{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \begin{bmatrix} \pi^1(\mathbf{s}_t, a_t^1) \\ \pi^2(\mathbf{s}_t, a_t^2) \\ \pi^3(\mathbf{s}_t, a_t^3) \\ \pi^4(\mathbf{s}_t, a_t^4) \end{bmatrix} \tag{4.4}$$

Each policy corresponds to one control agent. We assume that all agents share the same state $\mathbf{s}_t$. Each policy $\pi^k(\mathbf{s}_t, a_t^k)$ has the following form:

$$\pi_c^k(\mathbf{s}_t, a_t^k) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(a_t^k - \bar{a}_t^k(\mathbf{s}_t))^2\right) \tag{4.5}$$

In the equation above, σ is the standard deviation. It is important to note the overloaded use of the symbol π in the left and right hand side of the above equation. In the left hand side, the symbol $\pi_c^k(\mathbf{s}_t, a_t^k)$ is used to denote a policy. In the right hand side of the equation, the symbol π is used to denote a constant. The mean or the current approximate optimal value is represented by $\bar{a}_t^k(\mathbf{s}_t)$. The actual action a_t^k is the output chosen by sampling around $\bar{a}_t^k(\mathbf{s}_t)$ given by a *function approximator*. Figure 4.4 illustrates how an action for a control agent is sampled.

To learn the control actions a three layer back propagation neural network as described in Section 3.2 is used as a function approximator for each agent A_c^k. The neural network is shown in Figure 4.5.

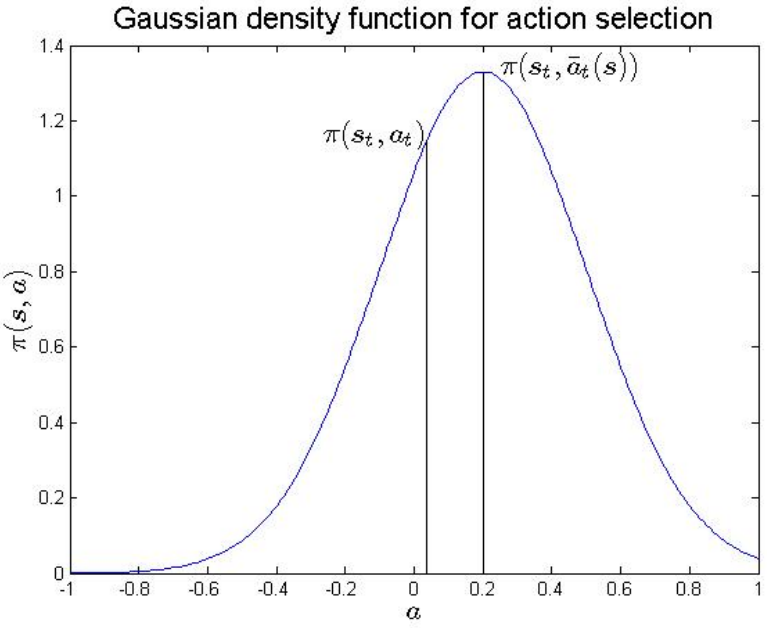


FIGURE 4.4: Gaussian density function for action selection. a_t is sampled around the mean $\bar{a}_t$.

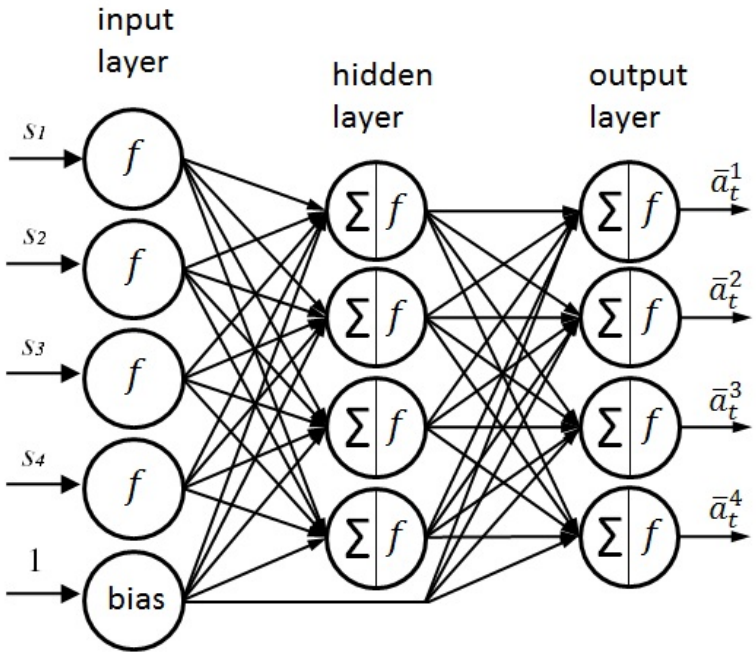


FIGURE 4.5: Three layer backpropagation neural network with 5 inputs, 4 hidden units and 4 outputs corresponding to the control agents' output.

The network consists of four inputs $s_1, \ldots, s_4$ corresponding to the reinforcement learning states that will be illustrated in Chapter 6 plus a bias term 1, four hidden units and four outputs $\bar{a}_t^1, \ldots, \bar{a}_t^4$ corresponding to four control agents $A_c^1, \ldots, A_c^4$. The control agents' output can be represented in this way since we have assumed that all control agents share the same states. Since neural networks are part of the supervised learning family, training examples must be provided to the network in order for it to learn. Training examples here mean examples of a good action to be taken. In our work however, we assume we have no knowledge of what a good action is. We have to *explore* the right actions to take. A natural question that arises is how do we provide training examples to the network?

To learn training examples for the neural networks, we adopt a temporal difference (TD) approach to the problem. We proceed as follows. At each time step t and for every k, we sample a value from the policy function $\pi(\mathbf{s}_t, a_t^k)$ around the mean $\bar{a}_t^k(\mathbf{s}_t)$ to get a_t^k. The action a_t^k is executed and we obseve the reward obtained. If the reward obtained at time step $t+1$ is bigger than the reward obtained at time step t, we feed a_t^k back to the network and use it as a training example. Otherwise, no update is done. The parameters of the network are updated to get a new current approximate optimal action $\bar{a}_{t+1}^k(\mathbf{s}_t)$ which is edged closer to be more like a_t^k. A factor α which is the learning rate is included in the update rule to control the speed of learning. The output $\bar{a}_{t+1}^k(\mathbf{s}_{t+1})$ of our neural network is then used as the new mean in $\pi(\mathbf{s}_{t+1}, a_{t+1}^k)$. This process is repeated until the policy $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$ converges to the optimal policy $\boldsymbol{\pi}^*(\mathbf{s}, \mathbf{a})$. The error signal for the output layer, denoted by δ_o, is given by

$$\delta_o = -(a_t^k - \bar{a}_t^k(\mathbf{s}_t)).$$

This error is propagated back through the network so that the hidden units can be updated. The weights of the network are updated with the following rule:

$$\theta_{ij,t+1}^{\bar{a}_k} = \theta_{ij,t}^{\bar{a}_k} + \alpha\delta_j\delta_r b_i \quad \text{if } \delta_r \geq 0. \tag{4.6}$$

In the above equation, $\delta_r = r_{t+1} - r_t$ is difference between the reward obtained at time step $t+1$ and t; the weight between node i and node j is denoted as θ_{ij} (refer to Figure 3.4). The output of node i is denoted as b_i and the error signal for node j is indicated by δ_j.

All control agents share the same reward function r_t. Maximizing individual reward will also maximize the rewards for all agents. At every state, the agents try to bring the system closer to its goal. This can only happen if all agents choose actions that will maximize the joint reward. The policy that we have defined ensures a small probability of exploring new actions that might lead to bigger rewards.

4.4 Monitor Agent

The monitor agent $A_m = \langle \Sigma_m, \mathcal{X}_m, \mathcal{K}_m \rangle$ also belongs to the learning agent category. Its members are defined as follows.

A_m must have the ability to halt the control agent's task when it detects an anamoly in the system and wants to pass its status to other agents. We therefore assign $\chi_m \in \mathcal{X}_m$ to be $\chi_m = \{g_3\}$.

In addition to the above, A_m also needs to know what sensors and actuators the system has in order to detect and diagnose a fault. For the resource set $\kappa_m \in \mathcal{K}_m$, we let $\kappa_m = \langle \mathcal{D}, \mathcal{F} \rangle$.

In order to fulfil this role, the monitor agent needs strong ability to classify fault patterns from sensors without supervision, we propose therefore that the internal architecture $\sigma_m \in \Sigma_m$ of the monitor agent A_m is based on *adaptive resonance theory neural network* (ART-NN) algorithms. As was mentioned in Section 4.1, the monitor agent takes the role of the FDD component in our multiagent reconfigurable control scheme. A_m utilizes the ART network to autonomously classify faults.

We have presented in Chapter 3 the foundations for the ART network and the wavelet transform. We will now describe how they are utilized as the internal architecture of the monitor agent A_m to detect and diagnose faults in a system.

We define a FuzzyART network for each sensor that belongs to A_m. We also define one extra FuzzyART network which accommodates unmodelled or undefined errors. A Discrete Wavelet Transform (DWT) is defined for each FuzzyART network. Figure 4.6 shows the internal structure of a monitoring agent A_m for one sensor. The output of A_m is an error vector identifying the faults that has occurred.

We detect faults in the sensors by looking for discontinuities or abrupt changes in the data pattern. This is done by continuously storing the sensor data in a buffer and analyzing it every T_a intervals. The goal is to analyze the buffered data using the FuzzyART network. To achieve this, it was discussed in Section 3.4 that the data needs to be prepocessed. A set of features must be extracted from the data and presented as the input vector $\mathbf{I}$ to the FuzzyART network. The features or markers characterize the data and reduce the number of elements in $\mathbf{I}$. DWT can be used as a method of feature extraction.

The buffered data is decomposed using a DWT. The filters $h[n]$ and $g[n]$ used are determined by the characteristics of the application. This includes for example the type of transform and the type of signal that is being transformed.

In our work, we use the *Daubeahies* wavelet and scaling function as the $h[n]$ and $g[n]$ filters (Daubechies, 1992). They are *finite impulse response* (FIR) filters which means

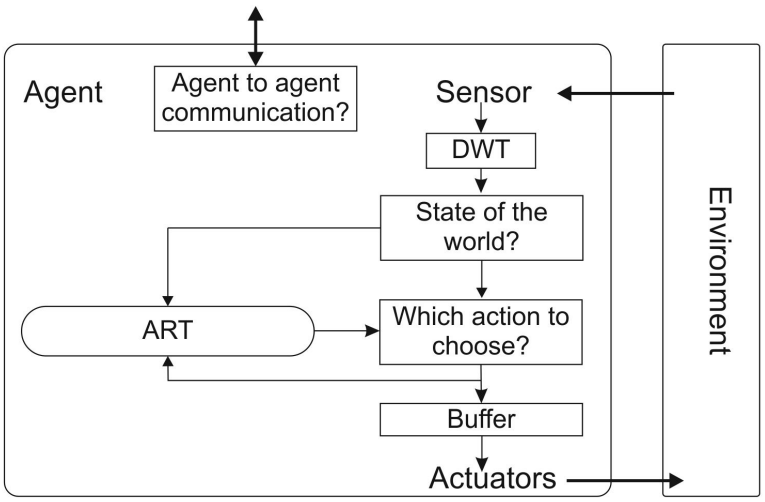


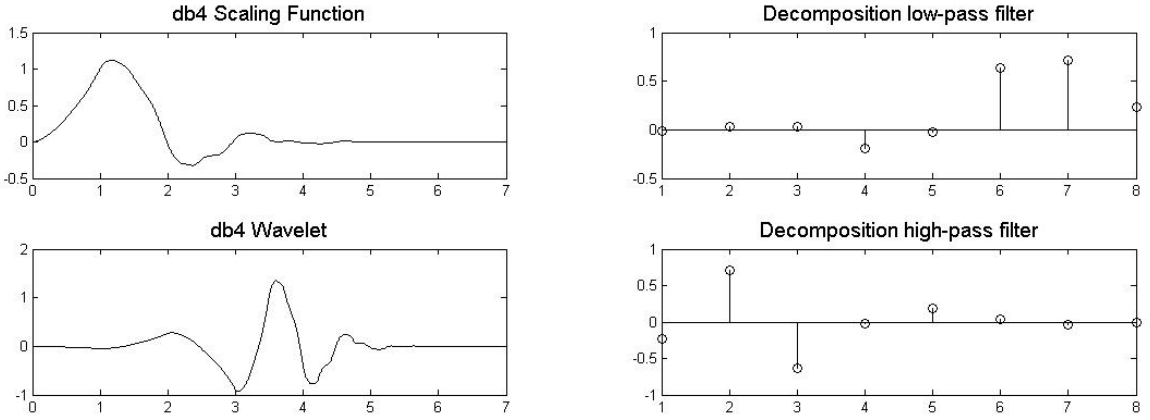
FIGURE 4.6: Internal architecture of the monitor agent A_m. The diagram shows the ART network for one sensor. The output of the networks are stored in an error vector which is the final output of the monitor agent.

that the duration of the impulse response settles to zero in finite time. They are also *orthogonal* which allows the reconstruction of the original signal. The Daubechies wavelet family has the capability of picking up details that some simpler wavelets such as the Haar wavelet might miss.

The Daubechies wavelets are characterized by the maximal number of *vanishing moments*. A vanishing moment limits the ability of the wavelet to represent polynomial information in a signal. The wavelet has a vanishing moment which is equal to half the number of coefficient N. For example, a Daubechies wavelet with $N = 2$ (written as D2) has one vanishing moment (written as Db1) and can comfortably represent a signal with one coefficient (a constant signal component). A D4 wavelet has two vanishing moments Db2 and can represent linear and constant components of a signal. In our case study we have chosen to use the Db4 wavelet heuristically. We found that this wavelet best suits the type of signal we would encounter in our system. Figure 4.7(a) shows the mother wavelet and scaling functions for Db4. Figure 4.7(b) shows the associated digital filter for the functions.

The first step in the feature extraction is to filter out the noise of the data. As was presented in Section 3.5, the low frequency component of a DWT contains the identity of a signal whereas the high frequency component contains the nuance and noise. If we discard the first few levels of the signal's DWT, the noise of the signal can be eliminated. This in shown in Figure 4.8 and Figure 4.9.

Figure 4.8 shows a four level DWT of a sine wave $x[n]$ with sample length $n = 1000$ corrupted by white noise. Notice that the high frequency components y_h^1 to y_h^4 have very small coefficient values. This indicates that they are associated with noise. If we eliminate these coefficients, we would get rid of the noise component of the signal.



(a) Db4 (b) Digital filter

FIGURE 4.7: Daubechies scaling and wavelet function with 4 vanishing moments (left). The associated discrete filter for the scaling and wavelet function (right).

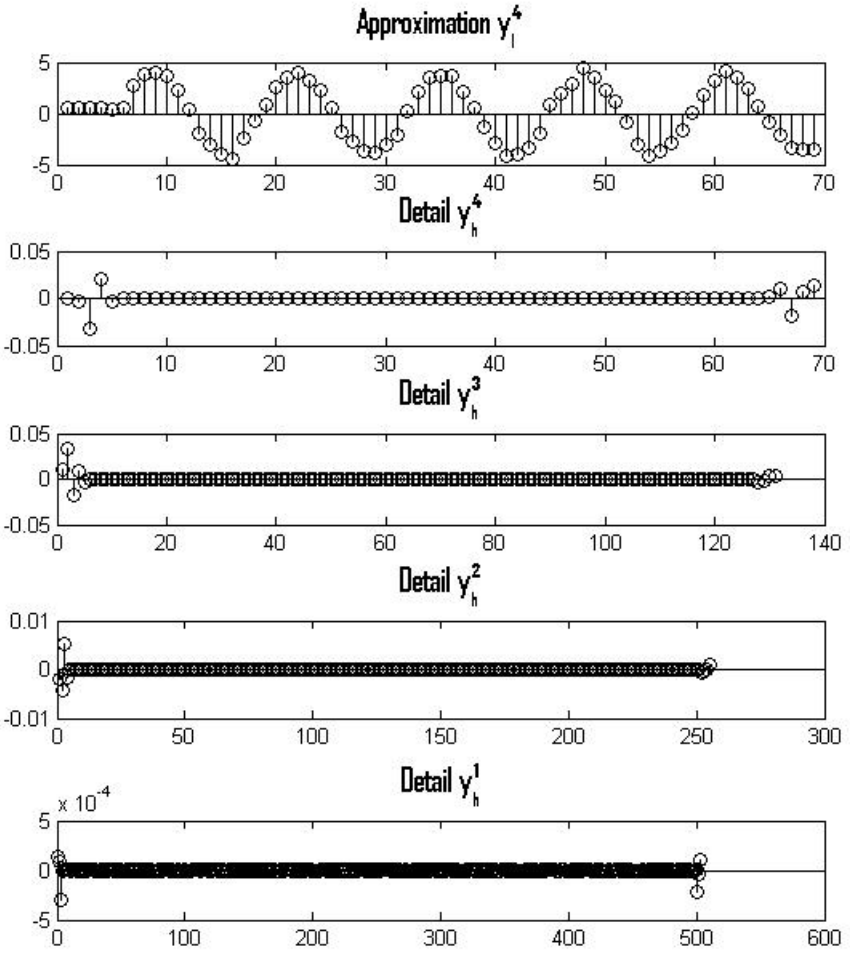


FIGURE 4.8: The coefficients of the DWT for the sine wave decomposition. The coefficients of the high frequency components are very small and can be discarded.

Figure 4.9 shows the reconstruction of the signal components in Figure 4.8. After several levels of filtering, the approximation component of the fourth level y_l^4 starts to resemble a sine wave without noise. Once the noise has been reduced, the coefficients of the approximation component y_l^i can be used as the basis for the features.

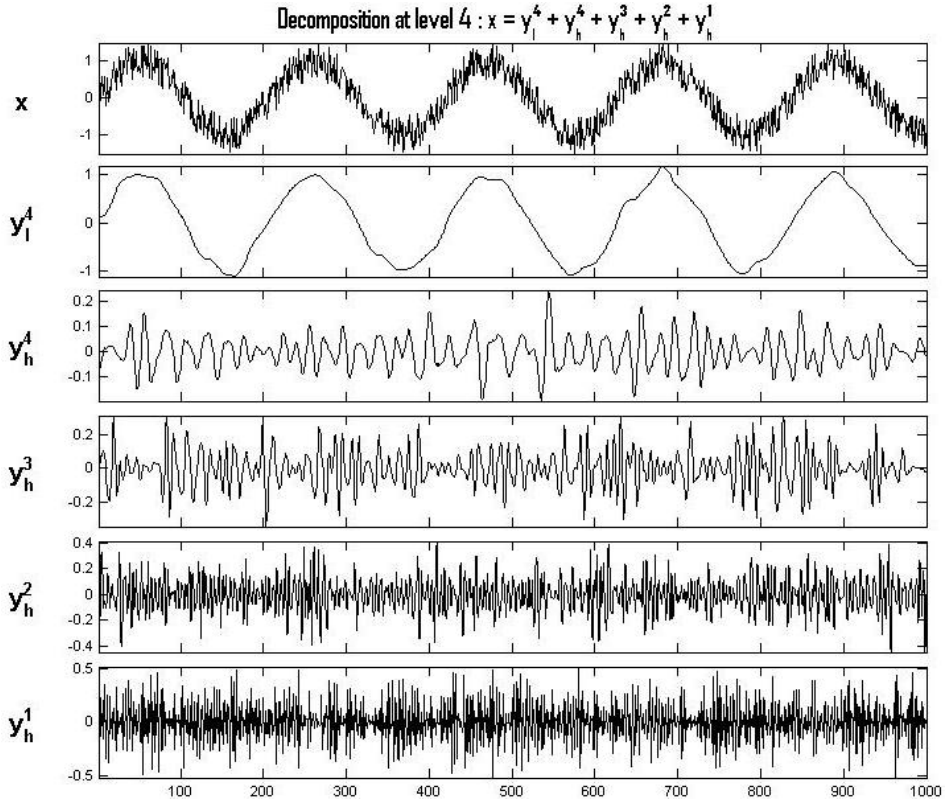


FIGURE 4.9: Decomposition of a sine wave $x[n]$ corrupted by white noise. Sample length $n = 1000$. Note that the graph shown above is not the graph of the DWT coefficients, rather a reconstruction of the signal components from the coefficients.

The next step in the feature extraction process is to reduce the number of features that would be presented to the FuzzyART network. This has to be done to accommodate the practical limitations of a real system. As was stated earlier, each sensor in our system is assigned a FuzzyART network. If there were a large number of sensors, this would slow the computation of the processor significantly because each FuzzyART network would have to take into account a large set of data.

In our case study we, figure 4.10 shows a data sample of a noisy signal from a wheel encoder and its fourth level decomposition. The data shows the velocity of a wheel in rotations per minute (rpm). The sample shows 400 seconds of data. An observation of the sensor data indicates that an abrupt change alters the data pattern of its decomposition. The component y_l^4 shows the underlying approximation of the signal. The bottom graph shows coefficients of the detail y_h^4. The abrupt change in data causes a spike in some of the coefficients. These spikes are the dominating components of the coefficients and fully characterizes the faults.

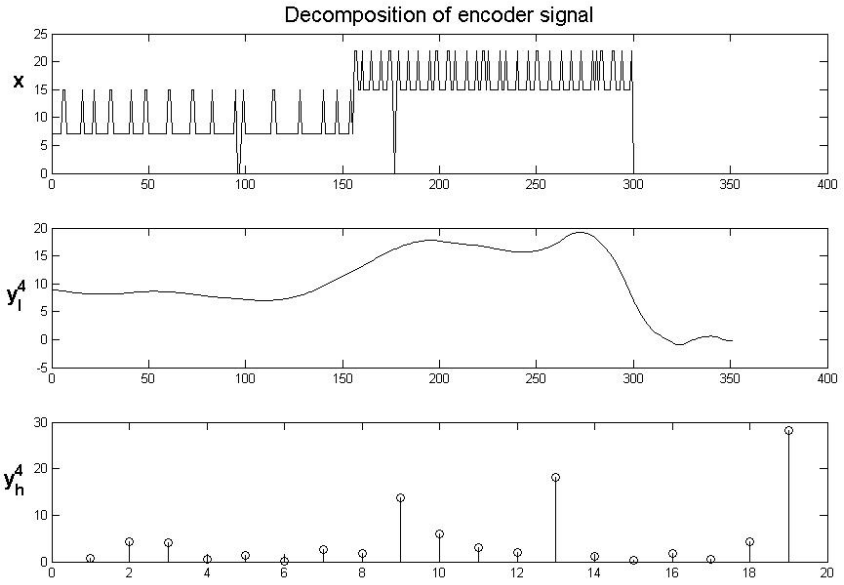


FIGURE 4.10: Decomposition of wheel encoder data. Abrupt change in data causes spikes in coefficients. The spikes are the dominant components of the coefficients.

Note that only 19 coefficients are left in y_h^4 due to the subsampling procedure. If we took the mean of all the coefficients, this will not drastically affect the characteristics of fault since the spikes dominate all other coefficient. The mean can then be used as the feature for the ART network. Based on this feature, the monitoring agent A_m detects which resources are faulty and passes them on to the other agents.

In order for the feature to be presented to the FuzzyArt network, the input I has to be normalized. This can be achieved by using a method called *complement coding* (Carpenter et al., 1991). If we had an incoming vector $\mathbf{a}$ in $\mathbb{R}^M$, the complement of a is represented by $\mathbf{a}^c$. This is also known as the on- and off-response of the incoming vector. For each a_i in the vector $\mathbf{a}$, we have

$$a_i^c \equiv 1 - a_i. \tag{4.7}$$

The input I is then defined as a vector in $\mathbb{R}^{2M}$,

$$I = (\mathbf{a}, \mathbf{a}^c) \equiv (a_1, \ldots, a_M, a_1^c, \ldots, a_M^c). \tag{4.8}$$

where

$$|I| = |(\mathbf{a}, \mathbf{a}^c)| \tag{4.9}$$

$$= \sum_{i=1}^{M} a_i + \left(M - \sum_{i=1}^{M} a_i \right) \tag{4.10}$$

$$= M. \tag{4.11}$$

4.5 Control Structure

The multiagent framework presented in this work can be utilized to control a dynamic plant. Figure 4.11 is a depiction of a classical fault tolerant control system implemented on a generic plant (Noura et al., 2009).

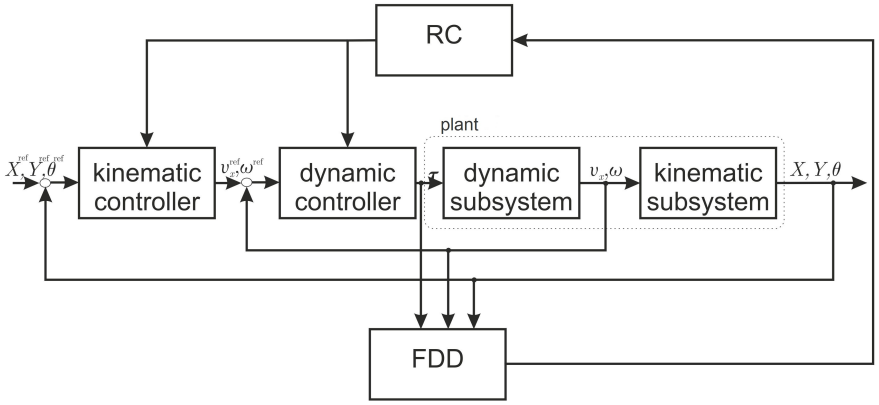


FIGURE 4.11: Cascaded control structure of a generic plant.

The fault detection and diagnosis (FDD) component analyzes the data from the subsystems and outputs the fault status of the system to the reconfiguration component (RC). The reconfiguration component modifies the system if the FDD component detects a fault.

Figure 4.12 shows our solution to the fault tolerant control system problem. As was stated in Section 4.1, the monitoring agent A_m takes the role of the FDD component in our scheme. The planner agent A_p and the control agents $A_c^k, k = 1, \ldots, 4$ take the role of the reconfiguration component. the action of A_m is passed to A_p. The planner decides which controller to use and passes it on to A_c. It also decides on the parameter settings of the system and changes it accordingly.

The action of A_m is passed to A_p. The planner decides which controller to use and passes it on to A_c. It also decides on the parameter settings of the system and changes it accordingly.

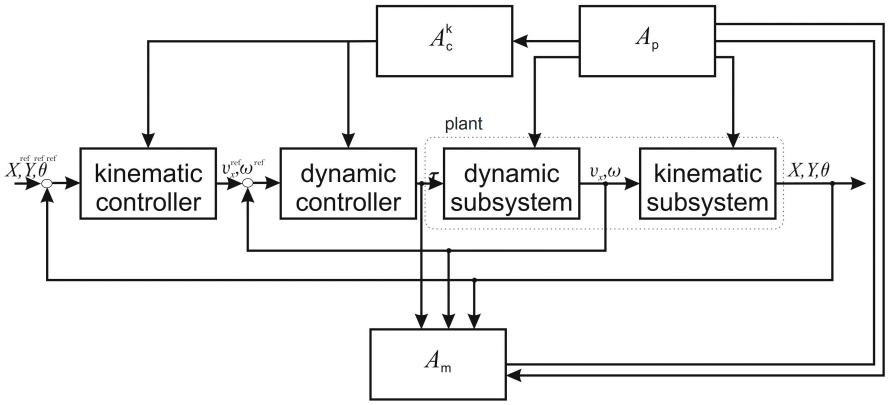


FIGURE 4.12: Multiagent control of a generic plant. Monitoring agent A_m takes the role of the FDD component. Similarly, A_p and A_c^k take the role of the reconfiguration component in a fault tolerant control system.

