7

Model Predictive Control in Discrete Manufacturing Shopfloors

Alessandro Brusaferri[1], Giacomo Pallucca[1], Franco A. Cavadini[2], Giuseppe Montalbano[2] and Dario Piga[3]

[1]Consiglio Nazionale delle Ricerche (CNR),
Institute of Industrial Technologies and Automation (STIIMA),
Research Institute, Via Alfonso Corti 12, 20133 Milano, Italy
[2]Synesis, SCARL, Via Cavour 2, 22074 Lomazzo, Italy
[3]Scuola Universitaria Professionale della Svizzera Italiana (SUPSI),
Dalle Molle Institute for Artificial Intelligence (IDSIA),
Galleria 2, Via Cantonale 2C, CH-6928 Manno, Switzerland
E-mail: alessandro.brusaferri@itia.cnr.it; giacomo.pallucca@itia.cnr.it;
franco.cavadini@synesis-consortium.eu;
giuseppe.montalbano@synesis-consortium.eu; dario.piga@supsi.ch

This chapter describes the fundamental components of the Software Development Kit architecture developed in Daedalus and its integration in IEC-61499 paradigm, presenting the methodologies selected to face the issues related to the control of aggregated Cyber Physical System (CPS). The aim of the Software Development Kit is to help automation system engineers to synthesize Hybrid Model Predictive Control for aggregated CPS environment.

The guidelines of future development steps of the tool are described. The SDK is composed of three main parts: On-line System Identification (OIS), Online Control Modeller (OCM) and Online Control Solver (OCS). The first one is dedicated to automatically infer the system's model of aggregated CPS from input and output measurements. OIS absolves two functions: in a preliminary design phase, it is used in order to estimate a first model of the system; successively during execution, it works in real time for tuning the parameter of the system in relation to input and output measurements. The OCM is the main component of SDK and it contains direct interface to modify and customize the parameters of controller to be designed, like observer tuning, prediction horizon and so on. Moreover, the OCM is the synergic element that orchestrate the work flow of OCS, which performs the calculations during execution. The main computational aspects are related to the requirements of the solution of an optimization problem in the receding horizon fashion: in each step, an MIQP problem must be solved in the cycle time: an adequate solver is fundamental to realize Hybrid Model Predictive Control.

7.1 Introduction

Part of the Daedalus project is dedicated to the design and implementation of the Software Development Kit (SDK) that provides helpful tools to develop, implement and deploy advanced control system within a distributed IEC-61499-based control framework, dedicated to automation system engineers.

To such an aim, optimal orchestration of distributed IEC-61499 application is investigated and advanced control techniques as optimal control and model predictive control are considered.

The main features of aggregated Cyber Physical System (CPS) are evaluated to realize an advanced optimal control system: it exhibits, in particular, both continuous and discrete variables to represent the aggregated CPS. Straightforwardly, Hybrid system will be considered, and the various modelling techniques are investigated in Section 7.2.

Another important feature of optimal orchestration of aggregated CPS is the compliance with system constrains on both output variables, i.e. physical limits, and manipulated variables, e.g. actuators saturation and limits. The optimization of a measure of the performance of the system, i.e. the minimization of the cost function, is now a well-established approach in the academia and in certain industries like the chemical and aerospace industries, which have to be widespread in every industrial sector. Therefore, optimization-based control algorithms are investigated for the SDK. Among these, Model Predictive Control stands out as the most promising, considering that Receding Horizon approach offers a way to compensate for disturbances on the system and model mismatch.

Following the last decades of development of control theory, the most suitable solution for above requirements and objectives is Hybrid Model Predictive Control (Section 7.3.1). Indeed, this family of control method guarantees in an implicit manner the respect of constrains and manages multi-objectives control in an optimal way, thanks to Quadratic Programming solver (details will be reported in further sections).

The aim of this chapter is to introduce and carry out an in-depth analysis of the main components of the SDK of Daedalus. Figure 7.1 shows the idea of optimal hybrid orchestrator for aggregated CPS. It is divided into three main subcomponents: Online System Identification tool, Online Control Modeller and Online Control Solver, which are discussed in the following sections.