Figure 4.13 depicts the actions taken by all agents during default mode. In default mode, A_p sends the default controller to the controller agents A_c^k. The controller agents load the default controller and sets its mode to "not learning". It then executes the control task at hand. The monitor agent A_m assesses the state of the system, does not detect a fault and sends this information to A_p. Since there aren't any faults, A_p sticks with the default strategy.

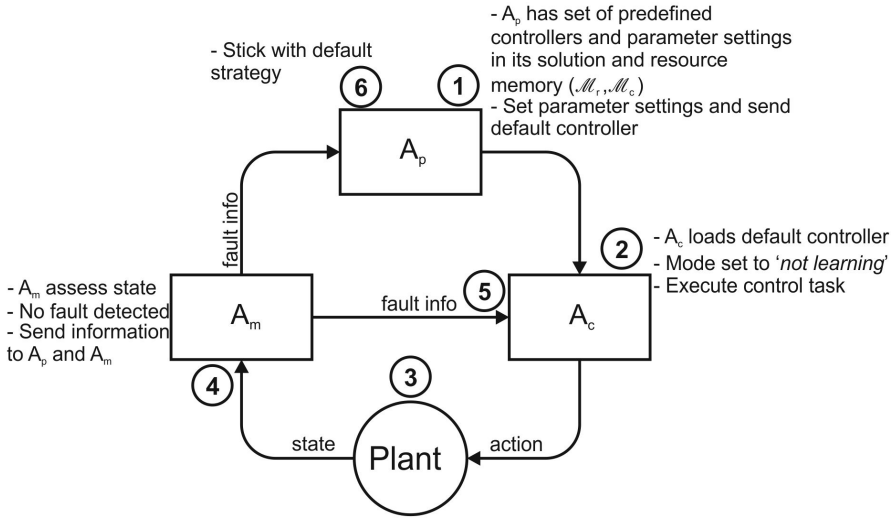


FIGURE 4.13: Actions taken by agents during default mode. A_p sends default controller to A_c. The control task is then executed. A_m checks for faults and relays the information to the other agents.

Figure 4.14 describes the system when a fault is detected. The monitor agent A_m detects a fault and tries to identify it. Once identified, it sends the fault information to A_p and A_c. The control agents save their parameters and stop the current control task. Upon receiving the fault information from A_m, the planner agent A_p searches its memory for a solution which best matches the identified fault.

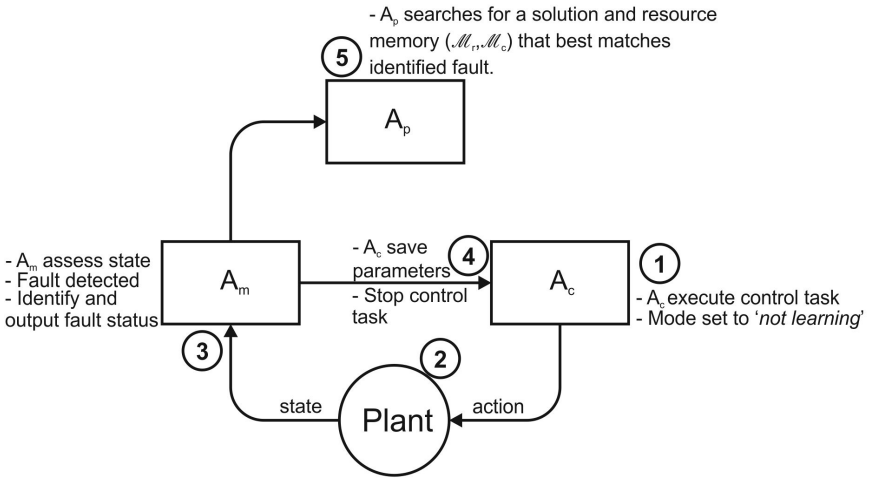


FIGURE 4.14: Actions taken by agents when a fault is detected. A_m identifies the fault and sends the information to A_p and A_c. A_c stops the control task and saves the current parameters. A_p searches its memory bank to find a solution for the fault.

Figure 4.15 and Figure 4.16 describe the processes involved during system reconfiguration. When the detected fault is a sensor fault (Figure 4.15), A_p reconfigures the system parameters by reallocating resources. This can be done by changing the parameters of the faulty sensors. Once the system is reconfigured, the planner agent tells A_c^k to complete the previous control task. The control agents reload saved parameters and execute the previous control task. The monitor agent A_m continues to monitor the system and sends this information to A_p and A_c^k. If there aren't any new faults, the planner agent sticks with the chosen strategy.

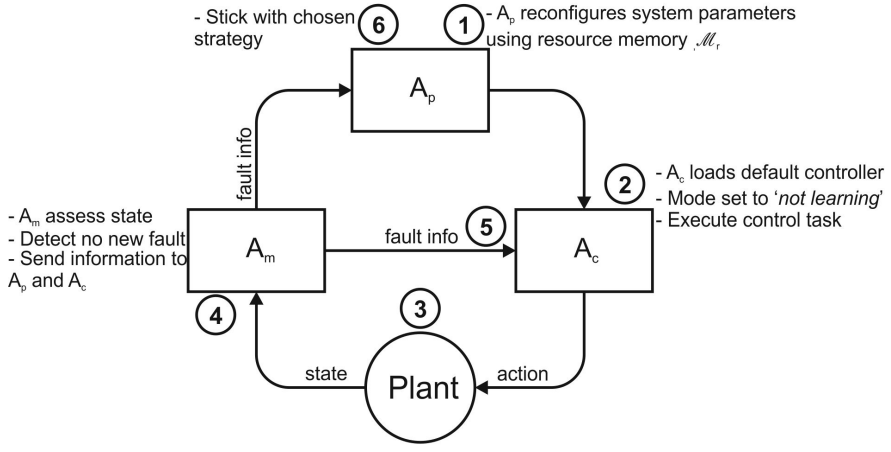


FIGURE 4.15: Reconfiguration of system after sensor fault. A_p changes parameters of the system. A_c reloads saved parameters and continues to complete previous control task. A_m continues to monitor system and relays the information to other agents.

In case of an actuator fault (Figure 4.16), the planner agent first disables the monitor agent A_m. This is done so that it does not interfere with the reconfiguration process.

The planner agent then searches its solution memory $\mathcal{M}_c$ for a suitable controller so that A_c^k can use it as a starting point to learn a new controller. If a suitable controller could not be found, A_p instructs A_c^k to learn a new controller from scratch. The controller agent sets its status to "learning". It then proceeds to learn a new controller. When a new controller has been learned, it sends the new controller to A_p for it to be stored. A_p then reenables the monitor agent to do its task.

If a new controller could not be learned due to say insufficient resources, this information is sent to A_p. The planner agent then shuts down the system and waits for an operator to diagnose the problem.

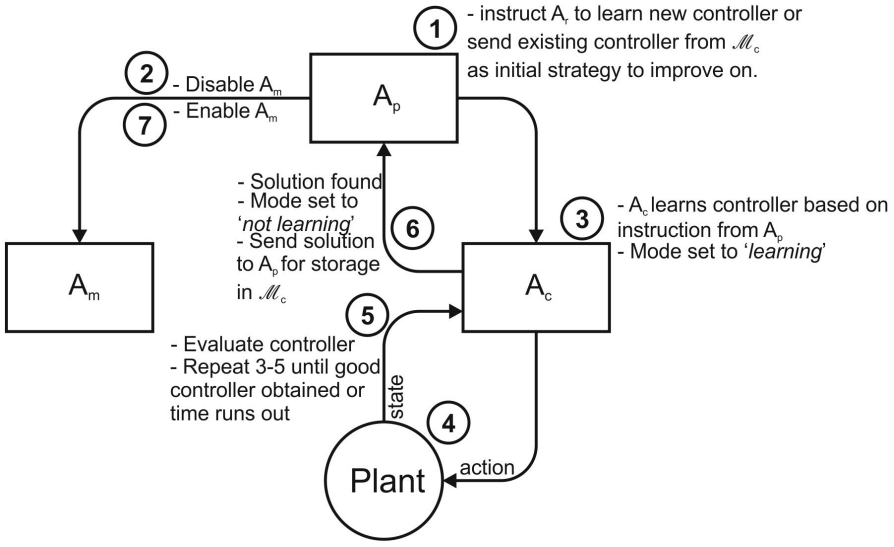


FIGURE 4.16: Reconfiguration of system after actuator fault. A_p tells A_c to learn new controller. A_c sets mode to "learning". A_m is disabled so that it does not interfere with learning. After a new controller is learned, it is stored in A_p. A_c sets mode to "not learning" and A_m is then reenabled.

4.6 Design Optimality

This section discusses the optimality of the design choices for the learning agents. The choice of reinforcement learning as the internal architecture for the control agents A_c^k lies in the fact that it is often necessary for the agents to learn new behaviour, such that their performance gradually improve (Sen and Weiss, 1999). The control agents in our work have to be able to learn new solutions to a control task when a fault occurs. It would be impossible to hard code the agents' behaviour to account for a large number of fault scenarios. Reinforcement learning provides a method for the agents to learn in a cooperative setting. By distributing responsibility across several control agents, the system becomes more robust because it is able tolerate failures by one or more agents (Stone and Veloso, 2000).

The following theorem by Singh et al. (2000) guarantees convergence of a multiagent direct policy search algorithm:

Theorem 4.4. *If players follow Infinitesimal Gradient Ascent (IGA), where* $\eta \to 0$*, then their strategies will converge to a Nash equilibrium or the average payoffs over time will converge in the limit to the expected payoffs of a Nash equilibrium.*

The theorem above covers any algorithm which employs a gradient ascent or descent technique which includes our algorithm. Singh et al. (2000) proved the above theorem for a two player matrix game. Bowling and Veloso (2002) and Banerjee and Peng (2003) presented empirically that the theorem could be extended to general stochastic games.

The monitor agent has to have the ability to detect known faults. It also has to be able to detect unmodelled faults. The detection of these faults have to be done in real time. This requires the monitor agent to learn fast and on the fly. The ART family of algorithm is the only unsupervised learning algorithm that can achieve this while maintaining stability of the learning process. The following theorem by Carpenter et al. (1991) guarantees stable category learning for FuzzyART used in conjunction with complement coding.

Theorem 4.5. *In response to an arbitrary sequence of analog or binary input vectors, a FuzzyART system with complement coding* (4.7) - (4.11) *and fast learning* (3.15) - (3.19) *forms stable categories as* $|\mathbf{w}_j|$ *monotonically decreases. In the conservative limit (i.e. the choice parameter* $\alpha \to 0$*), one-pass learning obtains such that no reset or additional learning occurs on subsequent presentations of any input.*

Other algorithms such as the k-means and the mixture of Gaussian (Bishop, 2006) are not able to learn in real time. The number of categories have to be fixed before the actual learning begins. This would hinder the monitor agent from detecting unmodelled faults. The Growing neural gas (GNG) algorithm (Fritzke, 1995) is able to learn in real time but lacks stability due to excessive plasticity.

The scheme presented in this work provides a unique approach to unsupervised learning of reconfigurable control of dynamic systems. While there have been other multiagent fault tolerant control schemes presented in the past (Zhu et al., 2004; Mendes et al., 2007), none have tackled the issue of integration between the FDD and CR component.

Recent attempts to integrate the FDD and CR components was discussed in the literature review chapter (Wang and Wang, 2011; Yetendje et al., 2012). The work from Wang and Wang (2011) only considered known faults. Yetendje et al. (2012) on the other hand took a robust control approach to integration. Our scheme is able to accommodate unmodelled faults and actively reconfigures the system by learning a new controller.

The interface between the different types agent in our scheme is optimal in the sense that the actions of one agent is directly used as a parameter in other agents. A multiagent system in our scheme is defined by

$$\Omega = \langle A_p, A_m, \{A_c\} \rangle, \tag{4.12}$$

where the individual agents are again listed for completeness:

$$\begin{aligned} A_p &= \langle \Sigma_p, \mathcal{X}_p, \mathcal{K}_p \rangle , \\ A_m &= \langle \Sigma_m, \mathcal{X}_m, \mathcal{K}_m \rangle , \\ A_c^k &= \left\langle \Sigma_c^k, \mathcal{X}_c^k, \mathcal{K}_c^k \right\rangle . \end{aligned}$$

The internal architecture $\sigma_p \in \Sigma_p$ is a conditional logic function $f : s_p \to a_p$ mapping its state to actions. The states of A_p is the fault state of the system. The fault state classification of the system is the action set of A_m. Therefore, the action of A_p can be written as

$$\begin{aligned} a_p &= f(s_p), \\ &= f(a_m). \end{aligned} \tag{4.13}$$

As was stated in Section 3.4, the action of A_m is defined by the output of the FuzzyART network

$$a_m = \frac{\sum_i \min(x_{1i}, \theta_{ij})}{\sum_i \theta_{ij}}.$$

The action of A_p is the configuration of the system, $a_p = \{b_j, c_i\}$ where $b_j \in \mathcal{B}$ is the parameter setting of the system stored in the resource memory $\mathcal{M}_r$ and $c_i \in \mathcal{C}$ is a controller stored in the solution $\mathcal{M}_s$ of A_p.

The internal architecture of the control agents σ_c^k is governed by the choice of controllers given to it by A_p. The solution c_i is the joint policy π containing policies of each control agent $\pi^k : \mathcal{S} \to A_c^k$. As was stated in Section 4.3, each control agent's policy is defined by

$$\pi_c^k(\mathbf{s}_t, a_t^k) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2\sigma^2} (a_t^k - \bar{a}_t^k(\mathbf{s}_t))^2 \right)$$

With regards to performance, the following theorem is stated.

Theorem 4.6. *Assuming that enough resources, i.e. redundancy in the system are available, and assuming that ample time is given to the system, all critical unmodelled faults will be detected by the monitor agent A_m and will correspondingly lead to a new control solution by the control agents A_c^k, $k = 1, \ldots, n$ in our scheme.*

In the theorem above, we assume that critical unmodelled faults are those which results in a significant deviation from the goal of the control task. The following gives an outline of the proof to the above theorem. The proof outline will be handled in two stages. The outline of the proof will first be handled with regards to the FDD component. Then, the proof outline with regards to the CR component will be discussed.

Proof. It is known from theorem 4.5 that the learning in A_m is stable. Therefore, any similar data patterns will not result in a false learning cycle of A_m. Since an ART network was defined to accommodate unmodelled faults which is highly correllated with the goal not being achieved, and since it was assumed that critical faults are those which results in a significant deviation from the goal of the control task, the monitor agent A_m will detect the unmodelled fault.

This complete the first part of the proof outline. The second part of the proof outline is done with regards to control reconfiguration.

We know from theorem 4.4 that our multiagent reinforcement learning algorithm will converge to a Nash equilibrium. It is also known that in fully cooperative games, the same reward is obtained by all agents. Therefore, since maximizing the reward of an agent equates to achieving the goal of the control task, a control solution will be found. □

4.7 Chapter Conclusion

The approach taken in this work was presented. A multiagent framework enables the two components (FDD and RC) of a reconfigurable control system to be integrated. In general, there should be at least one planner agent A_p, one monitor agent A_m and one control agent A_c. Each task (i.e. planning, monitoring and controlling) can be divided among several agents to reflect the complexity of a system. A total of six agents from the reactive and learning agents categories were defined in our scheme. We have one planner agent, one monitor agent and four control agents. Four control agents were used to accommodate for four independent actuators in the system.

A planner agent A_p supervises the overall control of a task. Being a reactive agent, A_p uses conditional logic to choose a control strategy.

A control agent A_c learns and executes a controller to complete a task. We have decided to use multiagent direct policy search algorithm as the internal architecture of A_c. We extend the algorithm to allow for continuous states and actions. A backpropagation neural network presented in Section 3.2 is used as a function approximator to generalize over continuous states and actions. We have defined four control agents A_c^k, $k = 1, \ldots, 4$. These agents need to take into account the actions of other agents in order to complete the control task.

All control agents share the same reward function r_t which means maximizing individual reward will also maximize the rewards for all agents. Our algorithm will converge to a Nash equilibrium since an agent's best response to other agents maximizing its rewards is also to maximize its own reward.

A monitor agent A_m detects and diagnoses a fault. An ART network enables A_m to classify faults autonomously without supervision. The wavelet transform depicted in Section 3.5 extracts features from an input pattern. The features are complement coded before being fed as input to the $F1$ nodes.

An ART network was defined for each sensor that belongs to A_m. One extra ART network was also defined which accommodates for unmodelled or undefined errors. A DWT is defined for each ART network. The output of A_m is an error vector identifying the faults that have occurred.

The structure of a multiagent reconfigurable control system was laid out. The role taken by each agent was defined. The state diagrams of different operating modes was presented to show how the agents communicated in executing a control task.

It was discussed why the design choices were optimal. Several advantages of reinforcement learning and the ART network in relation to multiagent systems were discussed. Several theorems were presented to support our argument.

Chapter 5

Formal Representation of Methodology in sEnglish

The theory of multi-agent reconfigurable control system was presented in Chapter 4. In this chapter we seek to formally represent our methodology. It is important to note that this chapter is a parallel contribution to this thesis and does not affect the outcome of the work. It is a tool to help bridge the field of control theory to the field of software programming. Using *Natural Language Programming* (NLP), important procedures can be conceptualized before formulating sentences and conceptual structures in a *machine ontology language* (MOL) (Veres, 2008). The sEnglish programming tool allows us to elegantly represent our work in NLP for interpretation and adoption by advanced intelligent agents of the future.

Sentences are written in such a way that they can be described further using other sentences. This is repeated until a sentence can only be meaningfully explained by conventional programming languages. The sentences can be separated into sections which help to modularize a task. This enables an engineer who wishes to adopt our methods to program their system without any difficulty. Apart from facilitating the use of our methods by humans, a document in NLP can be interpreted by advanced agents for use (Veres and Lincoln, 2008; Veres and Molnar, 2010; Veres, 2010). In the following, the sEnglish sections containing the sentences needed for realizing the FDD and CR components of a reconfigurable control system are presented. The low level meaning of some basic sentences is listed on the accompanying CD in the appendix.

The theory of multi-agent reconfigurable control system was presented in the previous section. In this section we seek to formally represent our methodology. Using *Natural Language Programming* (NLP), important procedures can be conceptualized before formulating sentences and conceptual structures in a *machine ontology language* (MOL) (Veres, 2008). The sEnglish programming tool allows us to elegantly represent our work in NLP for interpretation and adoption by advanced intelligent agents of the future.

Sentences are written in such a way that they can be described further using other sentences. This is repeated until a sentence can only be meaningfully explained by conventional programming languages. The sentences can be separated into sections which help to modularize a task. This enables an engineer who wishes to adopt our methods to program their system without any difficulty. Apart from facilitating the use of our methods by humans, a document in NLP can be interpreted by advanced agents for use (Veres and Lincoln, 2008; Veres and Molnar, 2010; Veres, 2010). In the following, the sEnglish sections containing the sentences needed for realizing the FDD and CR components of a reconfigurable control system are presented.

The machine knowledge representations of all the procedures of this methodology are contained in an sEnglish document with the following 'DocInfo.txt' file that is essentially the Contents:

```
Document title::  Multi agent integrated fault
                  tolerant control system
Author data:: Badril Abu Bakar
Section title 1:: Initializations
Section title 2:: Control and training procedures
Section title 3:: Reconfiguration and control
Section title 4:: Planner agent routine
Section title 5:: Monitoring agent routine
Section title 6:: Plant procedures
Section title 7:: Trivia } }
```

The interpretation of all sEnglish formulation depended on a single ontology that is defined in the following text:

sE- Data classes: Main classes are agent, centre of mass, character array (char), control input, coordinate, environment, episode (double), error, force (double), kalman filter, multiagent system, neural network, numeric array (double), physical object, proper name (char), reference values, rover model, sensor, time (double), two array structure, uncertainty parameter, var double (double). Attributes of physical object are name (char), class (char), mass (physical quantity), geometry (char), topology (char). Subclasses of agent are planner agent, monitor agent, reconfigure agent. Subclass of character array is text. Subclasses of control input are unnormalized control input, normalized control input. Subclasses of error are mean squared error, sum squared error. Subclasses of force are driving force, longitudinal friction, lateral friction, linear friction, angular friction, moment. Subclass of numeric array is matrix. Subclass of physical object is rover. Subclasses of sensor are dead reckoning sensor, gps sensor, ins sensor. Subclasses of time are time step, sampling time, time constant. Subclass of matrix is vector. Subclass of string is symbol. Subclass of text is string. Subclass of vector is scalar. Attributes of rover are length (double), width (double), maximum heading angle (double), model (rover model), actuator fault state (double), sensor fault state (struct). **-sE**

The above section is delimited by the **sE-** and **-sE** signs as the sEnglish plugin under Eclipse is able to directly extract/interpret this from a PDF version of this paper for an ontology of an sEnglish project [http://system-english.com, downloads].

5.1 Planner Agent Procedures

The planner agent chooses a strategy depending on the fault state of the system. Using the sentence:

```
Get Pagent choice of strategy passing M_action, Cagent,
Max_episode, Max_timestep as parameters.
```

the procedure to check the system for faults and reconfigure the system when necessary can be executed by A_p. The sEnglish sentence above has the following meaning:

```
Let Controller be the 'structure.solution memory' of Pagent.
Let Control_parameter be the 'structure.resource memory' of Pagent.
Let Number_stored_policy be a 'var double'.
Get Fault from M_action.
If no Unmodelled_fault, then do the following.
Display 'no unmodelled faults'.
Otherwise do the following.
Choose Controller for Cagent.
FCA.
Set Learningmode to '1'.
Set Max_episode to Max_episode.
Set Max_timestep to Max_timestep.
FCA.
Reconfigure Control_parameter according to M_action.
FCA.
```

Here FCA is an acronym for a sentence "Finish current action". In the above sEnglish sentences, variable `Controller` is assigned to the solution memory $\mathcal{M}_s$ of A_p and the variable `Control_parameter` is assigned to the resource memory $\mathcal{M}_r$. The planner agent then checks for faults. If an unmodelled fault is detected, an initial controller is chosen to be passed to the control agents and the system goes into learning mode. In the case where a known fault is detected, A_p reconfigures the system according to its conditional logic. When no faults are detected, the planner agent sticks with its default settings of controllers and commands the control agents to execute a control task.

The planner agent must also store newly learned controllers in its memory. The following sEnglish sentence can be executed by A_p to add controllers in its memory bank:

```
Add newly learned controller by Ragent in Pagent memory.
```

The above sEnglish sentence is interpreted as follows:

```
Let Memory be the 'solution memory' of Pagent.
Add Cagent to Memory.
Let the 'solution memory' of Pagent be Memory.
```

5.2 Control Agent Procedures

Depending on the learning mode, the control agents A_c^k have the responsibility to learn a controller or complete a control task. Using the sEnglish sentence:

```
Let Cagent choose actions based on Input, Exp_rate and Update_flag.
```

the procedure to choose their actions can be executed by A_c^k. The sEnglish sentence has the following meaning:

```
Let Previous_action be the 'action' of Cagent.
Let Cnet be the 'structure' of Cagent.
Update Cnet using Previous_action, Input, Update_flag and
Exp_rate as parameters.
Compute the Optimal_action from Cnet.
Get Current_action using gaussian function exploration
with Exp_rate as the standard deviation and Optimal_action as mean.
Set the 'action' of Cagent to Current_action.
Set the 'structure' of Cagent to Cnet.
```

The sEnglish sentences above implement the multiagent direct policy search reinforcement learning. Depending on the learning mode of the system, the control agents update their structure taking into account the previous action that was taken, the current input and the exploration rate. The control agents then computes the optimal action for the current time step. Again depending on the learning mode, the agents choose their actions according to a Gaussian exploration method or exploiting its current knowledge where the optimal action is calculated. Note that all sEnglish sentences quoted here unambiguously compile into MATLAB code that makes our agents operate, i.e. all descriptions are precise code and not merely pseudo code.

The control agents implement a neural network to approximate the continuous states and actions. The sentence:

```
Update Net using Pact, Input, Update_flag and Exp_rate as parameters.
```

updates the internal structure of the control agents. The control agents can use the sEngligh sentence by passing the network structure, previous action, network inputs, update flag and exploration rate as parameters. The above sEnglish sentence has the following meaning.

```
Load Input and bias in Input_vector.
Assign Target to Pact.
If Update_flag is active or random number is lower than Exp_rate,
then do the following.
Find Output from Net.
Calculate Delta from Target and Output.
Update Net weights using Update_flag and Delta as parameters.
Calculate mean squared error of Net from Delta.
FCA.
Compute inputs and outputs for all layers of Net using Input_vector.
```

In the meaning of the sEnglish sentence `Let Cagent choose actions based on Input, Exp_rate and Update_flag`, the sEnglish sentence `Update Net using Pact, Input, Update_flag and Exp_rate as parameters` is invoked as `Update Cnet using Previous_action, Input, Update_flag and Exp_rate as parameters`.

Updating the structure of the agents involves re-calculating the inputs, outputs and weights of a three layer neural network. The following sentence is used to execute the weight updates:

```
Update Net weights using Flag and Delta as parameters.
```

The sEnglish sentence above has the following meaning.

```
Let Weights be the 'weights' of Net.
Let Delta_weights be the 'delta weights' of Net.
Let Numlay be the 'number of layers' of Net.
Let M be the 'momentum' of Net.
Let N be the 'learning rate' of Net.
Let Activation be the 'activation vector' of Net.
Update Weights using Delta_weights, Numlay, M, N, Activation.
Let the 'weights' of Net be Weights.
Let the 'delta weights' of Net be Delta\_weights.
Let the 'number of layers' of Net be Numlay.
Let the 'activation vector' of Net be Activation.
```

The sEnglish sentences above update the network weights depending on the update flag and the output error `Delta`. To calculate the mean squared error of the network, the following sentence is used.

```
Calculate mean squared error of Net from Delta.
```

The meaning of the above sEnglish sentence is given below.

```
Let Sse be the 'sse' of Net.
Let Mse be the 'mse' of Net.
Let Nsamples be the 'number of samples' of Net.
Calculate Sse of Net.
Calculate Mse of Net.
Let the 'number of samples' of Net be Nsamples.
Let the 'sse' of Net be Sse.
Let the 'mse' of Net be Mse.
```

The network first calculates the sum of squared error `Sse`. It then calculates the mean squared error of the network `Mse`.

Finally, the following sEnglish sentence computes the activation function of all layers.

```
Compute inputs and outputs for all layers of Net using In.
```

The meaning of the above sEnglish sentence is given below.

```
Let Weights be the 'weights' of Net.
Let Delta_weights be the 'delta weights' of Net.
Let Numlay be the 'number of layers' of Net.
Let M be the 'momentum' of Net.
Let N be the 'learning rate' of Net.
Let Netvec be the 'net vector' of Net.
Let Activation be the 'activation vector' of Net.
Compute Activation.
Let the 'activation vector' of Net be Activation.
```

5.3 Monitor Agent Procedures

The monitor agent A_m takes the role of the FDD component in a reconfigurable control system. The following describes the procedure of detecting and diagnosing faults in a system.

The monitor agent periodically checks the system for faults. Initially, A_m preprocesses the data from its sensors. The following sentence can be executed by A_m to start the preprocessing stage.

```
Preprocess Sensor_data to get CcInput.
```

The above sEnglish sentence has the following meaning.

```
Compute noisy wavelet decomposition for Sensor_data.
Determine noise parameter Thr, Sorh, Keepapp for Sensor_data.
Eliminate noise to get Clean_sensor_data.
Compute clean wavelet decomposition Cc, Lc from Clean_sensor_data.
Compute clean coefficient Co from Cc, Lc.
Take the mean Mean_Co of the wavelet coefficients Co.
Arrange Input from Mean_Co.
Compute the compliment coding input CcInput from Input.
```

The monitor agent first computes the wavelet decomposition of the noisy data. In the next step, it eliminates the noise. After the noise has been eliminated, it computes a wavelet decomposition for the clean data. It then takes the mean of the coefficient which is compliment coded as a process of normalization. The compliment coded output is then ready to be used as the features for the input of the FuzzyART network. The following sentence can be executed by A_m to invoke the classification of inputs.

```
Classify CcInput to get Action of Magent.
```

The above sEnglish sentence has the following meaning.

```
Let Structure be the 'structure.artnet' of Magent.
Get fault Classification for CcInput from Structure.
Let the 'structure.artnet' of Magent be Structure.
Let Action be the 'action' of Magent.
Let Action be Classification.
Let the 'action' of Magent be Action.
Set Sensor_buffer_counter to '0'.
```

The sEngligh sentences above used by A_m above contains the actual FuzzyART algorithm. The FuzzyART algorithm is executed with the following sEnglish sentence by passing the pre-processed input and the structure of the monitor agent as parameters.

```
Get fault Classification for Input from Structure.
```

The sEnglish sentence above has is described by the following.

```
Let Net_numfeatures be the 'number of features' of Structure.
Let Net_vigilance be the 'vigilance parameter' of Structure.
Let Net_numepochs be the 'number of epochs' of Structure.
Let Net_bias be the 'bias' of Structure.
Let Net_weight be the 'weight' of Structure.
```

```
Let Net_numcat be the 'number of categories' of Structure.
Let Net_learningrate be the 'learning rate' of Structure.
Let Net_maxnumcat be the 'maximum number of categories' of Structure.
Let Resonance be a 'var double'.
Let Match be a 'var double'.
Let Weight_buffer be a 'var double'.
Extract Numfeature, Samples from In.
Let New_art_net be a 'art network'.
Let Numchanges be a 'var double'.
Let Current_sorted_index be a 'var double'.
Let Number_sorted_categories be a 'var double'.
Let Current_Category be a 'var double'.
Let Current_weightvector be a 'var double'.
Let First_category be a 'var double'.
Set First_category to '1'.
Let Temporary_index be a 'var double'.
Execute code "Categorization = ones(1, Samples);".
```

The procedure starts by first extracting the number of features and samples from the input. The network then computes the category activation from the input. The category activation is ranked from highest to lowest. This is done as follows.

```
Run cycle for "Epochnumber =1:Net_numepochs".
Set Numchanges to '0'.
Run cycle for "Sample_number = 1:Samples".
Extract Current_data from In, Sample_number.
Compute Category_activation passing Current_data, Net_weight,
Net_bias as parameters.
Rank Category_activation from highest to lowest to get Sorted_activation,
Sorted_categories.
```

If there are currently no categories, a new category is assigned to the network and the weights of the network are updated. This is done with the sentences below.

```
Set Resonance to '0'.
Set Match to '0'.
Determine the Number_sorted_categories from Sorted_categories.
Set Current_sorted_index to '1'.
Execute code "while(1)".
If Number_sorted_categories 'equals' '0',
then do the following.
Add new category passing Net_weight to get Weight_buffer.
Update Weight_buffer and detect Weight_change passing Current_data,
Weight_buffer, First_category,Net_learningrate as parameters.
Set Net_weight to Weight_buffer.
```

```
'Increase' Net_numcat by '1'.
'Increase' Numchanges by '1'.
Set Resonance to '1'.
Set Categorization equals '1'.
Terminate.
FCA.
```

Otherwise, the network calculates the match of the highest ranked category with the current data. If a match is bigger than a threshold, then "resonance" is achieved and the current category being matched becomes the output of the network. The weights of the network are updated accordingly.

```
Get Current_category and Current_weightvector.
Calculate Match given the Current_data, Current_weightvector.
If Match is bigger than Net_vigilance, then do the following.
Update Net_weight and detect Weight_change passing Current_data,
Net_weight, Current_category, Net_learningrate as parameters.
Set Categorization equals Current_category.
If Weight_change 'equals' to '1', then do the following.
'Increase' Numchanges by '1'.
FCA.
Set Resonance to '1'.
```

If the current category does meet the match criteria, the next highest ranked category is set as the current category to be matched. This is repeated until a match is found. If all the categories failed to match the input, a new category is assigned to the network and the weights of the network are updated. The sentences below describe this procedure.

```
Otherwise do as follows.
If Current_sorted_index 'equals' Number_sorted_categories,
then do the following.
If Current_sorted_index 'equals' Net_maxnumcat, then do the following.
Set Categorization equals '-1'.
Set Resonance to 1.
Otherwise do as follows.
Add new category passing Net_weight to get Weight_buffer.
Set Temporary_index to Current_sorted_index.
'Increase' Temporary_index by '1'.
Update Weight_buffer and detect Weight_change passing Current_data,
Weight_buffer, Temporary_index,
Net_learningrate as parameters.
Set Net_weight to Weight_buffer.
'Increase' Net_numcat by '1'.
'Increase' Numchanges by '1'.
Set Resonance to '1'.
```

```
Increase index of Categorization.
FCA.
Otherwise do as follows.
'Increase' Current_sorted_index by '1'.
FCA.
FCA.
If Resonance equals '1', then Terminate.
Finish cycle.
If Numchanges equals '0', then Terminate.
Finish cycle.
Let the 'weight' of Structure be Net_weight.
Let the 'number of categories' of Structure be Net_numcat.
```

Choosing agent actions is illustrated by the following executable paper section in sEnglish.

sE- Choosing agent actions : Let Cagent(r) choose actions based on Input(a) , Exp_rate(a) and Update_flag(a) . 'action' of Cagent. Let Rnet be the 'structure' of Cagent. Update Rnet using Previous_actions, Input, Update_flag and Exp_rate as parameters. Compute Let Previous_action be the the Optimal_action from Rnet. Get Current_action using gaussian function exploration with Exp_rate as the standard deviation and Optimal_action as mean. Set the 'action' of Cagent to Current_action. Set the 'structure' of Cagent to Rnet.

References: The rest of the sEnglish files can be found at
http://sysbrain.org/downloads/multiagent_reconfiguration_1.zip. **-sE**

In this sEnglish text the bold faced sentences are the activity title and the sample sentence that is followed by its meaning definition in normal font text. In the sample sentence brackets "(a)", "(r)" or "(ar)" are used to indicate that the object name is assumed to be available/known or resulting, respectively, when the sentence is used.

Here the web reference points to a web-location where the sEnglish definitions not printed in this paper can be found in a zip-file that contains also the text of the associated executable paper 'multi-agent_reconfiguration.exp'. When this pdf paper is read into a project under the *sEnglish Publisher* then the content of the zip file is automatically added to the project to form a complete representation of the research reported in this paper. There are also two sEnglish demo script available on *demo of rover control* and *training of rover*.

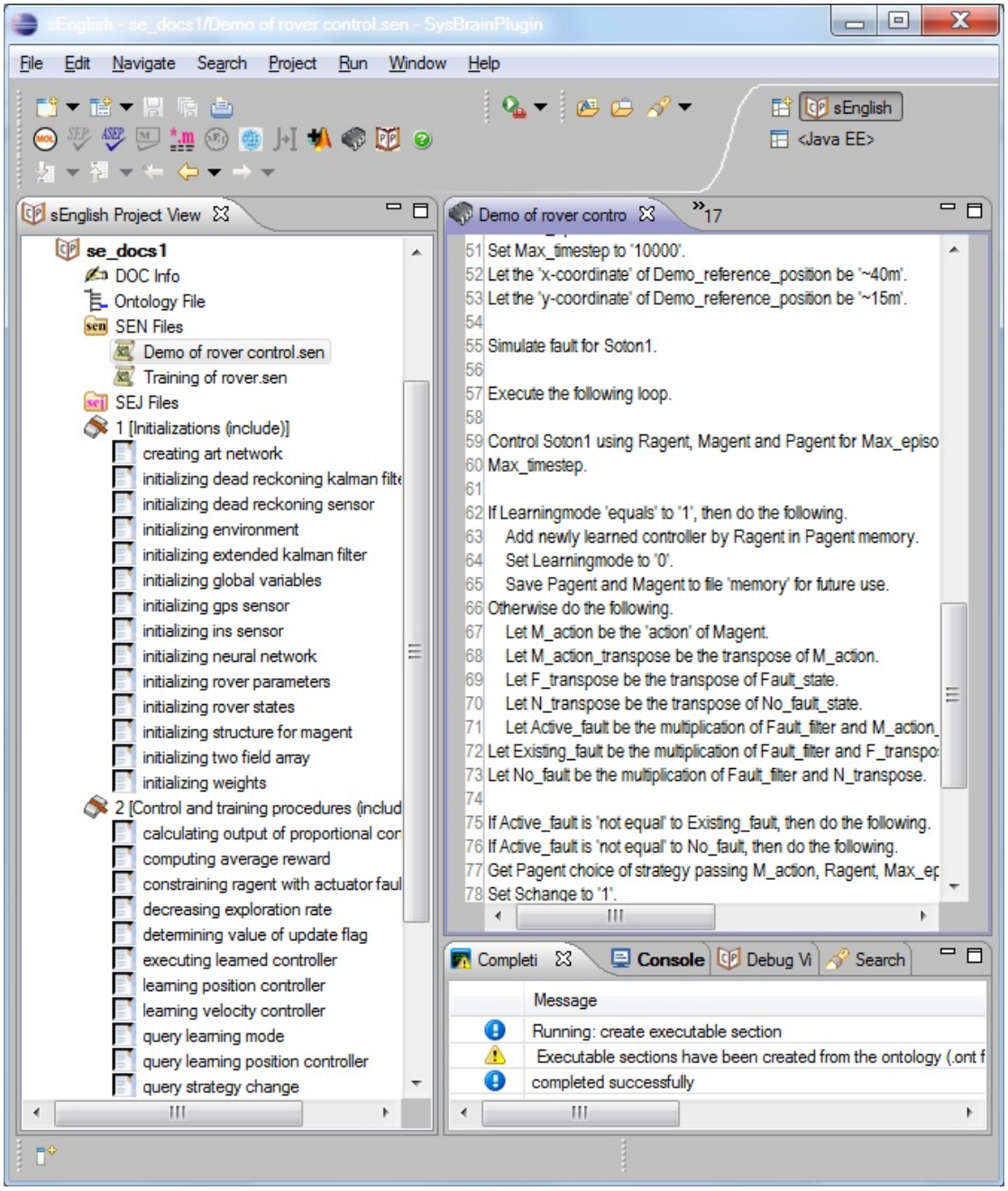


FIGURE 5.1: Illustration of the sEnglish plugin window under Eclipse.

5.4 Chapter Conclusion

We have formally represented our methodology using the natural language programming software sEnglish. This allows us to move away from the low level programming to a more user friendly abstraction. sEnglish sentences are written in such a way that it can be described further using other sEnglish sentences. This is repeated until a sentence can only be meaningfully explained by conventional programming languages.

Sentences were written to define procedures that are needed to realize the multiagent reconfigurable control system. Procedures for the planner, monitor and control agents were defined. These sentences could be used in a document to describe a multiagent reconfigurable control system. These documents would be read by the agents and interpreted as its task.

Chapter 6

Case Study

We have presented our solution to unsupervised learning of fault tolerant control in the previous chapters. This chapter gives an example of how the method can be implemented. Specifically, we will look at how to develop an active fault tolerant control system for a 4-wheel skid-steering vehicle which we simply call a rover for our purpose. A model for a rover is first derived. Three sensors that allow us to measure variables such as rover position and velocity will then be modelled. We will show how the sensors are fused which results in a complete simulation system where we can demonstrate our method of dealing with faulty sensors and actuators. In Section 6.3, we present how the control structure of the rover can be fitted in the multiagent framework. Subsequently, several cases of the rover fault states are analyzed, and discussed and finally conclusions are drawn.

6.1 Rover Model

The rover is equipped with four independent motors that drive the wheels and can be controlled separately. The steering mechanism is achieved by producing a differential velocity between its wheels. This type of vehicle is usually known as a skid-steering vehicle (Caracciolo et al., 1999). In order to turn, it has to skid laterally. A free body diagram of the rover is shown in Figure 6.1 where a, b and c are parameters defining the physical dimensions of the rover.

It assumed that the rover moves on a planar surface with the inertial orthonormal basis X_g, Y_g, Z_g making up a frame following the East-North-Up (ENU) coordinate system with X_g being East, Y_g being North and Z_g being Up. A local coordinate frame at the rover's center of mass (COM) is defined as x_l, y_l, z_l. We assume in our work that the rover moves on a two dimensional plane resulting in Z_g and z_l being constants. The linear and angular velocity of the rover can be expressed in the local frame as $\mathbf{v} = \begin{bmatrix} v_x & v_y & 0 \end{bmatrix}^T$

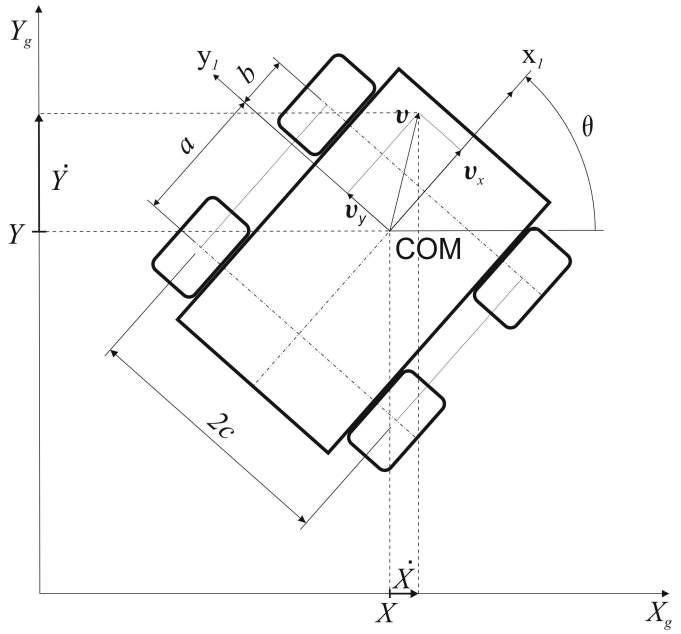


FIGURE 6.1: A free body diagram of a rover

and $\omega = \begin{bmatrix} 0 & 0 & \omega \end{bmatrix}^T$ respectively. The heading angle of the rover with respect to X_g is denoted as θ.

We define the nonlinear model of the rover as follows (Kozlowski, 2004):

$$\begin{aligned}
\dot{X} &= v_x \cos\theta + \omega x_{\text{ICR}} \cos\theta \\
\dot{Y} &= v_x \sin\theta - \omega x_{\text{ICR}} \cos\theta \\
\dot{\theta} &= \omega \\
\dot{v}_x &= \frac{v_x \omega}{m} - \frac{v_x c}{rm}(\tau_R - \tau_L) + \frac{x_{\text{ICR}} F_{ry} + M_r}{m} v_x \\
\dot{\omega} &= \frac{-v_x^2}{m x_{\text{ICR}} + I} + \frac{\theta}{r(m x_{\text{ICR}}^2 + I)}(\tau_R + \tau_L) + \frac{F_{rx}}{m x_{\text{ICR}}^2 + I}\theta.
\end{aligned} \tag{6.1}$$

In Eq. 6.1, $\dot{X}$ and $\dot{Y}$ are the velocity components of the rover with respect to the inertial frame; $\dot{\theta}$ is the angular velocity and is equal to ω; m is the mass of the rover; r is the *effective rolling* radius of a wheel; x_{ICR} is the fixed coordinate of the *instantneous centre of rotation* (ICR); τ_R and τ_L are the input torque applied to the right and left wheels respectively; F_{rx} and F_{ry} are linear resistive forces; I is the rover moment of inertia; and M_r is the resistive moment of the rover.

The linear and angular velocity of the rover body v_x and ω are obtained by deriving a relationship between the body and wheel velocities. Figure 6.2 shows the linear and angular velocity for one wheel.

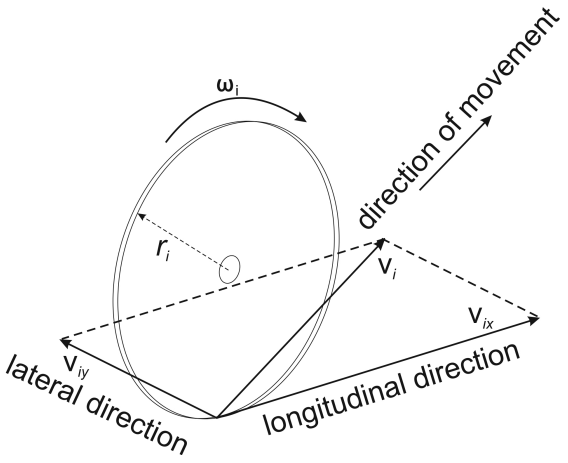


FIGURE 6.2: Velocity components for one wheel.

The wheel rotates with angular velocity ω_i where $i = 1, \ldots, 4$. The lateral velocity is represented by v_{iy} and the longitudinal velocity is represented by. v_{ix}. The total velocity for wheel i is written as $\mathbf{v}_i$. The longitudinal velocity can be obtained by

$$v_{ix} = r_i \omega_i. \tag{6.2}$$

In Eq. 6.2, r_i is the effective rolling radius of the wheel. In order to build a relationship with the body velocity v_x and w, all wheels have to be considered together. Figure 6.3 depicts the rover with the individual wheel velocities.

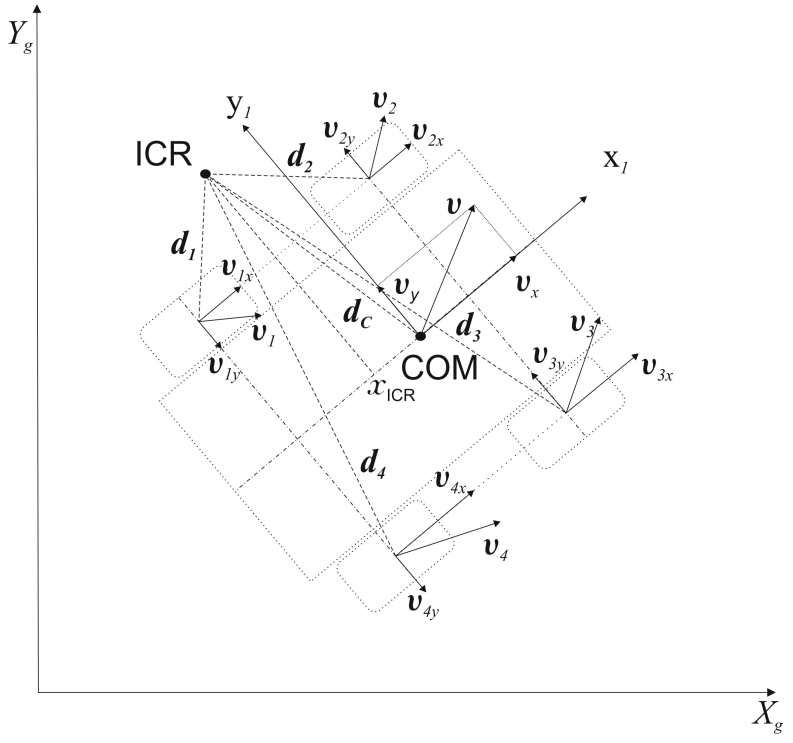


FIGURE 6.3: Components of the body and wheel velocities

From the figure, the *instantaneous center of rotation* (ICR) is the virtual center at which the body rotates around. The radius vectors $\mathbf{d}_i = \begin{bmatrix} d_{ix} & d_{iy} \end{bmatrix}^T$ and $\mathbf{d}_C = \begin{bmatrix} d_{Cx} & d_{Cy} \end{bmatrix}^T$ are defined with respect to ICR. The following expression for the body angular velocity ω can be obtained:

$$\frac{\|\mathbf{v}_i\|}{\|\mathbf{d}_i\|} = \frac{\|\mathbf{v}\|}{\|\mathbf{d}_C\|} = |\omega|. \tag{6.3}$$

The symbol $\|\cdot\|$ represents the Euclidean norm. It could also be written component wise as

$$\frac{v_{ix}}{d_{iy}} = \frac{v_x}{d_{Cy}} = \frac{v_{iy}}{d_{ix}} = \frac{v_y}{d_{Cx}} = \omega. \tag{6.4}$$

If we define the coordinates of ICR in the local frame as

$$ICR = \begin{bmatrix} x_{ICR} & y_{ICR} \end{bmatrix} = \begin{bmatrix} -d_{Cx} & d_{Cy} \end{bmatrix}, \tag{6.5}$$

then we could write the angular velocity as

$$\omega = \frac{v_x}{y_{ICR}} = -\frac{v_y}{x_{ICR}}. \tag{6.6}$$

Looking at Figure 6.3, the following are the relationships of the radius vectors:

$$\begin{aligned} d_{1y} &= d_{2y} = d_{Cy} - c, \\ d_{3y} &= d_{4y} = d_{Cy} + c, \\ d_{1x} &= d_{4x} = d_{Cx} - a, \\ d_{2x} &= d_{3x} = d_{Cx} + b. \end{aligned} \tag{6.7}$$

In the equation above, a, b and c are the parameters taken from Figure 6.1. Using Eq 6.7 in Eq. 6.4, the relationships between the wheel velocities are obtained:

$$\begin{aligned} v_L &= v_{1x} = v_{2x}, \\ v_R &= v_{3x} = v_{4x}, \\ v_F &= v_{2y} = v_{3y}, \\ v_B &= v_{1y} = v_{4y}. \end{aligned} \tag{6.8}$$

The velocities v_L and v_R represent the longitudinal velocity of the left and right wheels respectively. The velocities v_F and v_B represent the lateral velocities of the front and back wheels.

Using Eq. 6.4 - Eq. 6.8, the relationship between the body and the wheel velocities can be states as

$$\begin{bmatrix} v_L \\ v_R \\ v_F \\ v_B \end{bmatrix} = \begin{bmatrix} 1 & -c \\ 1 & c \\ 0 & -x_{ICR} + b \\ 0 & -x_{ICR} - a \end{bmatrix} \begin{bmatrix} v_x \\ \omega \end{bmatrix}. \tag{6.9}$$

Assuming we take $r_i = r$ for each wheel, the angular velocity of the right and left wheels can be written as

$$\omega_w = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} = \frac{1}{r} \begin{bmatrix} v_L \\ v_R \end{bmatrix}. \tag{6.10}$$

Finally, combining Eq.6.9 and Eq.6.10 we can write the rover body velocities in terms of wheel velocities as follows:

$$\begin{bmatrix} v_x \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{v_L + v_R}{2} \\ \frac{-v_L + v_R}{2c} \end{bmatrix}. \tag{6.11}$$

Let us define the states of the system as

$$\begin{aligned} x_1 &= X \\ x_2 &= Y \\ x_3 &= \theta \\ x_4 &= v_x \\ x_5 &= \dot{x}_3 = \omega. \end{aligned}$$

We can then organize the nonlinear model of the rover in the form of a state space equation:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & \cos x_3 & x_{\mathrm{ICR}}\sin x_3\\ 0 & 0 & 0 & \sin x_3 & x_{\mathrm{ICR}}\cos x_3\\ 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & \frac{x_4}{m}\\ 0 & 0 & 0 & \frac{-x_4}{mx_{\mathrm{ICR}}^2+I} & \frac{x_{\mathrm{ICR}}}{mx_{\mathrm{ICR}}^2+I}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ \frac{-x_4 c}{rm} & \frac{x_4 c}{rm}\\ \frac{x_3}{r(mx_{\mathrm{ICR}}^2+I)} & \frac{x_3}{r(mx_{\mathrm{ICR}}^2+I)}\end{bmatrix}\begin{bmatrix}\tau_L\\ \tau_R\end{bmatrix} + \begin{bmatrix}0\\ 0\\ 0\\ \frac{x_4(x_{\mathrm{ICR}}F_{ry}+M_r)}{m}\\ \frac{x_3 F_{rx}}{mx_{\mathrm{ICR}}^2+I}\end{bmatrix}. \tag{6.12}$$

The work presented by Kozlowski (2004) used a cascading control structure. A kinematic level controller based on Dixon et al. (2001) was proposed and an extension of the control law to the dynamic and motor level was done using a Lyapunov analysis and backstepping technique. In Caracciolo et al. (1999), the dynamic feedback linearization paradigm was exploited to design a controller for the above system. In our work, we will follow the path similar to Kozlowski (2004) in that we will make use of the cascading control structure. However, only the dynamic and kinematic level will be considered. Figure 6.4 illustrates how the rover model is decomposed into a dynamic and kinematic subsystem.

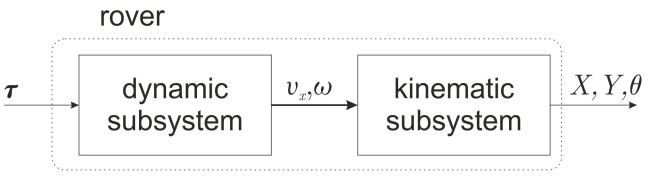


FIGURE 6.4: Decomposition of rover model

6.2 Sensor Modelling

In our work, the sensors required to simulate a control task were modelled. In all, three modules make up the sensor system of the rover. It consists of a global positioning system (GPS) module, an inertial navigation system (INS) module and local encoders on the wheels. An extended Kalman filter was designed to deal with the nonlinear nature of the system and to fuse the sensor readings to get an improved state estimate. There are different ways in which the sensor data can be fused. We have opted for the *loosely coupled* integration of the sensors as detailed in Grewal et al. (2007). In loosely coupled sensor integration, only the output of the sensor modules are used as inputs to the integrating filter. Each sensor module may have an internal Kalman filter themselves,

but it is not included in the integrating filter and will not be dealt with in our work. The following describes the measurement model of the sensors and how they are fused in an extended Kalman filter.

6.2.1 GPS receiver

The GPS model for integration is formulated mathematically as

$$\mathbf{z}_{\text{gps}} = \mathbf{h}(\mathbf{x}_{\text{rov}}) + \delta_{\text{gps}} \tag{6.13}$$

$$\frac{d}{dt}\delta_{\text{gps}} \approx \mathbf{f}_{\text{gps}}(\delta_{\text{gps}}, \mathbf{x}_{\text{rov}}) + \mathbf{w}_{\text{gps}}(t). \tag{6.14}$$

In the above equation, $\mathbf{z}_{\text{gps}}$ is the GPS output; $\mathbf{h}(\mathbf{x}_{\text{rov}})$ is the measurement function; δ_{gps} represents the GPS output error; $\mathbf{f}_{\text{gps}}(\delta_{\text{gps}}, \mathbf{x}_{\text{rov}})$ and the white noise $\mathbf{w}_{\text{gps}}(t)$ represent the dynamic model for δ_{gps}.

An exponentially damped position error model was used for the GPS receiver (Grewal et al., 2007). It has the form

$$\frac{d}{dt}\begin{bmatrix}\delta p_{\text{gpsN}}\\ \delta p_{\text{gpsE}}\\ \delta p_{\text{gpsD}}\end{bmatrix} = \begin{bmatrix}-1/\tau_{\text{hor}} & 0 & 0\\ 0 & -1/\tau_{\text{hor}} & 0\\ 0 & 0 & -1/\tau_{\text{ver}}\end{bmatrix}\begin{bmatrix}\delta p_{\text{gpsN}}\\ \delta p_{\text{gpsE}}\\ \delta p_{\text{gpsD}}\end{bmatrix} + \begin{bmatrix}w_{\text{hor}}(t)\\ w_{\text{hor}}(t)\\ w_{\text{ver}}(t)\end{bmatrix}. \tag{6.15}$$

In the equation, $\delta p_{\text{gpsN}}, \delta p_{\text{gpsE}}, \delta p_{\text{gpsD}}$ represent the GPS North, East and Down component position error; τ_{hor} and τ_{ver} denote the position error correlation times in the horizontal and vertical plane respectively.