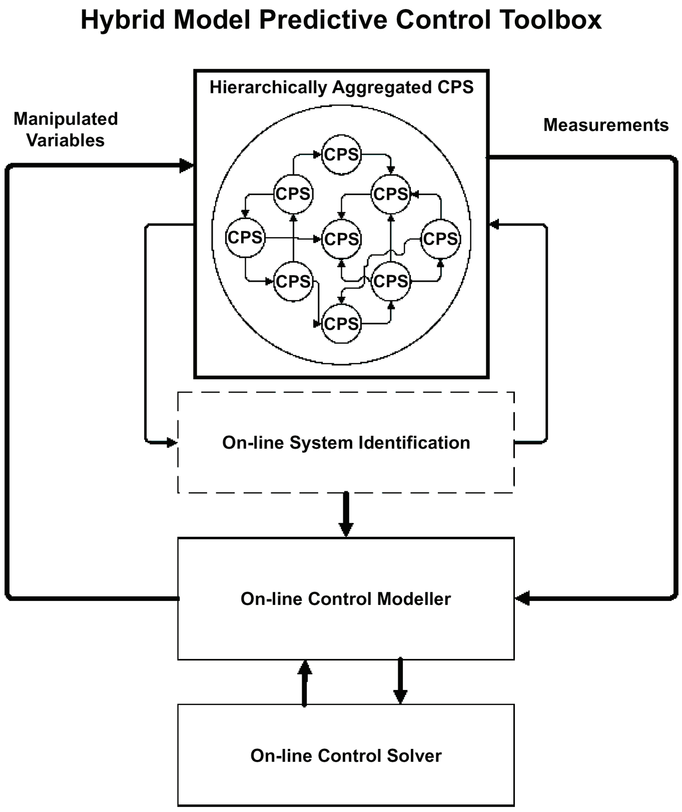

Hybrid Model Predictive Control Toolbox
Hierarchically Aggregated CPS
Manipulated Variables
Measurements
CPS
On-line System Identification
On-line Control Modeller
On-line Control Solver

Figure 7.1 Schematic representation of Hybrid Model Predictive Control Toolbox.

7.1.1 Hybrid Model Predictive Control SDK

The proposed reference framework is composed of three main parts (shown in Figure 7.1). The first one is the On-line System Identification (OIS) tool, which is able to deduce the model of complex Multi-Input Multi-Output (MIMO) hybrid system. This data-driven tool uses input/output variables to extrapolate mathematical model of the system and it is based on iterative real-time procedure, and more details are reported in Section 7.4. The second block is the Online Control Modeller (OCM), where, given a model from the OIS, an optimal predictive controller able to orchestrate the aggregated Cyber-Physical Systems is synthesized. The OCM is developed based on latest paradigm of HMPC, explained in depth in Section 7.3. The last one is Online Control Solver (OCS) that is strictly related to OCM. This solver must be able to deal with Mixed-Integer Quadratic Problem (MIQP), to solve optimal predictive control problem for hierarchically aggregated CPS with quadratic function cost.

To such an aim, the proposed framework is developed to help control engineer to easily create an optimal controller for complex distributed CPS architecture. Each component will be developed with platform-independent software (see Section 7.1.2), which must be flexible and easy to use in order to create a standard procedure that deals with hybrid complex systems. Moreover, the resulting SDK will be integrated in a distributed IEC-61499-based control architecture (see Figure 7.2).

As analysed in Section 7.3.3, the computational aspect cannot be negligible; indeed, Mixed Integer Programming problem requires high computational power to be solved in runtime. This is more critical when complex systems require large controller bandwidth (Hz order): at 1 Hz, the OCS has to solve a Mixed Integer Problem in less than a second. An additional problem is the non-deterministic solving time of MIP. For the robustness of the modelled controller, it is important to evaluate in simulation the worst case of execution time and use a safety factor to evaluate a realistic and safety bandwidth of the controller. To face this problem, virtual commissioning is helpful: it is indeed possible to test control performance and its feasibility in a virtual environment and tune all control parameters.