6.2.2 INS model

The INS model for integration is formulated mathematically as

$$\mathbf{z}_{\text{ins}} = \mathbf{h}(\mathbf{x}_{\text{rov}}) + \delta_{\text{ins}} \tag{6.16}$$

$$\frac{d}{dt}\delta_{\text{ins}} \approx \mathbf{f}_{\text{ins}}(\delta_{\text{ins}}, \mathbf{x}_{\text{rov}}) + \mathbf{w}_{\text{ins}}(t). \tag{6.17}$$

In the above equation, $\mathbf{z}_{\text{ins}}$ is the INS output; $\mathbf{h}(\mathbf{x}_{\text{rov}})$ is the measurement function; δ_{ins} represents the INS output error; $\mathbf{f}_{\text{ins}}(\delta_{\text{ins}}, \mathbf{x}_{\text{rov}})$ and the white noise $\mathbf{w}_{\text{ins}}(t)$ represent the dynamic model for δ_{ins}.

We have used the *random-walk tilt model* as the INS error model (Grewal et al., 2007). It has the form

$$\begin{bmatrix} \delta p_{\text{insN}} \\ \delta p_{\text{insE}} \\ \delta v_{\text{insN}} \\ \delta v_{\text{insE}} \\ \rho_{\text{insN}} \\ \rho_{\text{insE}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -\Omega_{\text{sch}}^2 & 0 & 0 & -2s\Omega_{\odot} & -g & 0 \\ 0 & -\Omega_{\text{sch}}^2 & -2s\Omega_{\odot} & 0 & 0 & g \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta p_{\text{insN}} \\ \delta p_{\text{insE}} \\ \delta v_{\text{insN}} \\ \delta v_{\text{insE}} \\ \rho_{\text{insN}} \\ \rho_{\text{insE}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ w_{\rho}(t) \\ w_{\rho}(t) \end{bmatrix}. \tag{6.18}$$

In the above equation, $\delta p_{\text{insN}}, \delta p_{\text{insE}}$ are the INS North and East component position error; $\delta v_{\text{insN}}, \delta v_{\text{insE}}$ are the INS North and East velocity error; $\rho_{\text{insN}}, \rho_{\text{insE}}$ are the INS North and East axis tilt error; Ω_{sch} is the Shuler frequency and is ≈ 0.00124 rad/s; s is the sine of the latitude; $\Omega_{\odot}$ is the earth rotation rate and is $\approx 7.3 \times 10^{-5}$; and g is the gravitational acceleration which is $\approx 9.8\text{ms}^{-2}$.

6.2.3 Encoder model

We assume that the encoder measures a scalar random constant, velocity in this case. We also assume that the measurement is corrupted by white noise. We can therefore model the process as

$$\begin{aligned} x(t) &= Ax(t-1) + Bu(t) + \psi(t) \\ &= x(t-1) + \psi(t). \end{aligned} \tag{6.19}$$

In the above equation, the state does not change from step to step so $A = 1$. There is no control input so $u = 0$. The process noise is denoted by $\psi(t)$. The measurement model can be written as

$$\begin{aligned} z(t) &= Hx(t) + w(t) \\ &= x(t) + w(t). \end{aligned} \tag{6.20}$$

The noisy measurement is of the state directly so $H = 1$. The measurement noise is represented by $w(t)$. Each wheel has a local encoder. From Eq.6.8, $v_L = v_{1x} = v_{2x}$ and $v_R = v_{3x} = v_{4x}$. If there were a discrepancy between the encoder readings for v_{1x} and v_{2x} due to noise for example, how do we know which one represents the correct value of v_L?

We solve this problem by implementing a Kalman filter in fusing each encoder's reading to estimate the states v_L and v_R. We have assumed at the beginning of this section that the encoders are trying to track a scalar random constant. From Eq.6.19, $A = 1$. We can write the discrete form of this equation as

$$x_k = \Phi x_{k-1} + \psi_k. \tag{6.21}$$

Here, $\Phi = e^{AT_s} \approx 1$. We assume that the noise term ψ_k has zero mean with standard deviation σ_Q. Therefore, the process noise covariance matrix is defined by

$$Q = E\psi_k\psi_k = \sigma_Q^2. \tag{6.22}$$

The discrete measurement model is described by

$$z_k = Hx_k + w_k. \tag{6.23}$$

Since there are two observations of v_L or v_R, the measurement sensitivity matrix is written as

$$H = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{6.24}$$

We assume that the noise term w_k has zero mean and standard deviation σ_R. Therefore, the measurement uncertainty covariance matrix is defined by

$$R = Ew_k w_k^T = \begin{bmatrix} \sigma_{R1}^2 & 0 \\ 0 & \sigma_{R2}^2 \end{bmatrix}. \tag{6.25}$$

Using all the information, a Kalman filter for wheel velocity estimation can be implemented as follows.

1. Compute a priori value of state estimate $\hat{x}(-) = \Phi_{k-1}\hat{x}_{k-1}(+)$.
2. Compute a priori value of the state estimate uncertainty covariance matrix $P_k(-) = \Phi_{k-1}P_{k-1}(+)\Phi_{k-1}^T + Q_{k-1}$.

3. Compute the Kalman gain $K_k = P_k(-)H_k^T[H_kP_k(-)H_k^T + R_k]^{-1}$.
4. Compute posteriori value of $P_k(+) = [I - K_kH_k]P_k(-)$.
5. Compute posteriori value of state estimate $\hat{x}_k(+) = \hat{x}(-) + K_k[z_k - H_k\hat{x}(-)]$.

6.2.4 Sensor fusion

For this example of reconfigurable control a 16 state discrete extended Kalman filter (EKF) was implemented to fuse all the different sensors. Table 6.1 shows the equations involved in an EKF (Grewal and Angus, 2001). The states of the integrated system are listed in Table 6.2.

The states of the rover model are augmented by the states of the error model of the GPS and INS modules. In the following we discuss how we obtain the variables listed in Table 6.1.

The integrated state transition matrix of the new system is a 16×16 matrix defined by

$$\Phi_{\text{int}} = \left[\begin{array}{c|c|c} \Phi_{\text{rov}} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \Phi_{\text{gps}} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \Phi_{\text{ins}} \end{array}\right]. \tag{6.26}$$

In the right hand side of the block matrix equation above, $\Phi = e^{T_sF}$ where F is the dynamic coefficient matrix and T_s is the sampling time. The state transition matrix Φ_{gps} and Φ_{ins} are defined as follows (Grewal et al., 2007):

$$\Phi_{\text{gps}} = \exp\left(T_s \cdot \begin{bmatrix} -1/\tau_{\text{hor}} & 0 & 0 \\ 0 & -1/\tau_{\text{hor}} & 0 \\ 0 & 0 & -1/\tau_{\text{ver}} \end{bmatrix}\right) \tag{6.27}$$

$$\Phi_{\text{ins}} = \exp\left(T_s \cdot \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -\Omega_{\text{sch}}^2 & 0 & 0 & -2s\Omega_{\odot} & -g & 0 \\ 0 & -\Omega_{\text{sch}}^2 & -2s\Omega_{\odot} & 0 & 0 & g \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\right) \tag{6.28}$$

In order to get the state transition matrix of the rover Φ_{rov}, the nonlinear state space equation of the rover has to be discretized and linearized about the estimated trajectory. Discretizing using Euler's method, we get an equation of the form

Nonlinear dynamic model:

$\mathbf{x}_k = f_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_k) + w_k, \quad w_k \sim \mathcal{N}(0, Q_k)$

Nonlinear measurement model:

$\mathbf{z}_k = h_k(\mathbf{x}_k) + v_k, \quad v_k \sim \mathcal{N}(0, R_k)$

Nonlinear implementation equations:

Computing the predicted state estimate:

$\hat{\mathbf{x}}_k(-) = f_{k-1}(\mathbf{x}_{k-1}(+), \mathbf{u}_k)$

Computing the predicted measurement:

$\hat{\mathbf{z}}_k = h_k(\hat{\mathbf{x}}_k(-))$

Linear approximation equations:

$$\Phi_{k-1} \approx \frac{\partial f_k}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}_{k-1}(-)}$$

$$H_k \approx \frac{\partial h_k}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}_{k-1}(-)}$$

Conditioning the predicted estimate on the measurement:

$\hat{\mathbf{x}}(+) = \hat{\mathbf{x}}(-) + K_k(\mathbf{z}_k - \hat{\mathbf{z}}_k)$

Computing the a priori covariance matrix:

$P_k(-) = \Phi_{k-1} P_{k-1}(+) \Phi_{k-1}^T + Q_{k-1}$

Computing the Kalman gain:

$K_k = P_k(-) H_k^T [H_k P_k(-) H_k^T + R_k]^{-1}$

Computing the a posteriori covariance matrix:

$P_k(+) = I - K_k H_k P_k(-).$

TABLE 6.1: Discrete Extended Kalman filter equations

State	Variable	Description
x_1	X	x-coordinate in the inertial frame
x_2	Y	y-coordinate in the inertial frame
x_3	θ	heading angle of the rover
x_4	v_x	rover linear velocity
x_5	ω	rover angular velocity
x_6	δp_{gpsN}	GPS north component position error
x_7	δp_{gpsE}	GPS east component position error
x_8	δp_{gpsD}	GPS down component position error
x_9	δp_{insN}	INS north component position error
x_{10}	δp_{insE}	INS east component position error
x_{11}	δv_{insN}	INS north component velocity error
x_{12}	δv_{insE}	INS east component velocity error
x_{13}	ρ_{insN}	INS north axis tilt error
x_{14}	ρ_{insE}	INS east axis tilt error
x_{15}	δp_{insD}	INS down component position error
x_{16}	δv_{insD}	INS down component velocity error

TABLE 6.2: kalman states

$$\mathbf{x}_k = (\mathbf{I}_5 + T_s) f_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_k) + w_k). \tag{6.29}$$

Using Eq. 6.12 in Eq. 6.29,

$$\begin{aligned}
\begin{bmatrix} x_1^k \\ x_2^k \\ x_3^k \\ x_4^k \\ x_5^k \end{bmatrix} = & \left(\mathbf{I}_5 + T_s \cdot \begin{bmatrix} 0 & 0 & 0 & \cos x_3^{k-1} & x_0 \sin x_3^{k-1} \\ 0 & 0 & 0 & \sin x_3^{k-1} & x_0 \cos x_3^{k-1} \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & \frac{x_4^{k-1}}{m} \\ 0 & 0 & 0 & \frac{-x_4^{k-1}}{mx_0^2+I} & \frac{x_0}{mx_0^2+I} \end{bmatrix} \right) \begin{bmatrix} x_1^{k-1} \\ x_2^{k-1} \\ x_3^{k-1} \\ x_4^{k-1} \\ x_5^{k-1} \end{bmatrix} \\
& + T_s \cdot \left(\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{-cx_4^{k-1}}{rm} & \frac{cx_4^{k-1}}{rm} \\ \frac{x_3^{k-1}}{r(mx_0^2+I)} & \frac{x_3^{k-1}}{r(mx_0^2+I)} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{(x_0 F_{ry} + M_r) x_4^{k-1}}{m} \\ \frac{F_{rx} x_3^{k-1}}{mx_0^2+I} \end{bmatrix} + w^{k-1} \right).
\end{aligned} \tag{6.30}$$

In the above equations,$\{\mathbf{x}_k | k = 0, 1, 2, 4, \dots\}$ is a vector valued sequence. To get Φ_{k-1}, the first term of the right hand side of Eq. 6.30 is linearized about the estimated trajectory by taking the partial derivative of $f_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_k)$ with respect to $\mathbf{x}$:

$$\Phi_{k-1} = \mathbf{I}_5 + T_s \frac{\partial f_{k-1}(x_{k-1})}{\partial x}\Big|_{x=\hat{x}(-)}. \tag{6.31}$$

The matrix of partial derivatives or the *Jacobian matrix* from Eq. 6.31 is represented by

$$\begin{bmatrix} 0 & 0 & -\hat{x}_4^{k-1}\sin\hat{x}_3^{k-1} + \hat{x}_5^{k-1}d_0\cos\hat{x}_3^{k-1} & \cos\hat{x}_3^{k-1} & d_0\sin\hat{x}_3^{k-1} \\ 0 & 0 & \hat{x}_4^{k-1}\cos\hat{x}_3^{k-1} + \hat{x}_5^{k-1}d_0\sin\hat{x}_3^{k-1} & \sin\hat{x}_3^{k-1} & -d_0\cos x_3^{k-1} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \dfrac{\hat{x}_5^{k-1}}{m} & \dfrac{\hat{x_4}^{k-1}}{m} \\ 0 & 0 & 0 & -2\dfrac{\hat{x}_4^{k-1}}{md_0 + I} & 0 \end{bmatrix}. \tag{6.32}$$

The integrated state estimation uncertainty covariance matrix P is a 16×16 matrix defined by

$$P_{\text{int}} = \left[\begin{array}{c|c|c} P_{\text{rov}} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & P_{\text{gps}} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & P_{\text{ins}} \end{array}\right]. \tag{6.33}$$

From table 6.1, P is updated at every timestep. However, the designer will have to give its initial values. The initial values of P_{rov}, P_{gps} and P_{ins} can be set to reflect the designer's uncertainty of the initial states:

$$\begin{aligned} P_{\text{rov}}^0 &= \begin{bmatrix} p_{1\text{rov}} & 0 & 0 & 0 & 0 \\ 0 & p_{2\text{rov}} & 0 & 0 & 0 \\ 0 & 0 & p_{3\text{rov}} & 0 & 0 \\ 0 & 0 & 0 & p_{4\text{rov}} & 0 \\ 0 & 0 & 0 & 0 & p_{5\text{rov}} \end{bmatrix} \\ P_{\text{gps}}^0 &= \begin{bmatrix} p_{1\text{gps}} & 0 & 0 \\ 0 & p_{2\text{gps}} & 0 \\ 0 & 0 & p_{3\text{gps}} \end{bmatrix} \\ P_{\text{ins}}^0 &= \begin{bmatrix} p_{1\text{ins}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & p_{2\text{ins}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{3\text{ins}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_{4\text{ins}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{5\text{ins}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & p_{6\text{ins}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p_{7\text{ins}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{8\text{ins}} \end{bmatrix}. \end{aligned} \tag{6.34}$$

The integrated process noise covariance matrix $\mathbf{Q}$ is a 16×16 matrix defined by

$$Q_{\text{int}} = \left[\begin{array}{c|c|c} Q_{\text{rov}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & Q_{\text{gps}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & Q_{\text{ins}} \end{array}\right]. \tag{6.35}$$

The process noise covariance matrices $Q_{\text{rov}}, Q_{\text{gps}}$ and Q_{ins} have the forms:

$$\begin{aligned}
Q_{\text{rov}} &= \begin{bmatrix} q_{1\text{rov}} & 0 & 0 & 0 & 0 \\ 0 & q_{2\text{rov}} & 0 & 0 & 0 \\ 0 & 0 & q_{3\text{rov}} & 0 & 0 \\ 0 & 0 & 0 & q_{4\text{rov}} & 0 \\ 0 & 0 & 0 & 0 & q_{5\text{rov}} \end{bmatrix} \\
Q_{\text{gps}} &= \begin{bmatrix} q_{1\text{gps}} & 0 & 0 \\ 0 & q_{2\text{gps}} & 0 \\ 0 & 0 & q_{3\text{gps}} \end{bmatrix} \\
Q_{\text{ins}} &= \begin{bmatrix} q_{1\text{ins}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_{2\text{ins}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_{3\text{ins}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_{4\text{ins}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_{5\text{ins}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_{5\text{ins}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & q_{5\text{ins}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & q_{5\text{ins}} \end{bmatrix}.
\end{aligned} \tag{6.36}$$

We next turn our attention to the measurement model $\mathbf{z}_k = h_k(\mathbf{x}_k) + v_k$. The GPS module outputs 3 position components according to the north, east and down frame. The INS module outputs 3 position components and one heading angle. The wheel encoder outputs a linear velocity and an angular velocity component. In all, 9 outputs are observed by the measurement model. The Measurement sensitivity matrix has a 9×16 dimension and is defined as follows:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The last variable needed to implement the extended Kalman filter is the measurement uncertainty covariance matrix R. The values of the covariance matrix reflects the noise associated with the sensor modules. In our work, the integrated measurement uncertainty covariance matrix is a 9×9 matrix defined by

$$R_{\text{int}} = \left[\begin{array}{c|c|c|c} R_{\text{gps}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & R_{\text{ins}} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & R_{\text{lenc}} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & R_{\text{renc}} \end{array}\right]. \tag{6.37}$$

The components R_{gps}, R_{ins}, R_{lenc} and R_{renc} are defined as follows:

$$\begin{aligned} R_{\text{gps}} &= \begin{bmatrix} r_{1\text{gps}} & 0 & 0 \\ 0 & r_{2\text{gps}} & 0 \\ 0 & 0 & r_{3\text{gps}} \end{bmatrix} \\ R_{\text{ins}} &= \begin{bmatrix} r_{1\text{ins}} & 0 & 0 & 0 \\ 0 & r_{2\text{ins}} & 0 & 0 \\ 0 & 0 & r_{3\text{ins}} & 0 \\ 0 & 0 & 0 & r_{4\text{ins}} \end{bmatrix} \\ R_{\text{lenc}} &= \begin{bmatrix} r_{1\text{l}} & 0 \\ 0 & r_{2\text{l}} \end{bmatrix} \\ R_{\text{renc}} &= \begin{bmatrix} r_{1\text{r}} & 0 \\ 0 & r_{2\text{r}} \end{bmatrix}. \end{aligned} \tag{6.38}$$

6.3 Multiagent Reconfigurable Control of Rover

The multiagent framework presented in this work is utilized to control the rover.

Figure 6.5 shows our solution to the fault tolerant control system. As a reminder, the monitor agent A_m takes the role of the FDD component in our work. The planner agent A_p and the control agents $A_c^k, k = 1, \ldots, 4$ take the role of the reconfiguration component. The action of A_m is passed to A_p. The planner decides on which controller to use and passes it on to A_c. It also decides on the parameter settings of the system and changes it accordingly.

6.3.1 Planner Agent

The planner agent A_p is realized with two memory banks. The solution memory $\mathcal{M}_s$ stores the parameters of learned controllers $c_1, \ldots, c_n$. When an unmodelled fault occurs,

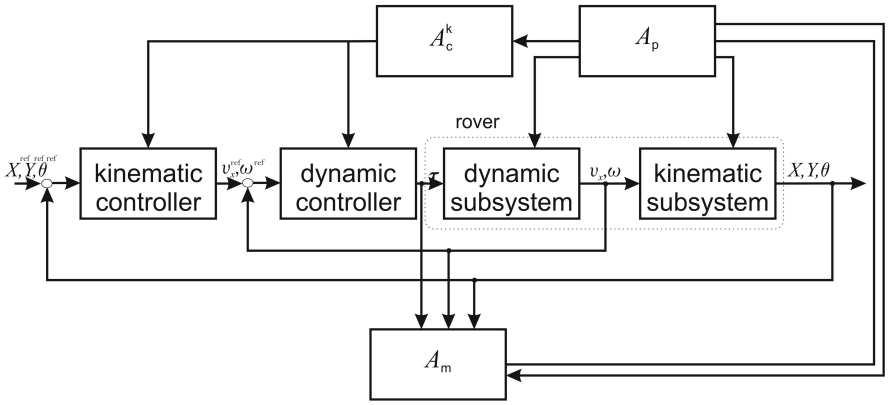


FIGURE 6.5: Multiagent control of rover. Monitoring agent A_m takes the role of the FDD component. Similarly, A_p and A_c^k take the role of the reconfiguration component in a fault tolerant control system.

A_p instruct the control agents to learn a new controller. If a solution is found, the newly learned control parameters are stored and $\mathcal{M}_s = c_1, \ldots, c_{n+1}$. If no solution is found, A_p shuts down the system and waits for an operator intervention.

The resource memory $\mathcal{M}_r$ stores parameter settings of the system. In our case study, the resource memory of the planner agent contains the Kalman filter parameters of the rover. So $\mathcal{M}_r = \{b_{\text{gps}}, b_{\text{ins}}, b_{\text{lenc}}, b_{\text{renc}}\}$ where b denotes the Kalman filter structure. By changing the Kalman filter parameters, specifically the measurement noise matrix R, A_p has the ability to control whether a sensor should be turned on or off.

6.3.2 Control Agent

For our case study, we define a continuous state set $\mathcal{S} \in \mathbb{R}^5$. The components are listed in Table 6.3.

Inputs	Description	Range
1	Bias term	1
v_x	Velocity of rover	$[-3, 3]$ ms^{-1}
ω	Angular velocity of AGV	$[-\pi, \pi]$ rad/s
e_v	Error between reference linear velocity and actual linear velocity	$[-6, 6]$ ms^{-1}
e_ω	Error between reference angular velocity and actual angular velocity	$[-2\pi, 2\pi]$ rad/s

TABLE 6.3: Inputs of network

At any time t, the input state to the reinforcement learning algorithm can be represented as $\mathbf{s}_t = \begin{bmatrix} 1 & v_x & \omega & e_v & e_\omega \end{bmatrix}^T$. We also define a normalized continuous action set $\mathcal{A} \in [-1, 1]$ for each agent A_c^k. The action set represents the torque that can be applied to a motor.

The function approximator depicting the states and action sets is shown in Figure 6.6 The network consists of four inputs corresponding to the rover model, one bias term, four hidden units and four outputs corresponding to the actions of the four control agents.

6.3.3 Monitor Agent

We have defined 6 sensor modules in our case study. A FuzzyART network is created for each sensor and one additional network is created to detect unmodelled faults. We define the action set of the monitor agent $\mathcal{A} \in \mathbb{R}^7$. The action set contains the fault status $\mathcal{F}$ and is defined as $a_t^{A_m} = \{f_{\text{enc1}}, f_{\text{enc2}}, f_{\text{enc3}}, f_{\text{enc4}}, f_{\text{gps}}, f_{\text{ins}}, f_{\text{unc}}\}$. Each fault status can take on any integer value $f_d \geq 1$ where d represents one of the seven networks. The default value is $f_d = 1$ and has the meaning that the module in question is not faulty.

The sensor data are sampled and stored in a buffer. The monitor agent processes the data every 64 timesteps. Unmodelled faults are detected by presenting the mean squared error of the linear and angular velocity for the last 64 timesteps as inputs to one of the FuzzyART network. If the data pattern is significantly different from the default mode, an error is flagged.

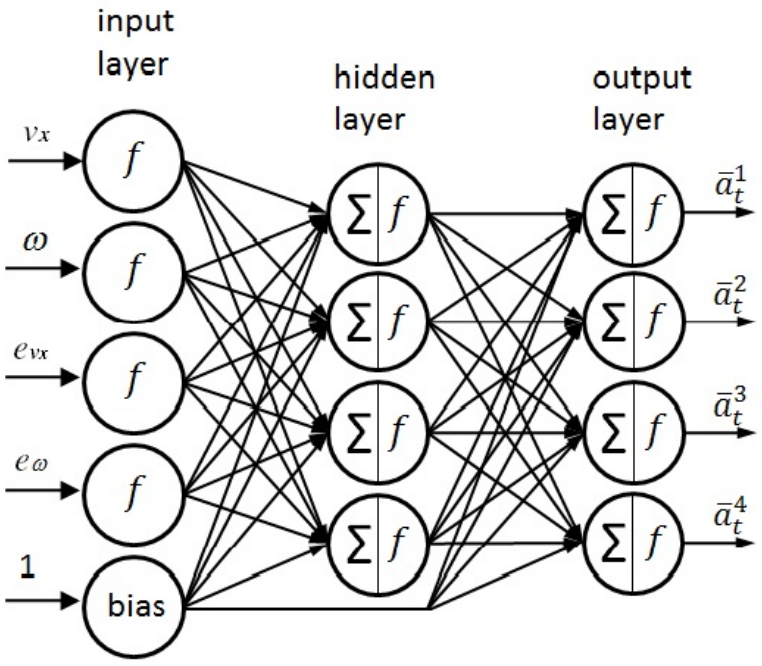


FIGURE 6.6: Direct policy search neural network with 4 states, 1 bias term, 4 hidden units and 4 outputs

6.4 Simulation Results

The multiagent fault tolerant control system presented in this work was tested on the rover model. It was assumed that no predefined controllers exist in the planner agent's memory. It is therefore necessary for the control agents A_c^k to initially learn a controller. Table 6.4 lists down the numerical values for all parameters that are used in the simulation.

Table 6.4: Numerical values for system parameters

Variable	**Value**	**Description**
		Rover
r	0.08 m	effective rolling radius of wheel
a	0.3 m	distance from COM to rear wheel axes
b	0.25 m	distance from COM to front wheel axes
c	0.2 m	body width of rover in metres
m	15 kg	total mass of rover
I	0.578 kgm^2	rover moment of inertia
x_{ICR}	-0.2 m	fixed coordinate of rover ICR
μ_s	0.01	longitudinal coulumb friction coefficient
μ_l	0.3	lateral coulumb friction coefficient
g	9.8 kgms^{-2}	gravitational acceleration
		Sensor
τ_{hor}	37 sec	horizontal position error correlation time
τ_{ver}	32 sec	vertical position error correlation time
Ω^2_{sch}	0.00124 rad/s	Schuler frequency
$\Omega_{\odot}$	7.3×10^{-5} rad/s	earth rotation rate
		Kalman filter
$p_{1\text{rov}}$	0.001	initial rover covariance matrix element,$P^0_{\text{rov}}(1,1)$
$p_{2\text{rov}}$	0.001	initial rover covariance matrix element,$P^0_{\text{rov}}(2,2)$
$p_{3\text{rov}}$	0.001	initial rover covariance matrix element,$P^0_{\text{rov}}(3,3)$
$p_{4\text{rov}}$	0.001	initial rover covariance matrix element,$P^0_{\text{rov}}(4,4)$
$p_{5\text{rov}}$	0.001	initial rover covariance matrix element,$P^0_{\text{rov}}(5,5)$
$p_{1\text{gps}}$	101.435	initial GPS covariance matrix element,$P^0_{\text{gps}}(1,1)$
$p_{2\text{gps}}$	82.863	initial GPS covariance matrix element,$P^0_{\text{gps}}(2,2)$
$p_{3\text{gps}}$	2.355	initial GPS covariance matrix element,$P^0_{\text{gps}}(3,3)$
$p_{1\text{ins}}$	0	initial INS covariance matrix element,$P^0_{\text{rov}}(1,1)$
$p_{2\text{ins}}$	0	initial INS covariance matrix element,$P^0_{\text{rov}}(2,2)$
$p_{3\text{ins}}$	0	initial INS covariance matrix element,$P^0_{\text{rov}}(3,3)$
$p_{4\text{ins}}$	0	initial INS covariance matrix element,$P^0_{\text{rov}}(4,4)$
$p_{5\text{ins}}$	0	initial INS covariance matrix element,$P^0_{\text{rov}}(5,5)$
$p_{6\text{ins}}$	0	initial INS covariance matrix element,$P^0_{\text{gps}}(1,1)$
$p_{7\text{ins}}$	1	initial INS covariance matrix element,$P^0_{\text{gps}}(2,2)$
$p_{8\text{ins}}$	0	initial INS covariance matrix element,$P^0_{\text{gps}}(3,3)$
$q_{1\text{rov}}$	2	rover process noise matrix element,$Q_{\text{rov}}(1,1)$
$q_{2\text{rov}}$	2	rover process noise matrix element,$Q_{\text{rov}}(2,2)$

Table 6.4: (continued)

Variable	Value	Description
$q_{3\text{rov}}$	0.001	rover process noise matrix element,$Q_{\text{rov}}(3,3)$
$q_{4\text{rov}}$	2	rover process noise matrix element,$Q_{\text{rov}}(4,4)$
$q_{5\text{rov}}$	0.001	rover process noise matrix element,$Q_{\text{rov}}(5,5)$
$q_{1\text{gps}}$	0.0520	GPS process noise matrix element,$Q_{\text{gps}}(1,1)$
$q_{2\text{gps}}$	0.0425	GPS process noise matrix element,$Q_{\text{gps}}(2,2)$
$q_{3\text{gps}}$	0.0012	GPS process noise matrix element,$Q_{\text{gps}}(3,3)$
$q_{1\text{ins}}$	0	INS process noise matrix element,$Q_{\text{ins}}(1,1)$
$q_{2\text{ins}}$	0	INS process noise matrix element,$Q_{\text{ins}}(2,2)$
$q_{3\text{ins}}$	0	INS process noise matrix element,$Q_{\text{ins}}(3,3)$
$q_{4\text{ins}}$	0	INS process noise matrix element,$Q_{\text{ins}}(4,4)$
$q_{5\text{ins}}$	4.558×10^{-16}	INS process noise matrix element,$Q_{\text{ins}}(5,5)$
$q_{6\text{ins}}$	4.558×10^{-16}	INS process noise matrix element,$Q_{\text{ins}}(1,1)$
$q_{7\text{ins}}$	0	INS process noise matrix element,$Q_{\text{ins}}(2,2)$
$q_{8\text{ins}}$	0	INS process noise matrix element,$Q_{\text{ins}}(3,3)$
$r_{1\text{gps}}$	0.1	GPS measurement noise matrix element,$R_{\text{gps}}(1,1)$
$r_{2\text{gps}}$	0.1	GPS measurement noise matrix element,$R_{\text{gps}}(2,2)$
$r_{3\text{gps}}$	0.1	GPS measurement noise matrix element,$R_{\text{gps}}(3,3)$
$r_{1\text{ins}}$	0.01	INS measurement noise matrix element,$R_{\text{ins}}(1,1)$
$r_{2\text{ins}}$	0.01	INS measurement noise matrix element,$R_{\text{ins}}(2,2)$
$r_{3\text{ins}}$	0.01	INS measurement noise matrix element,$R_{\text{ins}}(3,3)$
$r_{4\text{ins}}$	0.01	INS measurement noise matrix element,$R_{\text{ins}}(4,4)$
$r_{1\text{l}}$	2	left encoder measurement noise matrix element,$R_{\text{lenc}}(1,1)$
$r_{2\text{l}}$	2	left encoder measurement noise matrix element,$R_{\text{lenc}}(2,2)$
$r_{1\text{r}}$	2	right encoder measurement noise matrix element,$R_{\text{renc}}(1,1)$
$r_{2\text{r}}$	2	rignt encode measurement noise matrix element,$R_{\text{renc}}(2,2)$

6.4.1 Training phase

The first step is to learn a controller for the dynamic subsystem as in Figure 6.5. The dynamic controller regulates the states $x_4 = v_x$ and $x_5 = \omega$ of the rover system. The system is sampled at a frequency of $f = 100$ Hz. The sampling time $T_s = 1/f = 0.01s$. The learning process is broken down into episodes where each episode consist of 4000 timesteps. The horizon time of each episode can be calculated as $t_T = 4000 \cdot T_s = 40s$.

In each *controller learning* episode, a random reference linear velocity v_x^{ref} and a reference angular velocity ω^{ref} is generated. The control agents A_c^k try to minimize the *velocity errors* of the system. The velocity errors are defined as:

$$e_v = v_x^{\text{ref}} - v_x, \tag{6.39}$$

and

$$e_\omega = \omega^{\text{ref}} - \omega. \tag{6.40}$$

The episode is terminated when $e_v < \epsilon_v$ and $e_\omega < \epsilon_\omega$ or when the horizon time t_T is reached. After several episodes, the system is tested to see whether a reliable controller has been learned.

This leads us to the next step of learning a controller for the task. A kinematic controller which regulates the states $x_1 = X$, $x_2 = Y$ and $x_3 = \theta$ is put in place in a cascaded structure. The kinematic controller in our work is implemented as a proportional controller with gain K_X for the state $x_1 = X$, K_Y for the state $x_2 = Y$ and K_θ for the state $x_3 = \theta$. The reference values for the dynamic controller are generated by the output of the kinematic controller.

At the start of a *controller test* episode, reference end positions $X_{\text{end}}^{\text{ref}}, Y_{\text{end}}^{\text{ref}}$ are randomly generated. The reference end position is broken down into small increments,

$$\delta X = \frac{X_{\text{end}}^{\text{ref}}}{m},$$

and

$$\delta Y = \frac{Y_{\text{end}}^{\text{ref}}}{m}.$$

Here, m is an integer. At each timestep n, the reference values for the kinematic controller are obtained by,

$$X_n^{\text{ref}} = X_{n-1}^{\text{ref}} \text{sgn}(X_{\text{end}}^{\text{ref}} - X_{n-1}^{\text{ref}}) \delta X, \tag{6.41}$$

and

$$Y_n^{\text{ref}} = Y_{n-1}^{\text{ref}} \text{sgn}(Y_{\text{end}}^{\text{ref}} - Y_{n-1}^{\text{ref}}) \delta Y. \tag{6.42}$$

The reference heading angle θ_n^{ref} at each timestep n is calculated as

$$\theta_n^{\text{ref}} = \arctan \frac{Y_n^{\text{ref}} - Y_n}{X_n^{\text{ref}} - X_n}. \tag{6.43}$$

For every timestep n in a controller test episode, the kinematic controller outputs the reference velocities v_n^{ref} and ω_n^{ref}. The control agents A_c^k try to track the reference velocities with the controller obtained during the controller learning episodes. The performance of the controller can be measured by taking the mean squared error of the velocities for the whole episode. If the episode takes N timesteps to terminate, the mean squared error can be calculated as follows:

$$\text{MSE}(\mathbf{v}, \mathbf{v}^{\text{ref}}) = \frac{1}{N} \sum_{i=1}^{N} (v_i - v_i^{\text{ref}})^2, \tag{6.44}$$

$$\text{MSE}(\omega, \omega^{\text{ref}}) = \frac{1}{N} \sum_{i=1}^{N} (\omega_i - \omega_i^{\text{ref}})^2. \tag{6.45}$$

The controller learning and testing process is repeated until the mean squared errors of the velocities are smaller than a threshold,

$$\text{MSE}(\mathbf{v}, \mathbf{v}^{\text{ref}}) < \xi_v \wedge \text{MSE}(\omega, \omega^{\text{ref}}) < \xi_\omega. \tag{6.46}$$

Figure 6.7 and Figure 6.8 show the mean squared error of the linear and angular velocity as the training episodes increase.

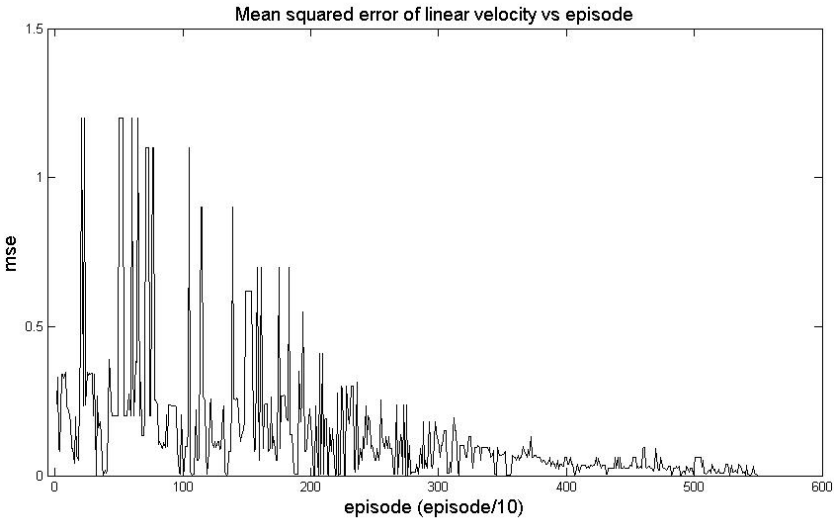


FIGURE 6.7: Mean squared error of linear velocity during training.

After the control agents have learned a controller, it is stored in $\mathcal{M}_s$ of the planner agent's memory.

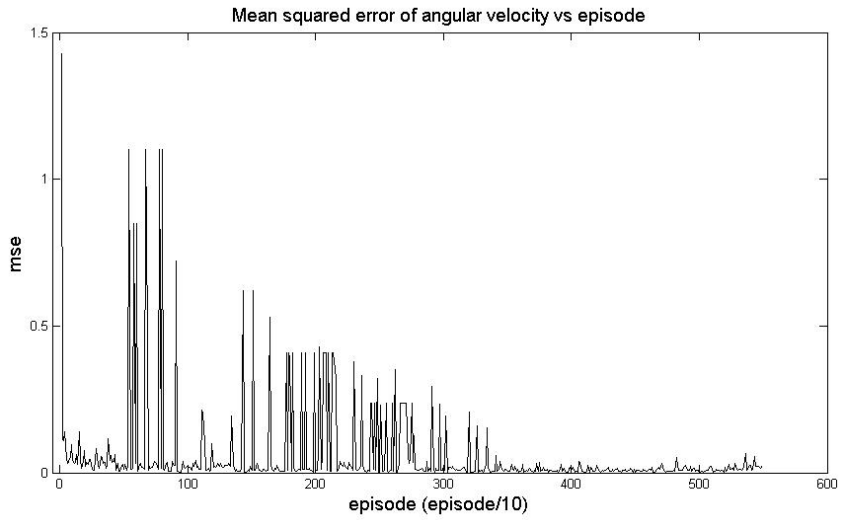


FIGURE 6.8: Mean squared error of angular velocity during training.

6.4.2 Executing a control task

The controller that was stored can now be chosen by the planner agent A_p to execute a control task.

It is important to note that no explicit training was done for the monitor agent. This is because the ART network is an online classification algorithm. The only necessary step that must be taken is to ensure that the system starts in the default mode (error free) at the beginning of the control task.

To test the multiagent fault tolerant control system, several cases were studied.

6.4.2.1 Case 1:

In the first case, no faults were introduced. This is the default operation mode. The agents were given a reference position. The agents must control the rover so that it reaches the reference position. Figure 6.9 illustrates how the agents track the reference linear velocity, reference angular velocity, reference heading angle, reference X position and reference Y position. The solid lines in the figure represent the actual value and the dashed lines represent the reference values.

The agents were given a reference coordinate of $[35\text{m}, 35\text{m}]$. The controller learned by the control agents succeeded in tracking the reference values. The rover arrived at its reference position after about 15 seconds and completely stops after about 29 seconds. Since no fault was introduced, the monitor agent did not detect any faults and outputs the action $a_t^{A_m} = \{1, 1, 1, 1, 1, 1, 1\}$ for all time t. The control agents executed its task until its goal was reached.

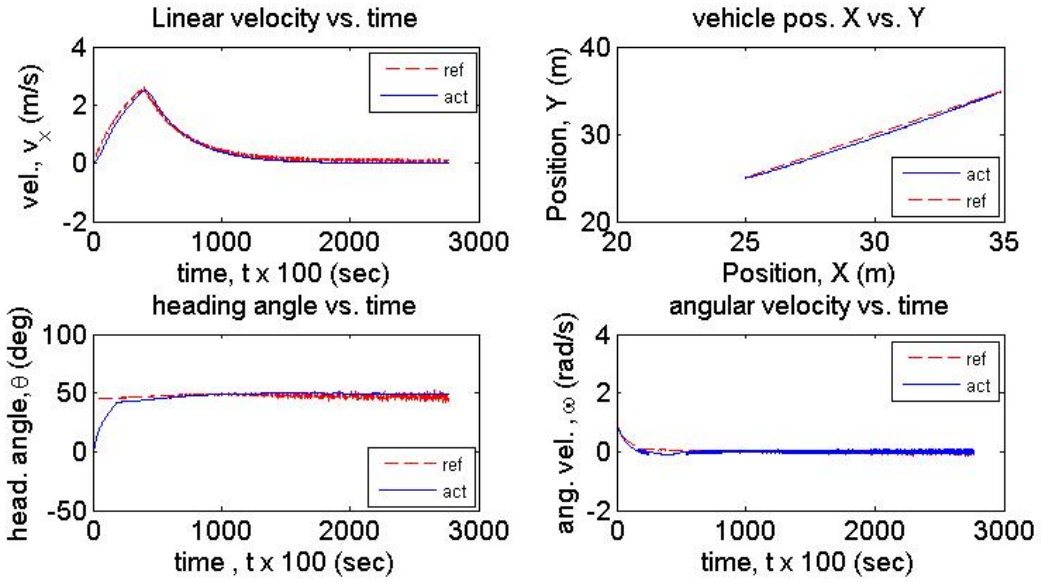


FIGURE 6.9: Default operation mode. No faults are introduced. The graph shows how the reference values are tracked.

Figure 6.10 shows the actions taken by the agents. It shows that the fluctuation of the action taken by the control agents were very high towards the end of the task. This is due to the nonlinear nature of the task. The agents needed to compensate the nonlinear resistive forces acting on the rover. However, this could result in jerky movements by the rover. A low pass filter was added to lessen the fluctuation of the actions thereby resulting in much smoother movements.

It was heuristically determined that a passband of 0 to 30 Hz and at least 60 dB of attenuation in the stop frequency of 40 Hz to the Nyquist frequency (50 Hz) yielded the best result. Figure 6.11 shows the result of the filtered actions.

6.4.2.2 Case 2:

In the second case, we introduced a sensor fault to the system after time $t > T_{\text{tr1}}$. Specifically, one of the encoders was turned off to simulate a faulty encoder. As with the first case, a reference position is given to the agents. The agents must control the rover so that it reaches the reference position. Figure 6.12 and Figure 6.13 show how the agents respond to faults.

From Figure 6.12, an encoder fault was triggered at $t > 500$ time steps. This is shown as a dashed line in the top left graph of Figure Figure 6.12. For the dashed line, the value of 0.25 means that no fault has been triggered. The value of 0.75 means that a fault has been triggered. So in Figure 6.12, a fault is only triggered in encoder 1. The output of the monitor agent A_m for encoder 1 changes value shortly thereafter and $a_t^{A_m} = \{2, 1, 1, 1, 1, 1, 1\}$ for roughly 5 sec $< t <$ 6 sec. This is shown as a solid line in

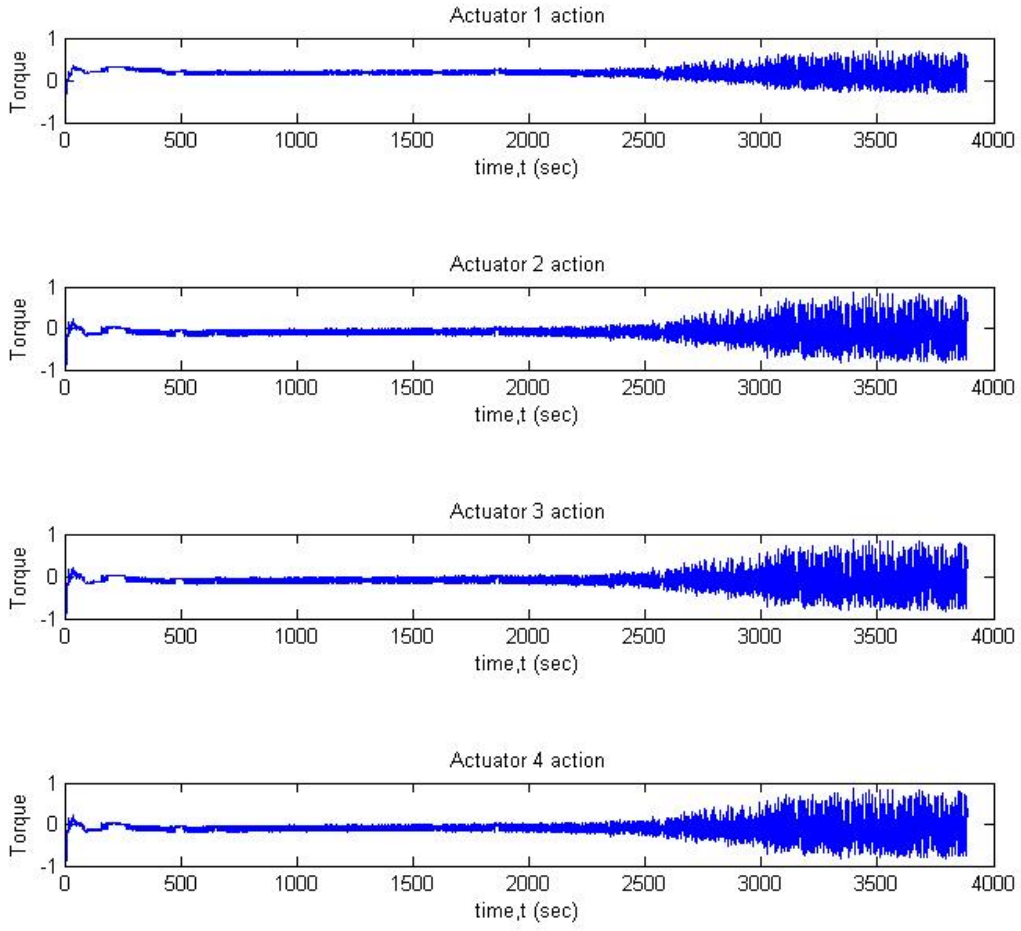


FIGURE 6.10: Torque applied to actuators (agent action) during the course of simulation. Action fluctuation was very high.

the graph. For the solid line, the value 1 means that the sensor is working properly. The value 2 means that the sensor is faulty. The fault value is passed to A_p. Upon receiving the fault status from A_m, A_p changed the parameter settings of the system. In this case, the parameter setting was the Kalman filter measurement uncertainty value of the sensor. By changing it to a very large value, the faulty sensor readings are discarded. After the system is reconfigured, the output of the monitor agent outputs $a_t^{A_m} = \{1, 1, 1, 1, 1, 1, 1\}$ for roughly $t > 6$ sec.

Figure Figure 6.13 shows how the linear velocity, angular velocity, heading angle and position are tracked. The solid lines are the actual values of the system and dashed lines are the reference values. It can be seen from the top left graph of Figure 6.13 that a sudden jump in the velocity value is detected. This is caused by a faulty sensor reading when a sensor is turned off. This anomaly can also be seen in the graphs of the heading angle and the angular velocity. After reconfiguration of the system, the agents successfully completed the control task and reached their goal after about 20

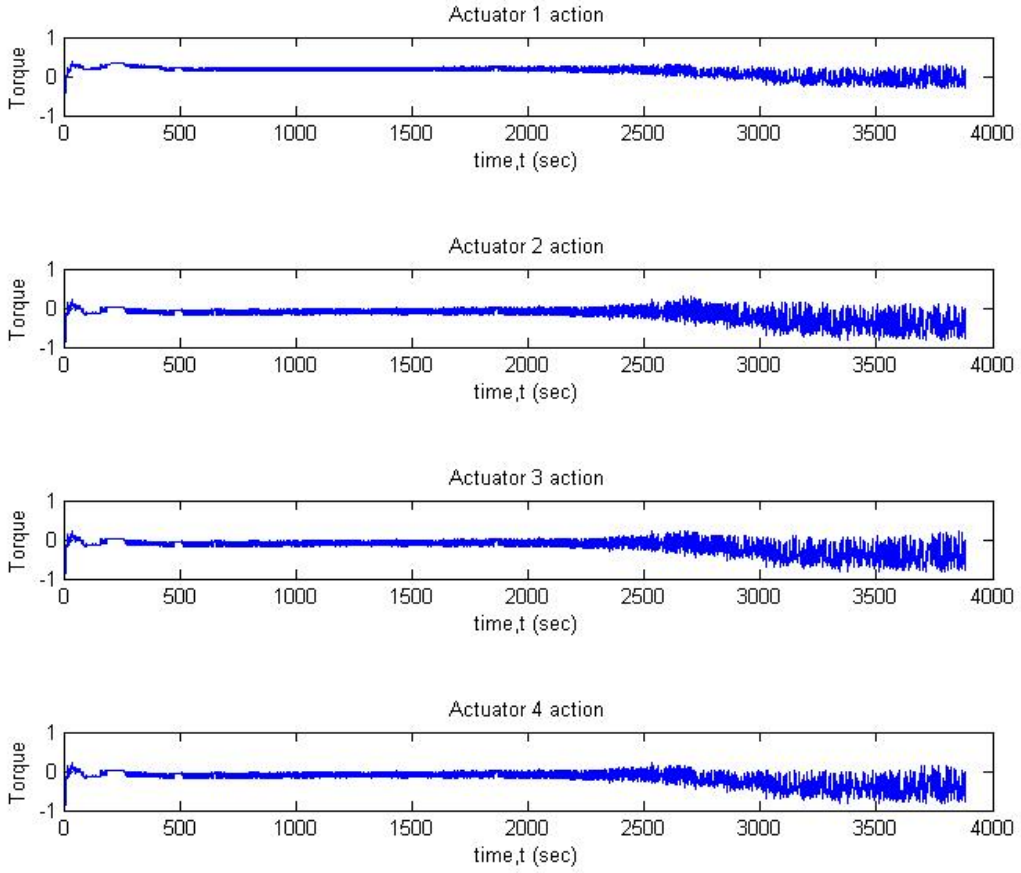


FIGURE 6.11: Torque applied to actuators (agent action) during the course of simulation with a low pass filter implemented. This resulted in smoother rover movements.

sec. It is interesting to note that the overall performance of the control system is not badly affected. This can be seen from the top right graph of Figure 6.13. The rover closely tracks its reference position even in the presence of faulty sensors. This can be attributed to the sensor fusion technique that was discussed earlier in this chapter.

6.4.2.3 Case 3:

In the third case, we introduce an actuator fault to the system after time $t > T_{\text{tr2}}$. This can also be seen as introducing an unmodelled fault since we did not explicitly train the agents to control the rover with anything less than four functioning actuators. As before, a reference position is given to the agents. The agents must control the rover so that it reaches the reference position. Figure 6.14 shows how the monitor agent A_m responds when one actuator is turned off.

The top graph of Figure 6.14 shows the response of the monitor agent A_m. The other four graphs show the fault trigger for each wheel. The monitor agent does not detect

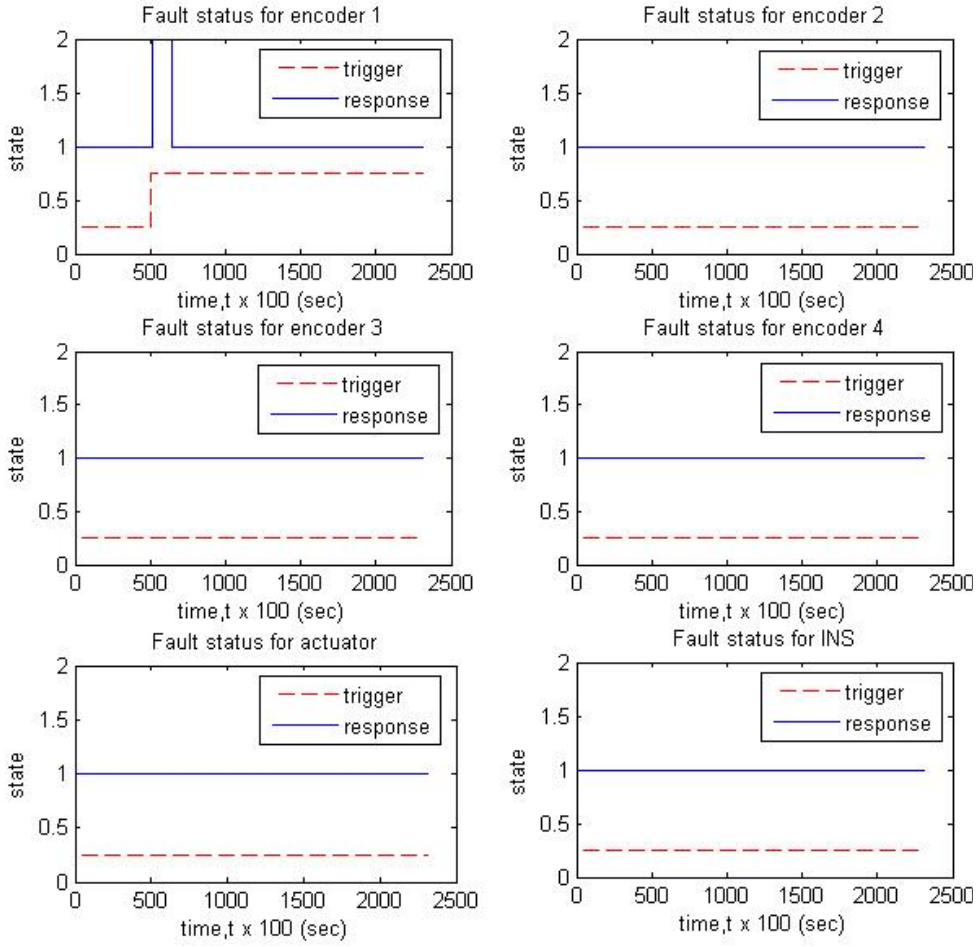


FIGURE 6.12: A sensor fault was introduced. The output value of A_m changes value for one of the sensor modules. A value of 1 means that the sensor is properly working. A value of 2 means that there is a fault. The graph shows that after reconfiguration, the output of the monitor agent returns no fault status.

that a fault has occurred. This is due to the fact that the system has learned a controller that is robust enough to accommodate faulty actuators and complete its intended task. This is shown in Figure 6.15 where the system was able to track its reference values even though one actuator is faulty.

Figure 6.16 to Figure 6.19 suggest that as long as there are redundant actuators that allow the system to maintain a course (i.e. there is a way to produce torque on the left and right side of the rover), the system is able to accommodate faulty actuators and complete its task.

In Figure 6.16, we triggered a fault by turning off the front left motor and the front right motor. Since the rover was still able to produce torque on both sides (through rear left and rear right motors), the agents found a way to control the rover and complete its task.

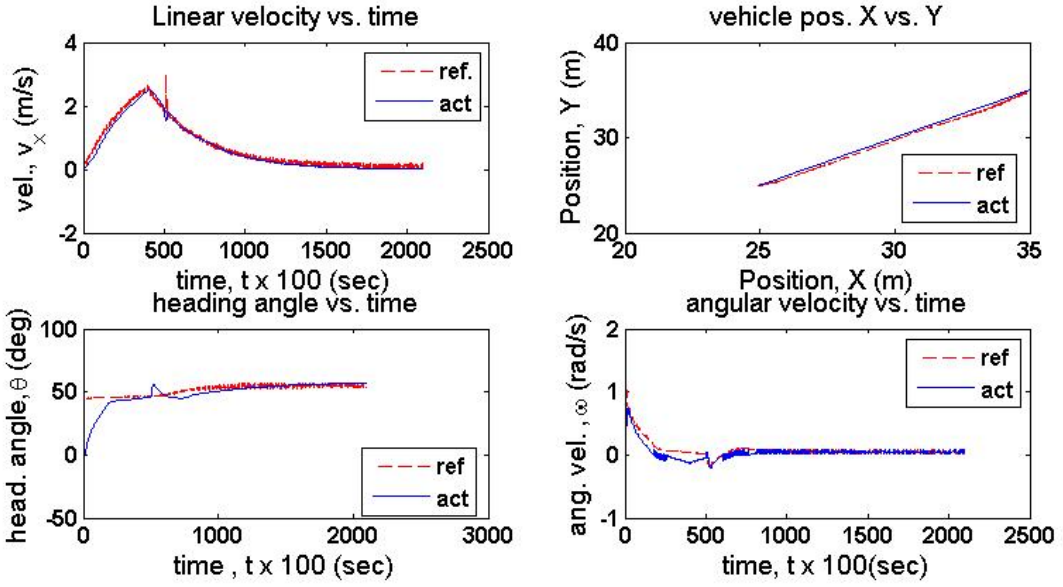


FIGURE 6.13: A sensor fault was introduced. The figures above show how the reference values are tracked. After reconfiguring system parameters, the rover is able to complete the control task.