7.1.2 Requirements

The investigation on orchestration of hierarchically aggregated CPS controller problems had led different needs. The basic development tools, to be compliant with IEC-61499 [1] and to have a platform-independent

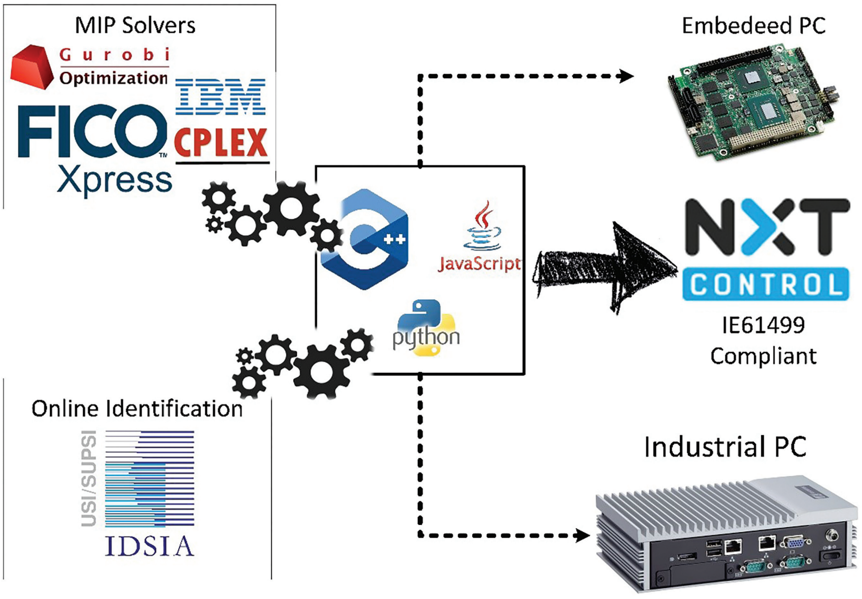


Figure 7.2 Conceptual map of used software. In the centre, there is object-oriented programming language that better supports an easy development and management between different application's needs.

toolbox, seem to be an object-oriented programming language used in cooperation with nxtControl. The nxtControl respects each paradigm of IEC-61499 and allows to build easily distributed control system using function blocks (for more details, see Section 7.5). The possible choice of object-oriented programming language allows to have a wide range of tool easily integrated in a single development environment. Object-oriented programming is easy to use for the purpose of this SDK, and this programming paradigm allows to develop effortlessly scalable and flexible software, independently from the application.

The investigated programming languages are Python, C++ and JavaScript. Even if the natural choice for a direct integration with nxtControl is C++, Python environment allows a better abstraction layer and enables easily the integration of a wide range of tools and libraries developed for optimization solver and control system. Moreover, nxtControl is able to compile Python with a wrapping toolkit, the computational time waste with the wrapper is negligible with respect to the computational time due to Quadratic Problem solver. This aspect conveys that choice of programming languages is not to be restricted to a specified one.

Another important benefit of possible Python's choice is availability of modelling and development environment of MIP solvers, both commercial and free-licence for it. Gurobi [2] and CPLEX [3] are the most powerful and optimized MIP commercial solvers [4], which have dedicated development and modelling environments for Python, also in C++. These environments are easy to configure and more important; they are easily integrable with hierarchically aggregated CPS controller. One limitation of industrial application is the license cost, but the difference of solving time and robustness respect freeware is not negligible. Regarding this, further investigation and benchmark will be done.

First release of the SDK will consider a centralized control scheme, where the on-line system identification tool returns the system's model. Straightforward Online control modeller builds up, based on identified model, a hybrid model predictive controller for the system with desired configuration. Finally, the proceeds controller sets up the online control solver and performs the desired performances respecting the tuning parameters chosen by the user, and moreover managing little modelling mismatching and disturbance on input and measurements.

Figure 7.2 shows the framework of the proposed toolbox. It is possible to see the different MIP solver and the Online Identification toolbox of the SDK; on the right, the different objective platforms where proposed Hybrid Model Predictive Controller will work are shown.

7.1.3 Hybrid System

The behaviour of physical phenomena can be represented by mathematical models. When these models exhibit continuous variable (like differential equation) and discrete/logical variables (like state machine), they are called Hybrid System Models. Every physical phenomenon can be described at different levels of detail; in applied science, it is possible to find various models of the same process, in relation of what the model had to describe. These models should not be too simple or too complicated. To formulate these models, we describe with sufficient level of details the behaviour of the physical phenomena efficiently by computational analysis point of view. In the following sections, the report analyzes the trade-off between simple and computational-light model with respect to more complex and computational-heavy model.