This can be seen from Figure 6.17. The agents were able use the remaining actuators to manipulate the rover to track its reference values very closely. However, the rover took longer to get to its goal. This due to the fact that when the rover is close to its goal, having only two motors to work with affects the agents' capability to manipulate the rover quickly when making small moves. This can be seen by the oscillating reference values seen in the bottom left and bottom right graph of Figure 6.17.

In Figure 6.18, we triggered a fault by turning off the front left motor and the rear right motor. As with the previous scenario, the rover was still able to produce torque on both sides (through rear left and front right motors) and the agents found a way to control the rover and complete its task.

Figure 6.19 shows how the rover is controlled by the agents. The agents were able use the remaining actuators to manipulate the rover to track its reference values. As with the previous scenario, the rover took longer to get to its goal and the same argument applies to this scenario.

In Figure 6.20, we triggered a fault by turning off the front right motor and the rear right motor. The rover is not able to produce torque on the right side. A_m detects a fault a few seconds after the fault was triggered.

This is due to the fact that the control agents were still able to control the rover for a while, even after the fault was triggered. This is shown inFigure 6.21 where the difference between the reference and actual values of the system begins to grow as time passes. After A_m realizes that the control agents are no longer capable to maneuvering

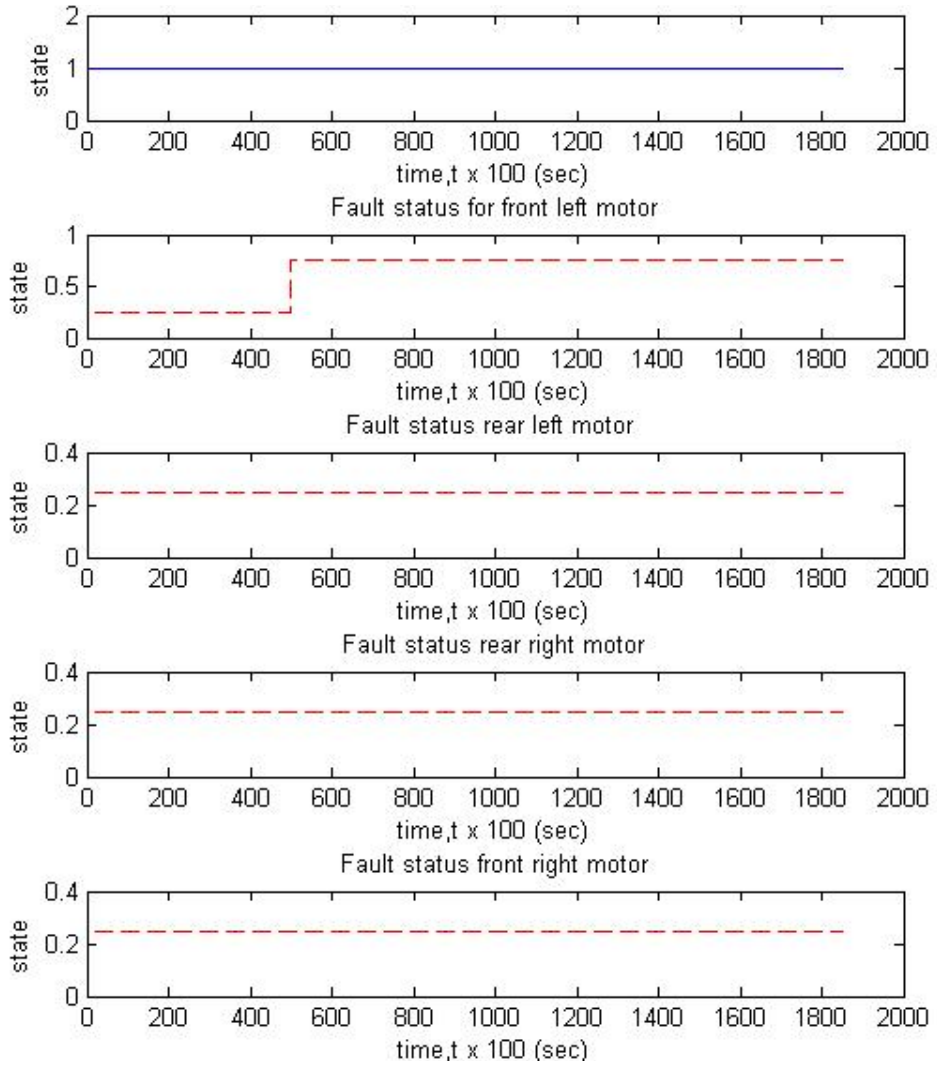


FIGURE 6.14: A_m response to actuator faults. The figure shows the simulation of a fault in the front left motor of the rover.

the rover, it raises a fault and outputs $a_t^{A_m} = \{1, 1, 1, 1, 1, 1, 2\}$ at time $t \approx 13$ sec. The fault information is sent to A_p. The planner agent shuts down the system since there is no way to maneuver the rover in a controlled manner.

Looking at the top right graph of Figure 6.21, the agents were able to maneuver the rover to its goal position but was unable to stop. It starts to veer off to its right since the torque of the rover only comes from the left side (through the rear left and front left motors). This results in the mean squared error of the linear and angular velocity to grow thus allowing A_m to detect the fault.

6.4.2.4 Case 4

In the next scenario, an actuator delay to was introduced in the simulation. As before, a reference position was given to the agents. The agents must control the rover so that

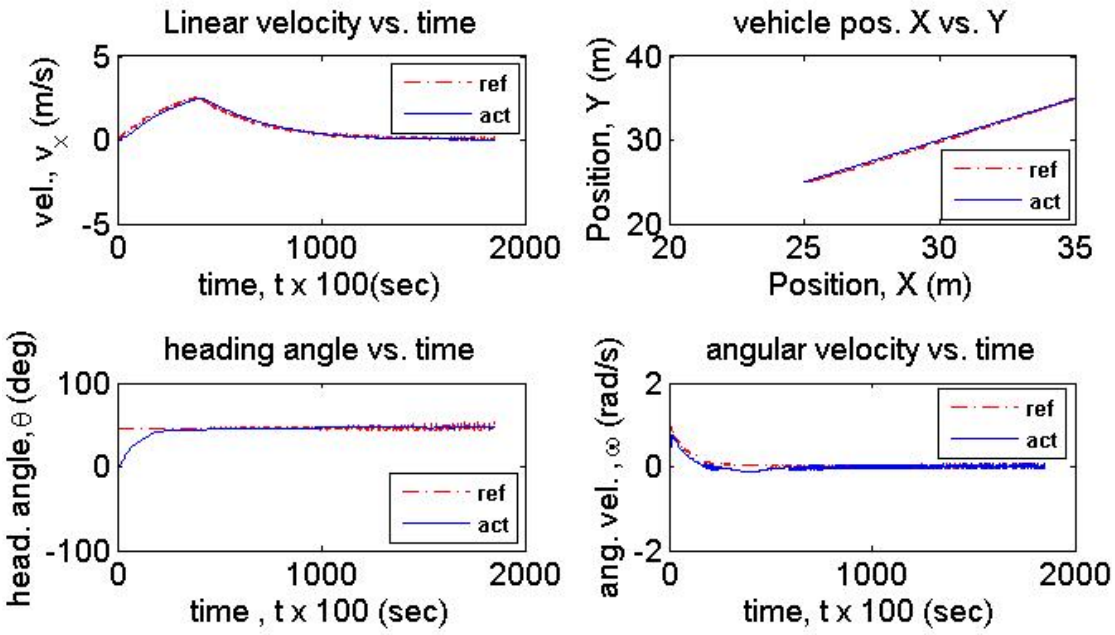


FIGURE 6.15: Reference value tracking during one episode. The system was still able to complete the control task in the presence of an actuator fault

it reaches the reference position. Figure 6.22 - Figure 6.25 show how the rover responds to the fault.

Figure 6.22 depicts how the system responds when an actuator delay of $T_d = 3T_s$ was simulated. The controller was able to accommodate the delay and the agents successfully controlled the rover to its goal. It can be seen however, that there was an added noise to the angular velocity due to the fault.

Figure 6.23 illustrates how the system reacts to an actuator delay of $T_d = 5T_s$. The agents were still able to complete their task. However, it can be seen that an oscillation of the heading angle started to form. Spikes in the desired linear and angular velocity was also apparent. This was due to the fact that the agents were trying to compensate for the fault. It had to explore actions that would minimize the error in the system.

Figure 6.24 shows a more degraded performance when an actuator fault of $T_d = 9T_s$ was presented. The agents were still able to complete the task. A significant oscillation of the heading angle and angular velocity appeared at the end of the simulation. Because of the delay in torque effect. The agents think that the actions taken were not sufficient, so it tried to compensate for the action. After sufficient data had been sampled, it settled on the actions that minimized the error and allowed it to complete its task.

Figure 6.25 shows the agents failure to control the rover when an actuator fault of $T_d = 10T_s$ was presented in the simulation. The graphs show that the agents were not able to predict the system's response when they took their actions. The fault was too

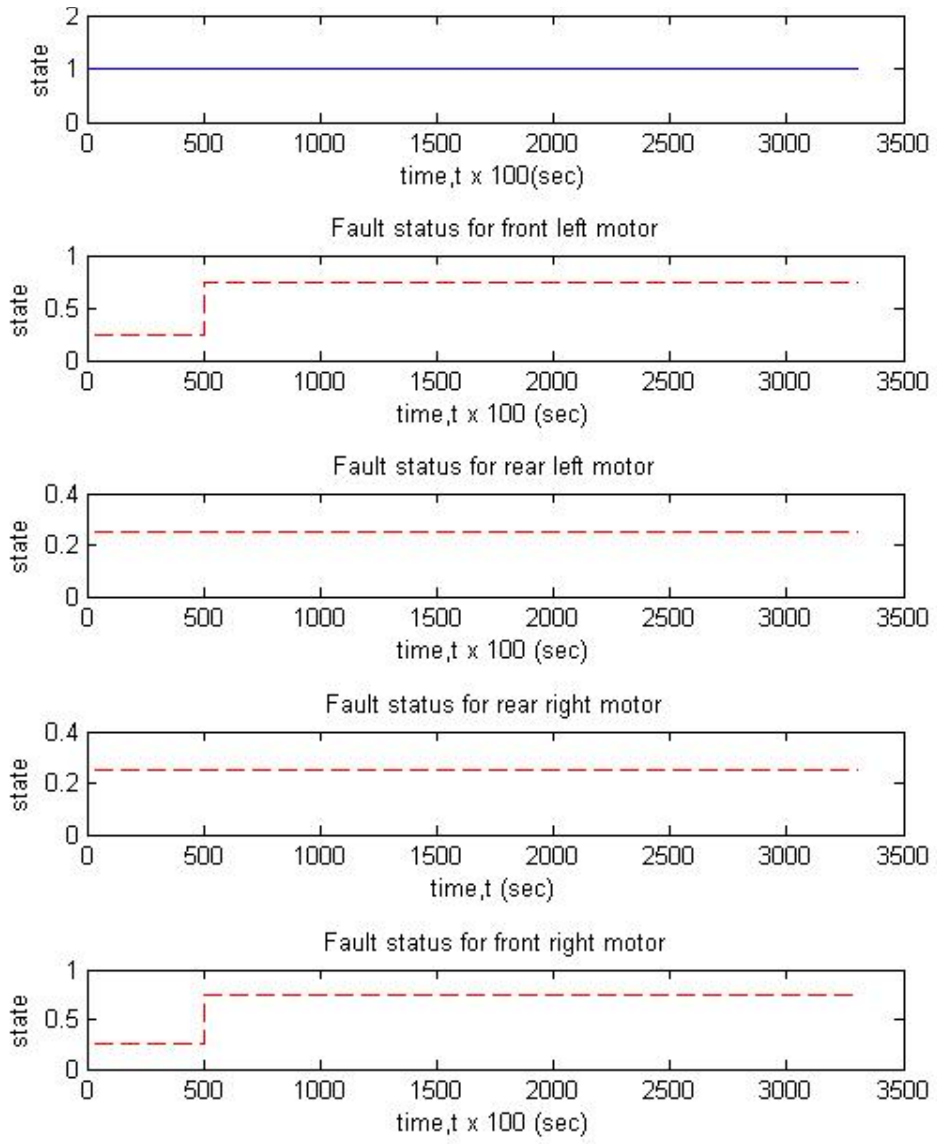


FIGURE 6.16: A_m response to front left motor and front right motor failure. No fault was detected due to robust nature of controller. See Figure 6.17.

severe for the system to recover. The system shuts down after 100 seconds and waits for an operator intervention.

6.4.3 Controller Comparison

In order to measure the performance of the proposed controller relative to other methods, a comparison with the controller designed by Kozlowski (2004) was carried out. Kozlowski (2004) designed a cascading control structure for a 4-wheel skid-steering vehicle similar to our rover model. It used a feedback linearization technique on the kinematic subsystem and a backstepping technique on the dynamic subsystem. In the

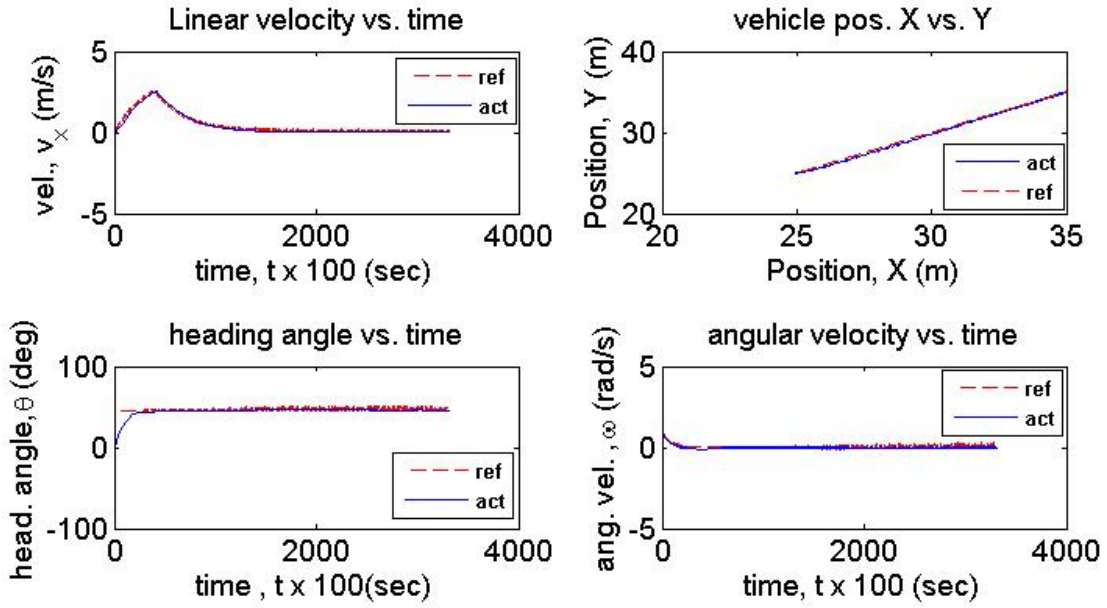


FIGURE 6.17: Reference value tracking corresponding to Figure 6.16. The system was still able to complete the control task in the presence of actuator faults.

following, our proposed controller will be referenced to as *Rl* whereas the controller by Kozlowski (2004) will be referenced to as *Fbl*.

Figure 6.26 shows the tracking performance of the two controllers with respect to a desired trajectory. It can be seen from the figure Fbl takes longer to converge to the trajectory compared to Rl. This is due to the ability of reinforcement learning algorithms to maximize its return according to the reward function.

In the next simulation, an actuator delay was introduced as was discussed in the previous section. Figure 6.27 - Figure 6.30 show the performance of both controllers.

Figure 6.27 shows the performance of the controllers when an actuator delay of $T_d = 3T_s$, where T_d is the delay time and T_s is the sampling time, was introduced. Both controllers were still able to control the rover. A small damped oscillation about the trajectory was seen to develop by Rl at the start of the tracking task. Fbl took longer to converge to the trajectory and overshot the trajectory at the end of the task.

In Figure 6.28 an actuator delay of $T_d = 4T_s$ was introduced. It can be seen from the figure that the Fbl showed a very large tracking error compared to its counterpart. Rl was able to reconfigure the system and functioned with a sligthly degraded performance.

Figure 6.29 illustrates the performance of both controllers when presented with an actuator delay of $T_d = 9T_s$. Fbl failed to track the desired trajectory. Rl on the other still manages to track the desired trajectory albeit with a more degraded performance compared to the previous case.

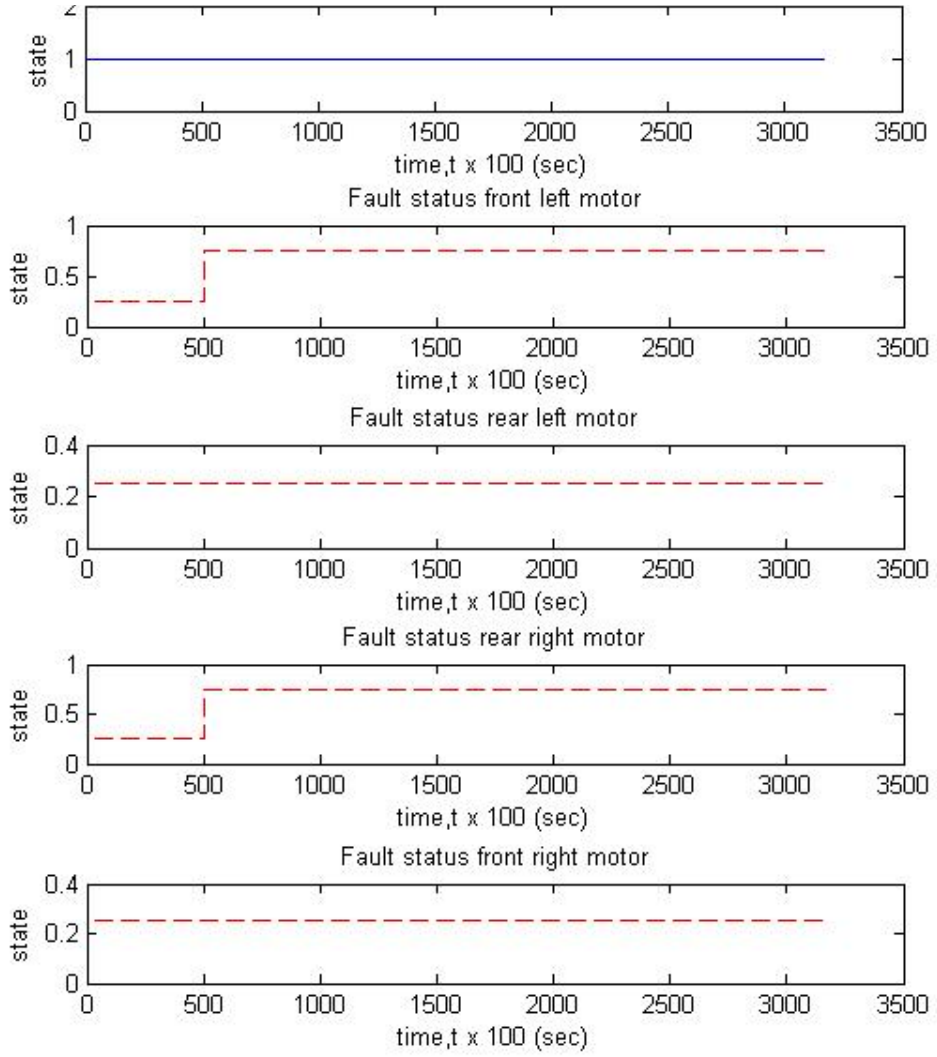


FIGURE 6.18: A_m response to front left motor and rear right motor failure. No fault detected due to robust nature of controller. Figure 6.19.

In the final simulation an actuator delay of $T_d = 10T_s$ was introduced. Figure 6.30 illustrates that both Rl and Fbl failed to track the desired trajectory. This is due to the fact that the error was too great to accommodate, even for Rl.

From the simulations that were carried out, it was shown that our proposed controller (Rl) outperformed the controller designed by Kozlowski (2004) (Fbl) with respect to trajectory tracking. It was shown that both controllers were robust enough to accommodate an actuator delay of up to $T_d = 3T_s$. Thereafter the delay was too much for Fbl to overcome. This was shown in Figure 6.28 and subsequent figures. On the other hand, Rl was able to deal with a delay of up to $T_d = 9T_s$. Actuator delays of more than $9T_s$ caused Rl to fail. We therefore conclude that Rl is significantly more robust compared to Fbl with regards to an actuator delay fault.

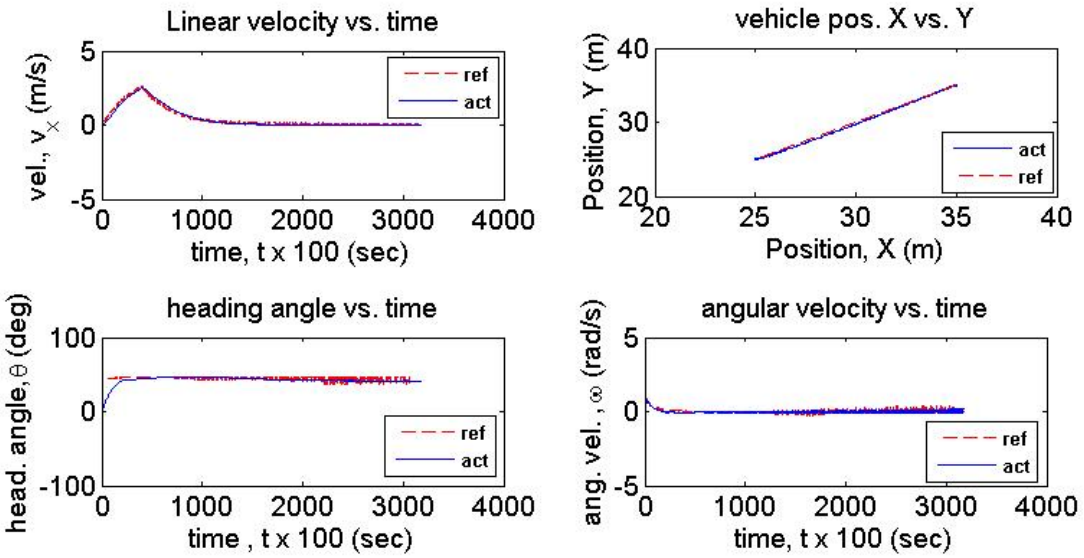


FIGURE 6.19: Reference value tracking corresponding to Figure 6.18. The system was still able to complete the control task in the presence of actuator faults

6.5 Chapter Conclusion

We have demonstrated in this chapter how our proposed approach to fault tolerant control problem can be implemented on a nonlinear system. A model for a 4-wheel skid-steering vehicle or simply called rover was derived.

In order to simulate a complete system which allows the demonstration of dealing with faulty sensors and actuators, three different types of sensors were modelled to measure variables such as rover position and velocity. A global position system (GPS) module, an inertial navigation system (INS) module, and four encoders situated on the wheels of the rover make up the sensory system of the rover. The sensors were then fused using an extended Kalman filter.

We then showed how the multiagent framework fits into the control structure of the rover. The implementation of A_p, A_m and A_c was discussed. A_m took the role of the fauld detection and diagnosis component of the fault tolerant control architecture. On the other hand, A_p and A_c took the role of the reconfiguration and control component. The multiagent fault tolerant control system was tested on the rover model. It was assumed that no predefined controllers exist in the planner agent's memory. The control agents A_c had to initially learn a controller.

After learning was complete, several cases were simulated and the response of the system was discussed. In the first case, no fault was introduced. The rover succeeded in reaching its goal. The monitor agent did not raise a fault alarm during the entire duration of simulation.

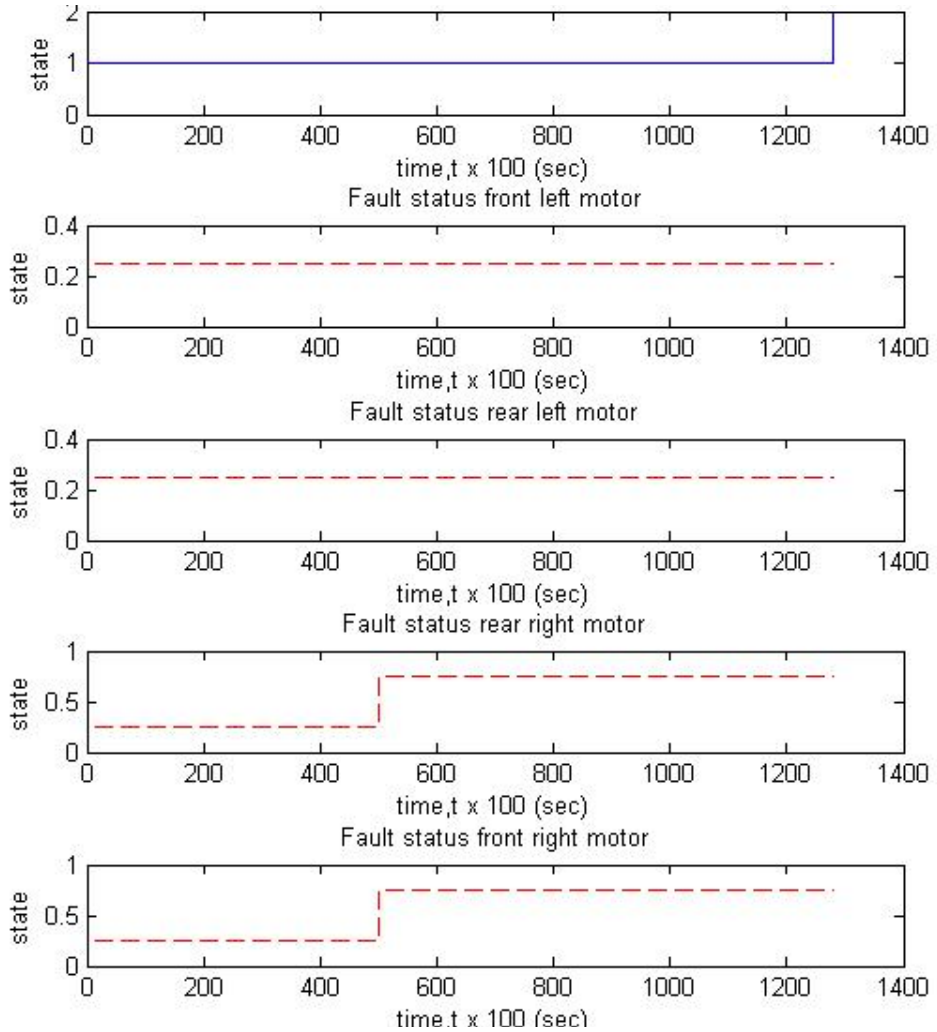


FIGURE 6.20: A_m response to front right motor and rear right motor fault. The monitor agent detects a fault after realizing that the rover cannot be controlled.

In the second case, a sensor fault was introduced in the simulation. The monitor agent was able to detect the fault and alert the other agents. The system was then reconfigured before proceeding with the original control task. The rover again succeeded in reaching its goal.

In the third and fourth case, actuator faults were introduced. It was shown that the agents were able to work together to detect faults that were triggered. We showed that our design was robust to actuator faults as long as the agents were given the means to produce torque on both sides. The monitor agent detected a fault as soon as the system was no longer able to maneuver the rover. It was also shown that the controller was robust enough to deal with actuator delays.

A comparison to another type controller was carried out. It was determined that the proposed controller had a higher robustness with regards to an actuator delay fault.