In the last three decades, several computer scientists and control theorists have explored models describing the interaction between continuous dynamics and logical components [5]. Such heterogeneous models

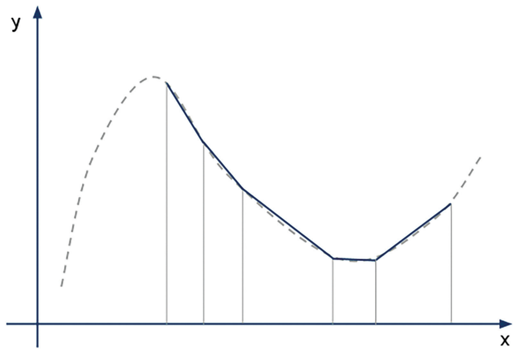


Figure 7.3 Subsequence approximation of a non-linear system.

are denoted as hybrid models; they switch among many operating modes described by differential equation, and mode transitions are triggered by events like states crossing pre-specified thresholds.

Another kind of system that is agreeably represented by hybrid model is non-linear system. Indeed, it is possible to represent non-linear system by a piece-wise linearized model, which consists in a sequence linearization of the system's model around consecutive operating points (see Figure 7.3). This kind of model representation is presented in Section 7.2.1, where its behaviour is also shown. Indeed, the relationship between every working mode is linear, whose slope changes in each region; this is called linearized model of non-linear system and can be represented like a Hybrid system that switches its operating mode.

7.1.4 Model Predictive Control

Model Predictive Control (MPC) arose in the late 1970s and has developed continuously since then. The term MPC does not correspond to specific control strategy, but fairly a wide range of control methods, which use mathematical model of the process to obtain control signal by minimizing an objective function.

Model Predictive Control is an advanced control technique that determinates the control action by solving on-line, at every sampling time k, an open-loop optimal control problem over a p-horizon (Equation (7.2)), based on the current state of the system at k-sample. The optimization generates an input sequence for the specified time horizon p. However, only the first calculated input is applied to the system (Figure 7.4).

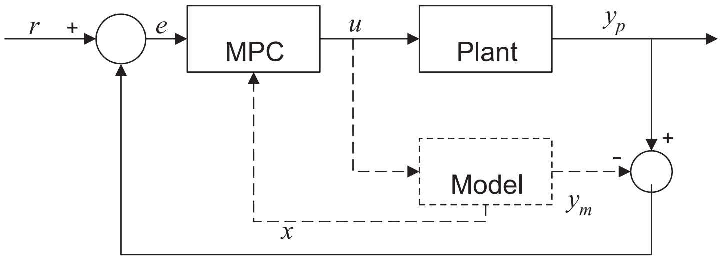


Figure 7.4 Model Predictive Control scheme.

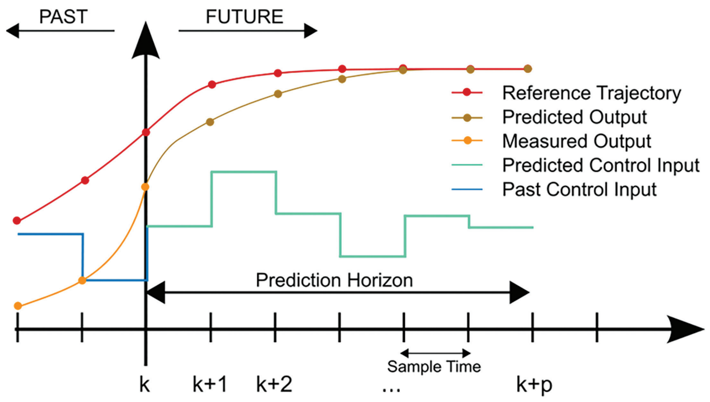


Figure 7.5 Receding horizon scheme.

The ideas at the basis of predictive control methods are:

- Explicit use of model to predict the process output evolution at future time instants (horizon).
- Calculation of control sequence minimizing an objective function.
- Receding strategy. As shown in Figure 7.5, at each sample time, the control computes the optimal sequence of control signal that minimizes the objective function along the horizon, but only the first control signal is applied to the system. This routine is called receding horizon strategy.