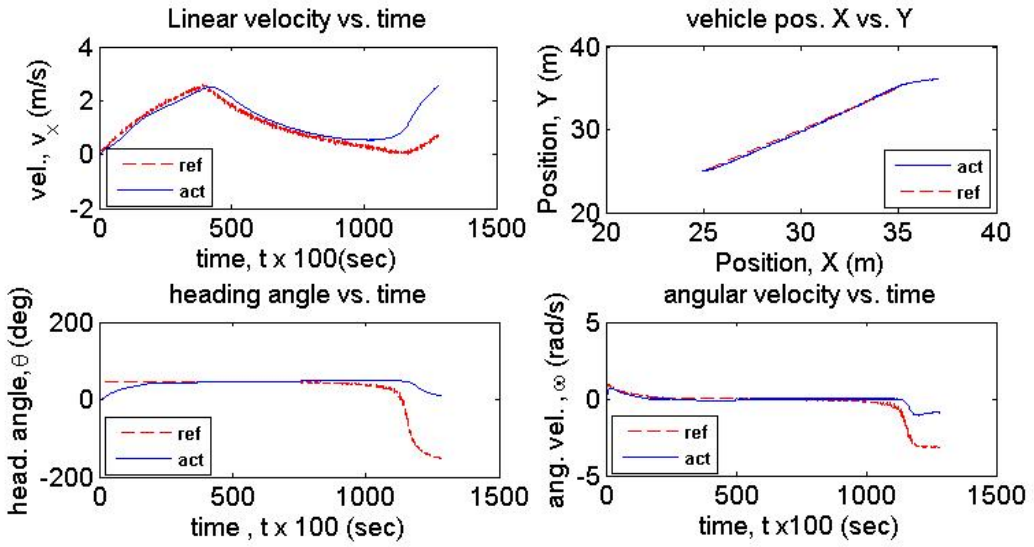


FIGURE 6.21: Rover trajectory after both motors on right side were turned off. A few seconds after the fault was triggered, agent loses the ability to maneuver the rover.

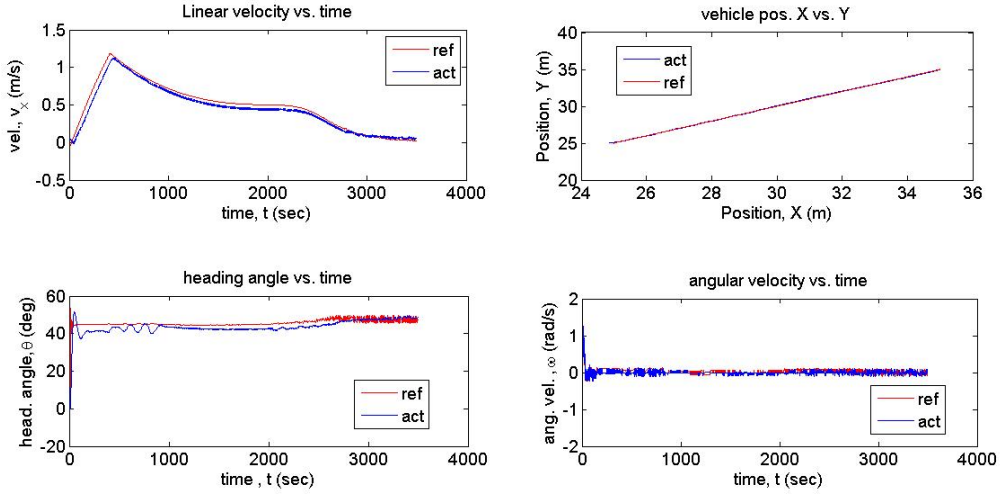


FIGURE 6.22: System response to an actuator delay of $T_d = 3T_s$. The controller was able to accommodate the delay and the agents successfully controlled the rover to its goal.

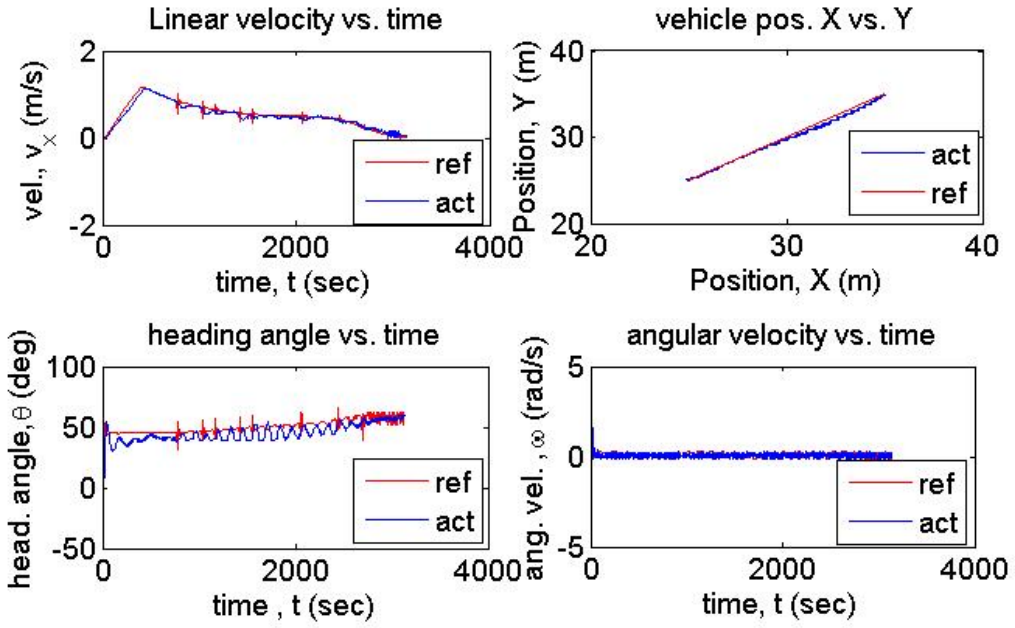


FIGURE 6.23: System response to an actuator delay of $T_d = 5T_s$. The agents were able to control the rover to reach its goal after reconfiguring the system, but with a degraded performance.

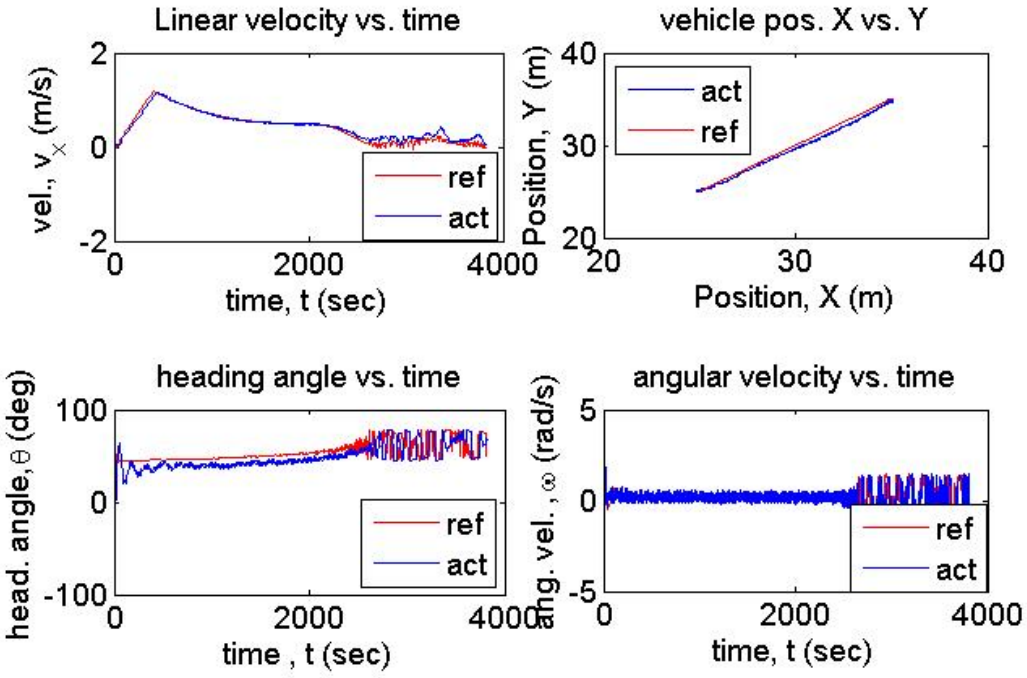


FIGURE 6.24: System response to an actuator delay of $T_d = 9T_s$. As with the previous case, the agents were able to control the rover to reach its goal after reconfiguring the system. However, the performance degradation of the system is more pronounced compared to the previous case.

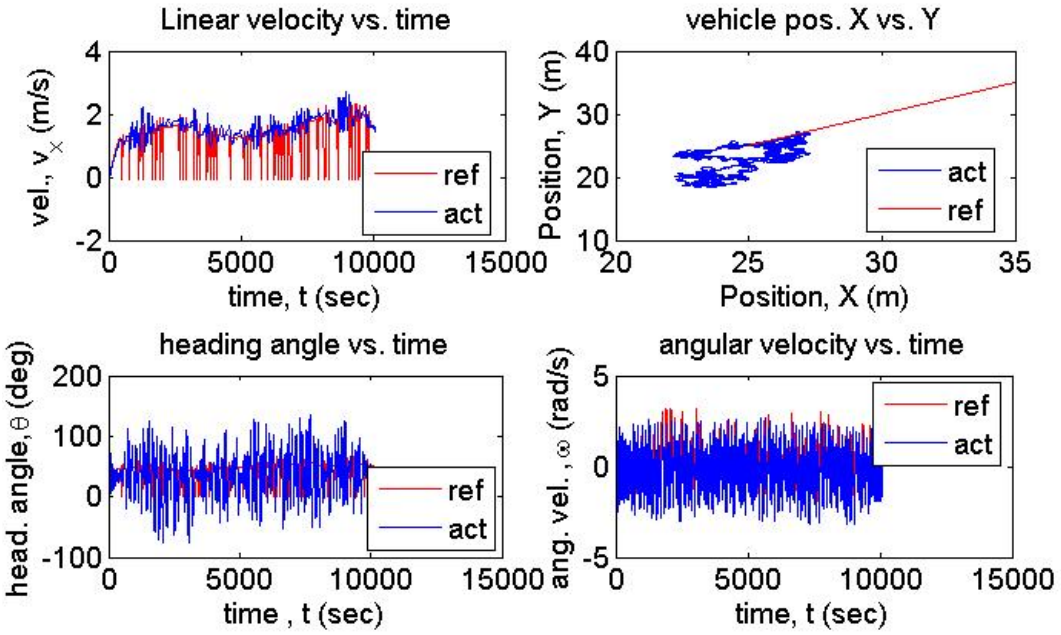


FIGURE 6.25: System response to an actuator delay of $T_d = 10T_s$. The agents failed totally to control the rover. The actuator delay was so great that the system was not able to accommodate the fault. The system shuts down and waits for an operator intervention.

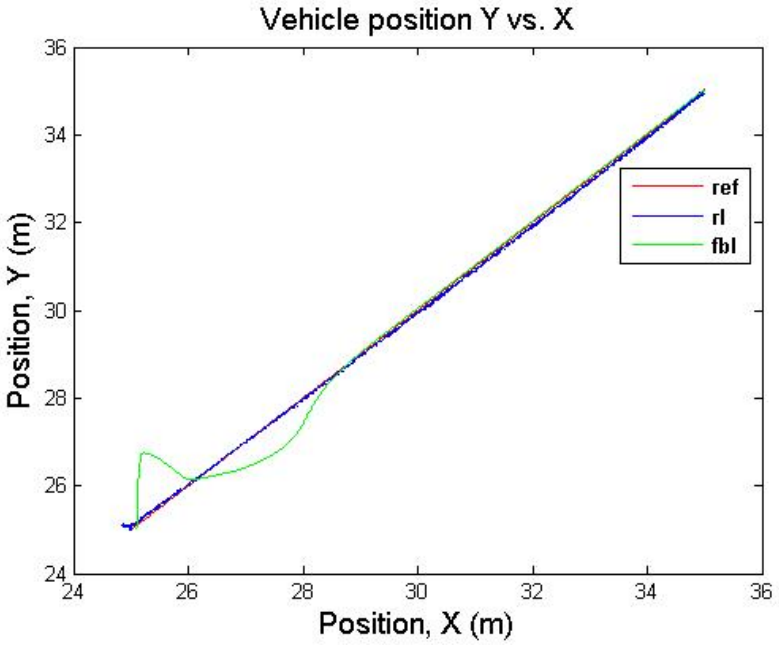


FIGURE 6.26: Tracking performance of the proposed controller (blue line) compared to the controller by (Kozlowski, 2004) (green line). No faults were introduced.

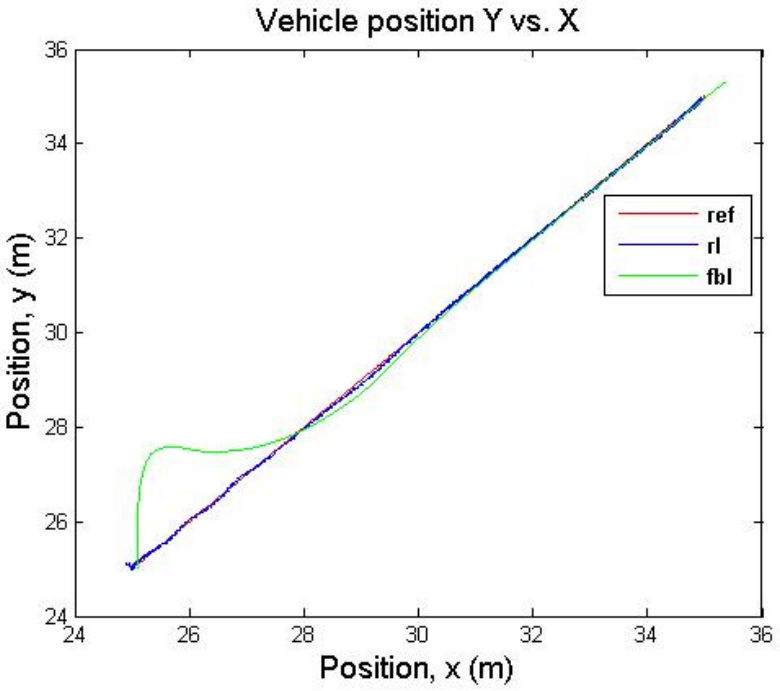


FIGURE 6.27: Tracking performance of Rl compared to Fbl (green line). Actuator delay of $T_d = 3T_s$ was introduced.

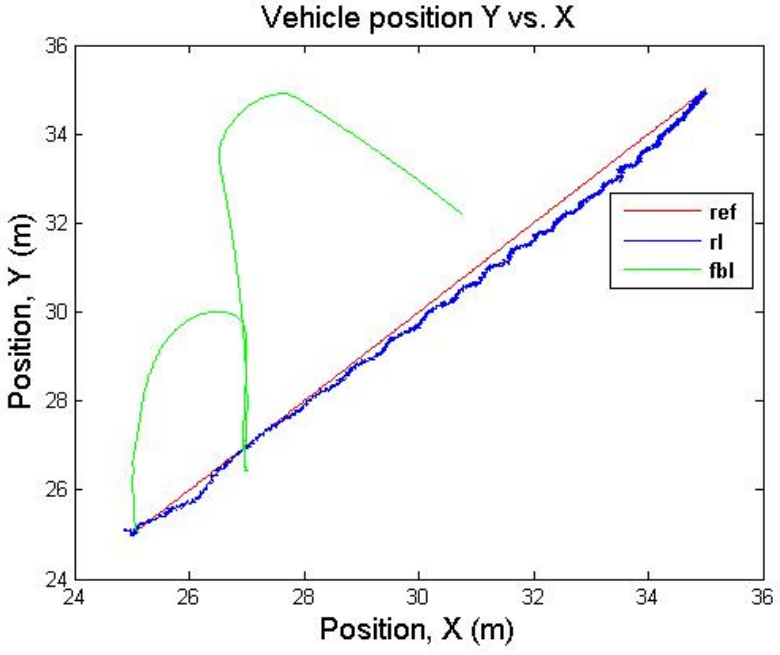


FIGURE 6.28: Tracking performance of Rl compared to Fbl (green line). Actuator delay of $T_d = 4T_s$ was introduced.

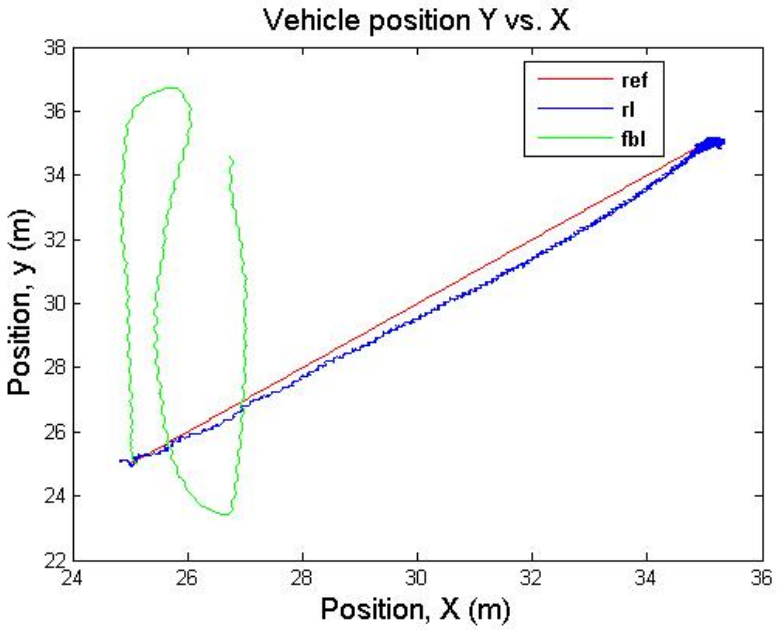


FIGURE 6.29: Tracking performance of Rl (blue line) compared to Fbl (green line). Actuator delay of $T_d = 9T_s$ was introduced.

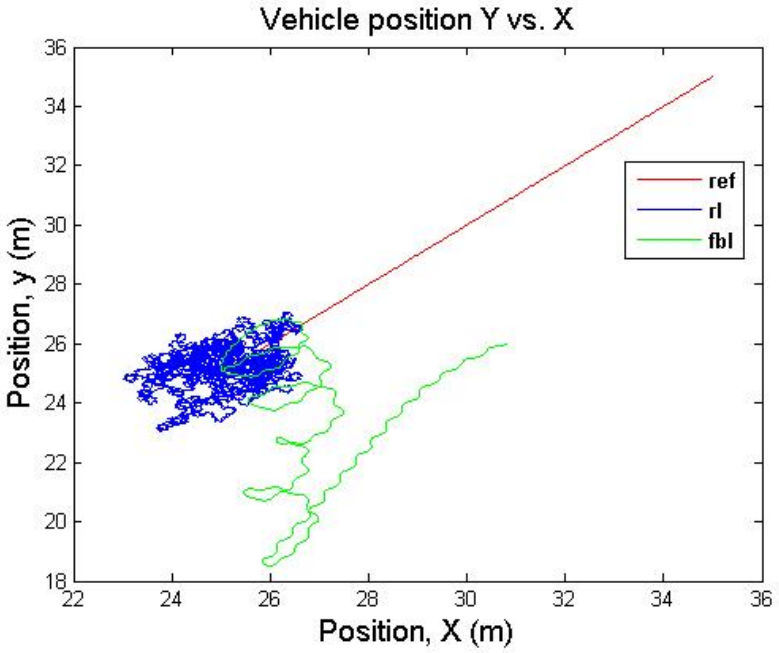


FIGURE 6.30: Tracking performance of Rl (blue line) compared to Fbl (green line). Actuator delay of $T_d = 10T_s$ was introduced.

Chapter 7

Conclusion

This chapter completes our investigation into online multiagent reconfigurable control systems. We will discuss some areas that could be extensions of the methods proposed. We will then summarize our work in the last section of this chapter.

7.1 Future Work

In the scheme presented the planner agent A_p only stores learned controllers and system parameter settings. When a new controller has to be learned, it instructs the control agents to do so. Since this was done on a simulation of a rover, this would not be a problem. In real systems however, exploring new actions could have adverse effects on the system. For example, say the rover is carrying a sensitive instrument when one of its motor has a fault. Letting the control agents explore new actions could result in the rover in behaving wildly thus damaging the potentially very expensive instrument.

A potential solution to this is to store a model of the system in the planner agent's memory. The planner agent would then learn a new controller from the stored model before passing it on to the controller agents. This would minimize the risk of damaging the system.

Work done by Tadepalli and Ok (1994) suggest that a model could be learned online. The authors introduced a model-based reinforcement learning method called H-learning. This algorithm updates the system model by estimating the probability $P(s'|s,a)$ of the reaching a state s' from s every time an action is taken. Let $N(s,a)$ be the number of times action a is taken from state s. Let $N(s,a,s')$ be the number of times taking action a in state s resulted being state s'. The probability of being in state s' by taking an action a in state s is then estimated as

$$P(s'|s,a) = \frac{N(s,a,s')}{N(s,a)}. \tag{7.1}$$

If we define $\mathcal{M}_m$ to be the model memory of A_p, the above equation could be used by A_p to update the system's model residing in $\mathcal{M}_m$. Every time the control agent takes an action a, the planner could update its model by observing the current state s and resulting state s' and using the above equation.

Since a fault would change the model of the system, the learned model would be invalid. This can be fixed by only taking the model of the vehicle for the last k time steps. This would ensure that only a recent model is stored in $\mathcal{M}_m$.

In our work, we have used a multiagent direct policy search algorithm as the internal architecture of a control agent A_c. We have stated in Section 3.3.3 that direct policy search reinforcement learning algorithms are preferred when a simple policy class $\mathcal{P}$ that maps the features of the state to actions can be found. Reinforcement learning with value functions on the other hand is more suitable for high level learning tasks. If the two classes of reinforcement learning were to be combined for fault tolerant control, different types of unmodelled faults could be solved.

Consider driving an underpowered rover up a steep mountain road as discussed in Sutton and Barto (1998) and shown in Figure 7.1. The only solution to this problem is to first build up inertia by moving up and down the slopes.

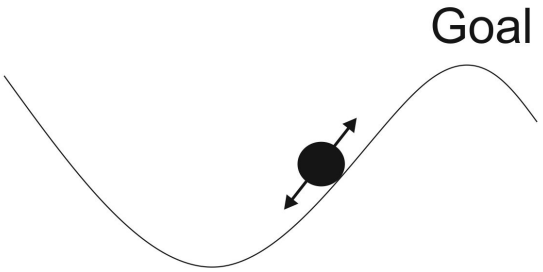


FIGURE 7.1: Driving an underpowered rover up a steep hill. The only solution is to build up inertia by moving up and down the slopes. Figure adapted from Sutton and Barto (1998).

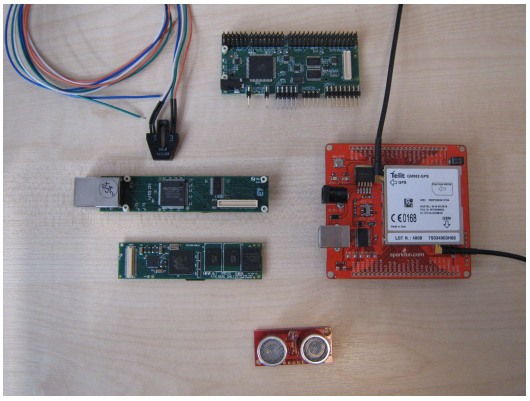
The multiagent direct policy search algorithm used in our work would not be able to solve this problem since this would be considered a high level learning task where a long term strategy is needed. Reinforcement learning with value function algorithms such as the SARSA algorithm is more suitable for this type of task. The only problem with this type of algorithm is that the action set must be discrete since a value would have to be stored for each state-action pair.

A possible solution for this is to switch algorithms when the situation calls for it. When a fault occurs that require a high level strategy, the planner agent would instruct A_c to learn a controller using value function reinforcement learning. The control agent could

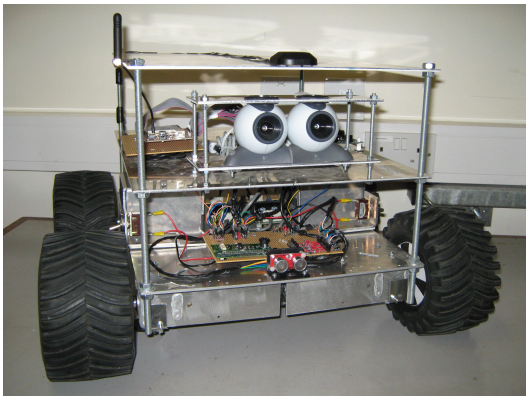
then restrict the action set to a subset $\mathcal{A}_s \in \mathcal{A}$. When the fault has been overcome, A_c could then switch back to using the multiagent direct policy search algorithm to control the rover.

Another extension to the current work is to test the designed control system on real world examples. This addresses the issue of building a complete autonomous reconfigurable controller from a practical point of view. The model used in our work was based on a rover built in our autonomous systems laboratory.

The rover has four independently driven actuators, each controlled through an individual embedded motor controller. It is equipped with four photodiode sensors mounted on each wheel which serves as wheel encoders. A GPS/GSM module is used to calculate the bearing and location of the rover. It can also be used to determine the velocity of the rover. Two sonar sensors are mounted on the front of the AGV to detect and avoid obstacles. It is equipped with a small yet powerful Gumstix motherboards with a Marvell PXA270 600MHz processor running linux. It is a fully functional computer the size of a gumstick. A true multi-agent system can be implemented in these embedded processors where the planner, monitor and control agents all reside on separate hardware. Figure 7.2 shows all the hardware components.



(a) Photodiode, sonar sensor, robostix, gumstix and GPS/GSM module



(b) AGV

FIGURE 7.2: Hardware Components

7.2 Conclusion

The problem of properly integrating the fault detection and diagnosis (FDD) and the control reconfiguration (CR) components of a fault tolerant control system was investigated in this work. The merger of different subsystems should be straight forward but is rarely the case in practice. The difficulty lies in providing instantaneous information for other subsystems despite working perfectly on its own. It was argued in Section 2.5 that an effective integration method of the two components still remains an open issue.

A novel approach using a multiagent framework was proposed to solve this hard problem. This solution offered the the advantage of viewing the reconfigurable control system in a modular manner. It allowed us to break down a system into subcomponents so as to modularize and simplify the design and implementation.

The foundations needed to tackle our problem was presented in Chapter 3. An agent is defined as a tuple $A = \langle \Sigma, \mathcal{X}, \mathcal{K} \rangle$ where $\sigma \in \Sigma$ is the internal architecture, $\chi \in \mathcal{X}$ is the communication set and $\kappa \in \mathcal{K}$ is the resource set. Agents are classified according to their properties. In our work, we dealt exclusively with reactive and learning agents.

A reactive agent's internal architecture is made up of conditional logic rules which maps sensors directly to actions. The internal architecture of a learning agent on the other hand involves implementing machine learning algorithms. Several machine learning algorithms were presented. They serve as the basis for realizing the internal architecture of the agents in our work.

Three types of agents were defined. They were the planner agent A_p, monitor agent A_m and control agent A_c. Section 4.5 described how the agents can cooperate to solve a control problem. A_p is realized as a reactive agent which maps logical conditions to actions. A_m and A_c are learning agents where machine learning algorithms presented in Chapter 3 were used as their internal architecture.

A planner agent A_p supervises the overall control of a task. A control agent A_c learns and executes a controller to complete a task. A multiagent direct policy search algorithm was used as the internal architecture of A_c. We extended the algorithm to allow for continuous states and actions. A monitor agent A_m detects and diagnoses a fault. An ART network enabled A_m to classify faults autonomously.

The methodology of this work was formally represented using the natural language programming software sEnglish in Chapter 5. Sentences were written to define procedures that are needed to realize the multiagent reconfigurable control system.

A case study to demonstrate the the multiagent online reconfigurable controller was presented in Chapter 6. A four wheeled vehicle was modelled along with the sensors used to capture data. The control structure to implement the multiagent framework was defined. Several faults were simulated and the response of the system was discussed. The performance of the proposed architecture compared the controller designed by Kozlowski (2004) was discussed.

We have also discussed some of the limitations of our work in the previous section and presented potential solutions for future research. It is our view that the contribution in this work opens up exciting new opportunities in the field of fault tolerant control systems for control and system engineers of future products.

Appendix A

Publication

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

.pdf

Appendix B

Source Code

Readers are referred to the CD attached.

References

S. Abdelwahed and N. Kandasamy. Fault-adaptive control for robust performance management of computing systems. *Fourth International Conference on Autonomic Computing*, pages 21–21, June 2007.

S. Aberkane, J.C. Ponsart, M. Rodrigues, and Dominique Sauter. Output feedback control of a class of stochastic hybrid systems. *Automatica*, 44(5):1325 – 1332, 2008. ISSN 0005-1098.

F. Allgower, A. Rehm, and E.D. Gilles. An engineering perspective on nonlinear $\mathcal{H}_\infty$ control. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, volume 3, pages 2537 –2542 vol.3, Dec 1994.

G. Bajpai, B. C. Chang, and H. G. Kwatny. Design of fault-tolerant systems for actuator failures in nonlinear systems. In *Proceedings of the 2002 American Control Conference*, volume 5, pages 3618–3623, 2002.

P. Balle, M. Fischer, D. Fussel, O. Nelles, and R. Isermann. Integrated control, diagnosis and reconfiguration of a heat exchanger. *Control Systems, IEEE*, 18(3):52 –63, Jun 1998. ISSN 1066-033X.

B. Banerjee and J. Peng. Adaptive policy gradient in multiagent learning. In *Proceedings Of The Second International Joint Conference On Autonomous Agents And Multiagent Systems*, AAMAS '03, pages 686–692, New York, NY, USA, 2003. ACM. ISBN 1-58113-683-8.

B.R. Barmish and C.M. Lagoa. On convexity of the probabilistic design problem for quadratic stabilizability. *Proceedings of the american control conference*, pages 430–434, 1999.

H. Benítez-Pérez, A. García-Zavala, and F García-Nocetti. *A Proposal for On-Line Reconfiguration Based upon a Modification of Planning Scheduler and Fuzzy Logic Control Law Response.* Springer-Verlag Berlin Heidelberg 2005, 2005.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, August 2006. ISBN 0387310738.

G. Biswas, M.-O. Cordier, J. Lunze, L. Trave-Massuyes, and M. Staroswiecki. Diagnosis of complex systems: Bridging the methodologies of the FDI and DX communities. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(5): 2159–2162, Oct. 2004. ISSN 1083-4419.

M. Blanke, R. Izadi-Zamanabadi, S. A. Bogh, and C. P. Lunau. Fault-tolerant control systems – a holistic view. *Control Engineering Practice*, 5(5):693–702, 1997.

M. Bodson and J.E. Groszkiewicz. Multivariable adaptive algorithms for reconfigurable flight control. *IEEE Transactions on Control Systems Technology*, 5(2):217 –229, mar 1997. ISSN 1063-6536.

H. Bojinov, A. Casal, and H. Hogg. Multiagent control of self-reconfigurable robots. *Artificial Intelligence*, Vol. 142:99, 2002.

F. Bolourchi and R. A. Hess. Nonlinear model reference adaptive control using tap-delay filters. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):360–368, 1992.

W. Borutzky. Bond graph model-based fault detection using residual sinks. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 223(3):337–352, 2009.

J.D. Boskovic, Sai-Ming Li, and R.K. Mehra. On-line failure detection and identification (FDI) and adaptive reconfigurable control (ARC) in aerospace applications. In *Proceedings of the 2001 American Control Conference*, volume 4, pages 2625 –2626 vol.4, 2001.

M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

S. Boyd, L.E. Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities In System And Control Theory*. 1994.

J.P. Briot, S. Aknine, I. Alvarez, Z. Guessoum, J. Malenfant, O. Marin, J.F. Perrot, and P. Sens. Multiagent systems and fault tolerance: State of the art elements. Technical report, LIP6 &MODECO-CReSTIC, Paris, 2007.

S. L. Britain, A. J. Gibb, and C. Roberts. Automatic reconfiguration of a robotic arm using a multi-agent approach. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 222(2):127–135, 2008. 10.1243/09596518JSCE335.

M. Brown and C. J. Harris. *Neurofuzzy Adaptive Modelling And Control*. Prentice Hall, 1994. none.

J.M.F. Calado, J.M.G. S da Costa, M. Bartys, and J. Korbicz. FDI approach to the DAMADICS benchmark problem based on qualitative reasoning coupled with fuzzy neural networks. *Control Engineering Practice*, 14(6):685 – 698, 2006. ISSN 0967-0661.

G. Calafiore and B. Polyak. Fast algorithms for exact and approximate feasibility of robust LMIs. *IEEE transactions on automatic control*, 46:1755–1759, 2001.

C. Candea, H. Hu, L. Iocchi, D. Nardi, and M. Piaggio. Coordination in multi-agent RoboCup teams. *Robotics and Autonomous Systems*, 36(2-3):67–86, 2001.

L. Caracciolo, A. de Luca, and S. Iannitti. Trajectory tracking control of a four-wheel differentially driven mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 2632–2638 vol.4, 1999.

G. A. Carpenter and S. Grossberg. Art-2 - self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, 1987.

G.A. Carpenter, S. Grossberg, and D.B. Rosen. Fuzzy art - an adaptive resonance algorithm for rapid, stable classification of analog patterns. *Ijcnn-91-Seattle : International Joint Conference on Neural Networks, Vols 1 and 2*, pages B411–B416, 1991.

L. Castañon, F. Ortiz, and R. Morales-Menéndez, R.and Ramírez. A fault detection approach based on machine learning models. In *MICAI 2005: Advances in Artificial Intelligence*, pages 583–592, 2005. 10.1007/11579427 59.

I. Chang, C. Yu, and C. Liou. Model-based approach for fault diagnosis. 1. Principles of deep model algorithm. *Industrial and Engineering Chemistry Research*, 33(6):1542–1555, 1994.

B. S. Chen, S. S. Wang, and H. C. Lu. Minimal sensitivity perfect model matching control. *IEEE Transactions on Automatic Control*, 34(12):1279–1283, 1989.

L.H. Chiang, E.L. Russel, and R.D Braatz. *Fault Detection And Diagnosis In Industrial Systems*. Advanced textbooks in control and signal processing. Springer, 2001.

H. J. Chizeck, A. S. Willsky, and D. Castanon. *Markovian jump linear quadratic optimal control in discrete time*, volume 22, pages 1138–1142. 1983.

H.J. Chizeck and A.S. Willsky. *Towards fault-tolerant optimal control*, volume 17, pages 19–20. 1978.

M.-O. Cordier, P. Dague, F. Levy, J. Montmain, M. Staroswiecki, and L. Trave-Massuyes. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(5):2163–2177, Oct. 2004. ISSN 1083-4419.

M.C. Cotting and J.J. Burken. Reconfigurable control design for the full X-33 flight envelope. *NASA/TM-2001-210396*, 2001.

I. Daubechies. *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

P. Dayan and C. Watkins. Reinforcement learning. In *Encyclopedia of Cognitive Science*, volume 2. MacMillan Press, 2001.

U. Demirci and F. Keresteciogl u. A reconfiguring sliding mode controller with adjustable robustness. *Ocean Engineering*, 31(13):1669–1682, 2004.

A.P. Deshpande and S.C. Patwardhan. Online fault diagnosis in nonlinear systems using the multiple operating regime approach. *Industrial & Engineering Chemistry Research*, 47(17):6711–6726, 2008.

Y. Diao and K.M. Passino. Intelligent fault tolerant control using adaptive and learning methods. *Control Engineering Practice*, 10(8):801–817, 2002.

T.G. Dietterich. Hierarchical reinforcement learning with the MAXQ vvalue function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

W.E. Dixon, D.M. Dawson, E. Zergeroglu, and A. Behal. *Nonlinear control of wheeled mobile robots*, volume 262 of *Lecture notes in control and information sciences*. Springer, 2001.

Y. Dote, S.J. Ovaska, and Xiao-Zhi Gao. Fault detection using RBFN- and AR-based general parameter methods. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 77–80 vol.1, 2001.

J.C. Doyle, K. Glover, P.P. Khargonekar, and B.A. Francis. State-space solutions to standard H_2 and H_∞ control problems. *IEEE Transactions on Automatic Control*, 34 (8):pp. 831–847, 1989.

D. Duhaut, E. Carrillo, and S. Saint-Aime. Avoiding deadlock in multi-agent systems. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 1642–1647, Oct. 2007.

Daniel Dvorak, Daniel Dvorak, Benjamin Kuipers, and Benjamin Kuipers. Process monitoring and diagnosis: A model-based approach. *IEEE Expert*, 6:67–74, 1991.

A. El-Fakdi, M. Carreras, and N. Palomeras. Direct policy search reinforcement learning for robot control. In *Proceeding of the 2005 conference on Artificial Intelligence Research and Development*, pages 9–16, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press. ISBN 1-58603-560-6.

J. Farrell, T. Berger, and B. D. Appleby. Using learning techniques to accommodate unanticipated faults. *IEEE Control Systems Magazine*, 13(3):40–49, 1993.

S. Ferrari and Robert F. Stengel. Online adaptive critic flight control. *Journal Of Guidance, Control, And Dynamics*, 27(5):777–786, 2004.

J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer-Verlag New York, Inc., New York, NY, USA, 1996. ISBN 0-387-94805-8.

P.M. Frank. Residual evaluation for fault diagnosis based on adaptive fuzzy thresholds. *IEE Colloquium on Qualitative and Quantitative Modelling Methods for Fault Diagnosis*, pages 4/1–411, Apr 1995.

S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, ECAI '96, pages 21–35, London, UK, 1997. Springer-Verlag. ISBN 3-540-62507-0.

B. Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.

Z. Gao and P.J. Antsaklis. Reconfigurable control system design via perfect model following. *International Journal of Control*, 56(4):783–798, 1992.

S. Gentil, J. Montmain, and C. Combastel. Combining FDI and AI approaches within causal-model-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(5):2207–2221, Oct. 2004.

J. Gertler. *Fault Detection And Diagnosis In Engineering Systems*. Marcel Dekker Inc., New York, 1998. ISBN 1-58603-560-6.

J. Gertler. Residual generation from principal component models for fault diagnosis in linear systems - part I: Extension to optimal residuals and dynamic systems. In *Proceedings of the 2005 IEEE International Symposium onIntelligent Control*, pages 634–639, June 2005a.

J. Gertler. Residual generation from principal component models for fault diagnosis in linear systems - part II: Review of static systems. In *Proceedings of the IEEE International Symposium on Intelligent Control*, pages 628–633, June 2005b.

K. Glover and D. McFarlane. Robust stabilization of normalized coprime factor plant descriptions with $\mathcal{H}_\infty$ bounded uncertainty. *IEEE Transactions on Automatic Control*, 34(8):821–830, 1989.

J.B. Gomm, D. Williams, and P. Harris. Application of on-line parameter estimation techniques to fault detection in controlled processes. *Proceedings of the Singapore International Conference on Intelligent Control and Instrumentation*, 2:1019–1023, Feb 1992.

M. S. Grewal and P. A. Angus. *Kalman Filtering: Theory And Practice Using Matlab*. John Wiley & Sons, Inc, 2001.

M.S. Grewal, L.R. Weill, and A.P. Andrews. *Global Positioning Systems, Inertial Navigation, and Integration (2nd Edition)*. John Wiley & Sons, 2007.

S. Grossberg. Adaptive pattern-classification and universal recoding. 2. feedback, expectation, olfaction, illusions. *Biological Cybernetics*, 23:187–202, 1976.

R. A. Hess and Y. C. Jung. An application of generalized predictive control to rotorcraft terrain-following flight. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5): 955–962, 1989.

M.W. Hofbaur and B.C. Williams. Hybrid estimation of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(5):2178–2191, Oct. 2004. ISSN 1083-4419.

C. Hsieh. Performance gain margins of the two-stage lq reliable control. *Automatica*, 38 (11):1985–1990, 2002.

A. Ichtev, J. Hellendoom, R. Babuska, and S. Mollov. Fault-tolerant model-based predictive control using multiple takagi-sugeno fuzzy models. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, volume 1, pages 346–351, 2002.

R. Isermann. Model-based fault-detection and diagnosis - status and applications. *Annual Reviews in Control*, 29(1):71–85, 2005.

V. Izosimov, P. Pop, P. Eles, and Z. Peng. Design optimization of time- and cost-constrained fault-tolerant distributed embedded systems. In *Proceedings of Design, Automation and Test in Europe*, pages 864–869 Vol. 2, 2005.

C. A. Jacobson and C.N. Nett. An integrated approach to controls and diagnostics using the four parameter controller. *IEEE Control System*, 11(6):22 – 28, 1991.

Y. Jiang, H. Wang, L. Fang, and M. Zhao. Fault detection and identification based on combining logic and model in a wall-climbing robot. *Journal of Control Theory and Applications*, 7(2):157 – 162, 2009. ISSN 16726340.

J. Jin and Y. Zhang. Accepting performance degradation in fault-tolerant control system design. *IEEE Transactions on Control Systems Technology*, 14(2):284–292, 2006.

K. Jongcheol, Y. Suga, and S. Won. A new approach to fuzzy modeling of nonlinear dynamic systems with noise: relevance vector learning mechanism. *IEEE Transactions on Fuzzy Systems*, 14(2):222–231, 2006.

G. Kaiser. *A Friendly Guide To Wavelets*. Birkhaeuser, 1994. ISBN 978-0-8176-8110-4.

N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04*, volume 3, pages 2619 – 2624 Vol.3, april-1 may 2004.

B. Koppen-Seliger, P.M. Frank, and A. Wolff. Residual evaluation for fault detection and isolation with RCE neural networks. *Proceedings of the American Control Conference*, 5:3264–3268, Jun 1995.

D. Kozlowski, K. Pazderski. Modeling and control of a 4-wheel skid-steering mobile robot. *International Journal Of Applied Mathematics And Computer Science*, 14(4): 477–496, 2004.

R.M. Kretchmar. *A synthesis of reinforcement learning and robust control theory.* PhD thesis, Department of computer science, Colorado State University, Fort Collins, Colorado, 2000.

S. Kurnaz, O. Kaynak, and E. Konakouglu. Adaptive neuro-fuzzy inference system based autonomous flight control of unmanned air vehicles. In *Proceedings Of The 4th International Symposium On Neural Networks: Advances In Neural Networks*, ISNN '07, pages 14–21, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72382-0.

H. Kwakernaak. Robustness optimization of linear feedback systems. In *The 22nd IEEE Conference on Decision and Control*, volume 22, pages 618 –624, Dec. 1983.

H. Kwakernaak. Robust control and $\mathcal{H}_\infty$ optimization - tutorial paper. *Automatica*, Vol. 29(No. 2):pp. 255–273, 1993.

K. Lerman, A. Galstyan, and T. Hogg. Mathematical analysis of multi-agent systems. *eprint arXiv:cs/0404002*, April 2004.

F. Liu. Data-based fault detection and isolation (FDI) methods for a nonlinear ship propulsion system. Master's thesis, Simon Fraser University, 2004.

J. Ma and S. Cameron. Robocup 2008: robot soccer world cup XII. pages 532–543. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-02920-2.

J.M. Martin-Sanchez. A new solution to adaptive control. *Proceedings of the IEEE*, 64 (8):1209 – 1218, aug. 1976. ISSN 0018-9219.

J. E. Mason, E. W. Bai, L. C. Fu, M. Bodson, and S. S. Sastry. Analysis of adaptive identifiers in the presence of unmodelled dynamics: Averaging and tuned parameters. In *IEEE Conference on Decision and Control*, volume 26, pages 360–365, 1987.

N. Mehta and P. Tadepalli. Multiagent shared hierarchy reinforcement learning. In *ICML Workshop on Richer Representations in Reinforcement Learning*, 2005.

M.J.G.C. Mendes, B.M.S. Santos, and J. Sa da Costa. Multi-agent platform for fault tolerant control systems. In *IEEE International Conference on Systems, Man and Cybernetics, 2007. ISIC.*, pages 1321 –1326, oct. 2007.

D.D. Moerder, N. Halyo, J.R. Broussard, and A.K. Cahlayan. Application of percomputed control laws in a reconfigurable aircraft flight control system. *Journal of Guidance, Control and Dynamics.*, Vol. 12:pp. 325–333, 1989.

F. Mrad and G. Deeb. Experimental comparative analysis of conventional, fuzzy logic, and adapative fuzzy logic controller. In *Industry Applications Conference, Thirty-Fourth IAS Annual Meeting*, volume vol.1, pages pp.664–673, 1999.

A. Ng and M. Jordan. Pegasus: A policy search method for large mdps and pomdps. In *In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16 (NIPS 2003)*. MIT Press, 2004. ISBN 0-262-20152-6.

H. Noura, D. Theilliol, J.C. Ponsart, and A. Chamseddine. *Fault-tolerant control systems - Design and practical application*. Springer-Verlag, 2009.

Y. Osa, S. Uchikado, and K. Tanaka. Synthesis of nonlinear model matching flight control system for high maneuver vehicle. In *Proceedings of International Conference on Control and Automation, ICCA*, pages 23–27, 2003.

M. Ozbek and D. Soffker. Feature-based fault detection approaches. In *IEEE International Conference on Mechatronics*, pages 342–347, July 2006.

A. Packard and J. Doyle. The complex structured singular value. *Automatica*, 29(1): 71–109, January 1993.

R. Patton. Robustness issues in fault-tolerant control. *IEE Colloquium on Fault Diagnosis and Control System Reconfiguration*, pages 1/1–125, May 1993.

R.J. Patton, J. Chen, and S.B. Nielsen. Model-based methods for fault diagnosis: some guide-lines. *Transactions of the Institute of Measurement and Control*, 17(2):73–83, 1995.

R.J. Patton, P.M. Frank, and R.N. Clarke. *Fault diagnosis in dynamic systems: theory and application*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. ISBN 0-13-308263-6.

R.A Paz and J.V. Medanic. A reliable control design for discrete-time systems. *Proceedings Of The American Control Conference*, pages 190–195, 1991.

M. M. Polycarpou and A. J. Helmicki. Automated fault detection and accommodation: a learning systems approach. *IEEE Transactions on Systems, Man and Cybernetics*, 25(11):1447–1458, 1995.

F. Previdi and T. Parisini. Model-free actuator fault detection using a spectral estimation approach: the case of the damadics benchmark problem. *Control Engineering Practice*, 14(6):635 – 644, 2006. ISSN 0967-0661.

B. Pulido and C.A. Gonzalez. Possible conflicts: a compilation technique for consistency-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(5):2192–2206, Oct. 2004. ISSN 1083-4419.

Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.

W. Qian and R.F. Stengel. Robust nonlinear flight control of a high-performance aircraft. *IEEE Transactions on Control Systems Technology*, 13(1):15–26, 2005.

A. K. Samantaray, S. K. Ghoshal, S. Chakraborty, and A. Mukherjee. Improvements to Single-Fault Isolation Using Estimated Parameters. *SIMULATION*, 81(12):827–845, 2005.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, 1959.

T. Satoh, T. Ishihara, and H. Inooka. Systematic design via the method of inequalities. *Control Systems Magazine, IEEE*, 16(5):57–65, Oct 1996. ISSN 0272-1708.

S.R. Savanur, S.K. Kashyap, and J. R. Raol. Adaptive neuro-fuzzy based control surface fault detection and reconfiguration. In *Proceedings of the International Conference on Aerospace Science and Technology*, Bangaloire, India, 2008.

A.M. Schaefer and S. Udluft. Solving partially observable reinforcement learning problems with recurrent neural networks. In *In Workshop Proceedings of the European Conference on Machine Learning*, 2005.

S. Sen and G. Weiss. *Learning in multiagent systems*, chapter 6, pages 259–298. MIT Press, 1999.

W.M. Shen, B. Salemi, and P. Will. Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots. *IEEE Transactions on Robotics and Automation*, Vol. 18, 2002.

D. Shin, G. Moon, and Y. Kim. Design of reconfigurable flight control system using adaptive sliding mode control: actuator fault. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 219(4):321–328, 2005. 10.1243/095441005X30333.

Y Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, Mar 1993. ISSN 0004-3702.

Y. Shoham and K. Leyton-Brown. *Multiagent systems: algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, New York, USA, December 2008. ISBN 0521899435.

D. Shore and M. Bodson. Flight testing of a reconfigurable control system on an unmanned aircraft. *Proceeding of the American Control Conference Boston. Massachusetts June 30 - July 2*, 2004.

D.D. Siljak. Reliable control using multiple control systems. *International Journal of Control*, 31:303–329, 1980.

S.P. Singh, M.J. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *Proceedings Of The 16th Conference On Uncertainty In Artificial Intelligence*, UAI '00, pages 541–548, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-709-9.

A. Steele and R. Leitch. A strategy for time-constrained qualitative model-based diagnosis. *IEE Colloquium on Qualitative and Quantitative Modelling Methods for Fault Diagnosis*, pages 2/1–2/7, Apr 1995.

R.F. Stengel. Stochastic robustness of linear-time-invariant control systems. *IEEE Transaction on Automatic Control*, 36(1):82–87, 1991.

P. Stone and M. Veloso. Multiagent systems: a survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, June 2000. ISSN 0929-5593.

R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction*. Bradford, Cambridge, 1998.

K. Swingler. *Applying Neural Networks: A Practical Guide*. Academic Press, NY, 1996.

P. Tadepalli and D. Ok. H-learning: A reinforcement learning method to optimize undiscounted average reward. Technical report, 1994.

A. Tahour, H. Abid, and A.G. Aissaoui. Adaptive neuro-fuzzy controller of switched reluctance motor. *Serbian Journal Of Electrical Engineering*, Vol. 4(No. 1):23–34, 2007.

T. Tanaka and A. Weitzenfeld. *Adaptive resonance theory*, chapter 8, pages 157–170. MIT, 2002.

K.K.T. Thanapalan, S.M. Veres, E. Rogers, and S.B. Gabriel. Fault tolerant controller design to ensure operational safety in satellite formation flying. In *IEEE Conference on Decision and Control*, pages 1562–1567, Dec. 2006.

L. Trave-Massuyes, T. Escobet, and X. Olive. Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 36(6):1146–1160, Nov. 2006. ISSN 1083-4427.

F.J Uppala, R.J. Pattona, and M. Witczakb. A neuro-fuzzy multiple-model observer approach to robust fault diagnosis based on the damadics benchmark problem. *Control Engineering Practice*, Vol. 14:699–717, 2006.

V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri. A review of process fault detection and diagnosis: Part ii: Qualitative models and search strategies. *Computers and Chemical Engineering*, 27(3):313 – 326, 2003a.

V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin. A review of process fault detection and diagnosis: Part iii: Process history based methods. *Computers and Chemical Engineering*, 27(3):327 – 346, 2003b.

V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri. A review of process fault detection and diagnosis: Part i: Quantitative model-based methods. *Computers and Chemical Engineering*, 27(3):293 – 311, 2003c.

Sandor M. Veres. *Natural language programming of agents and robotic devices*. SysBrain Ltd, London, 2008.

S.M. Veres. Theoretical foundations of natural language programming and publishing for intelligent agents and robots. In *Proceedings Of TAROS 2010: Towards Autonomous Robotic Systems*, 2010.

S.M. Veres and N.K. Lincoln. Sliding mode control for agents and humans- the use of senglish for publications. In *Proceedings Of TAROS 2008, Towards Autonomous Robotic Systems. Plymouth, UK*, 2008.

S.M. Veres and J. Luo. A class of bdi agent architectures for autonomous control. In *IEEE Conference on Decision and Control*, volume 5, pages 4746–4751, Dec. 2004.

S.M. Veres and L. Molnar. Documents for intelligent agents in english. In *IASTED conference on artificial intelligence and applications. Innsbruck, Austria*, 2010.

M. Vidyasagar. Control systems synthesis - a factorization approach. *MIT Press, Cambridge, MA*, 1985.

M. Vidyasagar, Schneider H., and B.A. Francis. Algebraic and topological aspects of feedback stabilization. *IEEE Trans. Aut. Control*, pages pp. 880–894, 1982.

J.M. Vinson and L.H. Ungar. Dynamic process monitoring and fault diagnosis with qualitative models. *IEEE Transactions on Systems, Man and Cybernetics*, 25(1): 181–189, Jan 1995. ISSN 0018-9472.

R. Wang and J. Wang. Fault-Tolerant Control With Active Fault Diagnosis for Four-Wheel Independently Driven Electric Ground Vehicles. *IEEE Transactions On Vehicular Technology*, 60(9):4276–4287, NOV 2011. ISSN 0018-9545.

C. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, U.K., 1989.

T. Wei, Y. Huang, and C.L.P. Chen. Adaptive sensor fault detection and identification using particle filter algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 39(2):201–213, March 2009. ISSN 1094-6977.

J.F. Whidborne, I. Postlethwaite, and D.-W. Gu. Robust controller design using $\mathcal{H}_\infty$ loop-shaping and the method of inequalities. *IEEE Transactions on Control Systems Technology*, 2(4):455–461, Dec 1994. ISSN 1063-6536.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

K.A. Wise and E. Lavretsky. Adaptive control of flight: Theory, applications, and open problems. *American Control Conference: Minneapolis, Minnesota, USA*, June 14-16 2006.

M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 1st edition, June 2002. ISBN 047149691X.

F. Wu, S. Zilberstein, and X. Chen. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence*, 175(2):487 – 511, 2011. ISSN 0004-3702.

N. E. Wu. Robust feedback design with optimized diagnostic performance. *IEEE Transactions On Automatic Control*, Vol. 42(no.9), 1997.

Z. Yang, M. Blanke, and M. Verhaegen. Robust control mixer method for reconfigurable control design using model matching. *Control Theory & Applications, IET*, 1(1): 349–357, 2007.

Z. Yang and J. Stoustrup. Robust reconfigurable control for parametric and additive faults with fdi uncertainties. In *Proceedings of the 39th IEEE Conference on Decision and Control*, Sydney, Australia, 2000.

A. Yetendje, M. M. Seron, and J. A. De Dona. Robust multisensor fault tolerant model-following mpc design for constrained systems. *International Journal Of Applied Mathematics And Computer Science*, 22(1):211–223, MAR 2012. ISSN 1641-876X.

K. Yin. Minimax methods for fault isolation in the directional residual approach. *Chemical Engineering Science*, 53(16):2921 – 2931, 1998.

W. Yin, W. Zhang, and X. Sun. A svm- based multiple faults classification scheme design in flight control fdi system. In *Second International Conference on Innovative Computing, Information and Control, ICICIC.*, pages 187–187, Sept. 2007.

X. Yu and J. Jiang. Hybrid fault-tolerant flight control system design against partial actuator failures. *IEEE Transactions on Control Systems Technology*, 20(4):871 –886, July 2012. ISSN 1063-6536.

L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965. ISSN 0019-9958.

G. Zames. Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms, and approximate inverses. *IEEE Transactions on Automatic Control*, 26(2):301–320, 1981.

P. Zhang and S.X. Ding. A model-free approach to fault detection of periodic systems. In *Proceedings of the IEEE International Symposium on Intelligent Control. Mediterrean Conference on Control and Automation*, pages 843–848, June 2005.

P. Zhang, S.X. Ding, G.Z. Wang, and D.H. Zhou. An fdi approach for sampled-data systems. In *Proceedings of the American Control Conference.*, volume 4, pages 2702–2707, 2001.

Y. Zhang and J. Jiang. Fault tolerant control system design with explicit consideration of performance degradation. *IEEE Transactions on Aerospace and Electronic Systems*, 39(3):838–848, July 2003. ISSN 0018-9251.

Y. Zhang and J. Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2):229–252, 2008. DOI: 10.1016/j.arcontrol.2008.03.008.

Z. Zhu, Z. Yang, and F. Lei. Multi agent-based dcs with fault diagnosis. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, volume 1, pages 308 – 311, Shanghai, China, 2004.