There are many successful applications of predictive control in use nowadays from process industry [6] to robots [7] through cement industry, chemical industry [8] or steam generation [9]. The good performance of these applications shows the capacity of the MPC to achieve highly durable and efficient control systems.

Moreover, MPC allows to adjust simultaneously all inputs to control all outputs, while accounting for all process interactions. As a result, MPC can take actions that improve plant performance that a more skilled and experienced operator can achieve.

Moreover, Model Predictive Control is able to consider limitations or constraints of the system, like saturation of actuators and/or physical constraints on output or state variables, directly in the problem formulation. This behaviour is a fundamental improvement that respects classical optimal control (like Linear Quadratic Regulator); in this way, the controller is able to calculate the optimal sequence of control actions that minimize a given cost function, respecting each specified constraint.

The most useful model formulation is the state-space form. This formulation is very helpful in both identification problem and optimal control problem. This modelling environment allows to easily relate inputs, outputs and states variable. In discrete time space for continuous variables, the formulation is (Equation (7.1)):

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases} \tag{7.1}$$

where $x(k)\ \ in\ \ \mathbb{R}^n$ is a vector of the state variables, $u(k)\, in\, \mathbb{R}^m$ are the input variables and $y(k)\ in\ \mathbb{R}^q$ are the output variables. The matrices $A, B, C\ and\ D$ have proper dimensions. In MPC framework, the control goals, such as the tracking of a reference or the satisfaction of constraints, are formulated as a numerical optimization problem. In most cases, this problem is represented as a Quadratic programming (QP) problem. For such an optimization problem, the cost function is the sum of individual terms that express various control requirements. The objective function is generally composed as follows (Equation (7.2)):

$$\begin{aligned} J \triangleq\ & \sum_{i=1}^{P} \|(y(k+i)-y_r)\|_{Q_y}^{N} + \sum_{i=1}^{P} \|(u(k+i)-u_r)\|_{Q_u}^{N} \\ & + \sum_{i=1}^{P} \|(\triangle u(k+i))\|_{Q_{\triangle u}}^{N} \end{aligned} \tag{7.2}$$

where $N=\{1, 2,\infty\}$ represents norm-type that defines the type of minimization problem. A linear problem is defined if $N=\{1,\infty\}$ and quadratic if $N= 2$. P is the prediction horizon that will be considered. $Q_{y,u,\triangle u}$ are positive defined matrices, also called weight matrices of different objectives

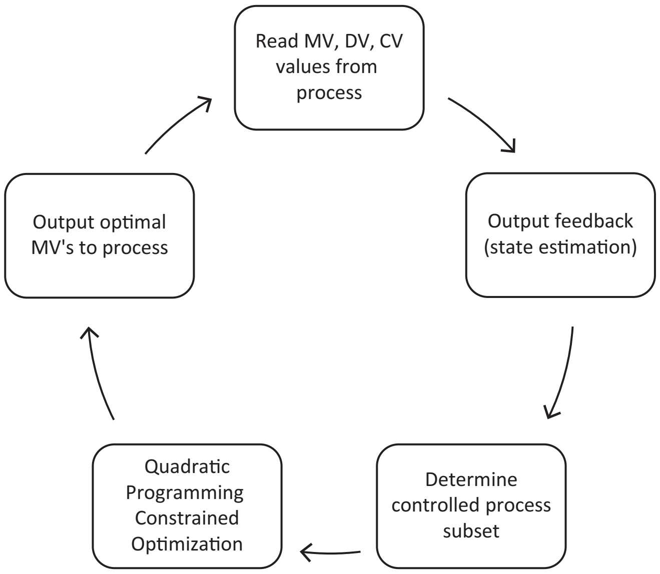

Read MV, DV, CV values from process
Output feedback (state estimation)
Determine controlled process subset
Quadratic Programming Constrained Optimization
Output optimal MV's to process

Figure 7.6 Flow of MPC calculation at each control execution.

of the controller: thanks to these parameters we can tune the controller. For example, if it is not important to control the first output y_1, it is possible to easily set $Q_{y_1} = 0$, and the same action will be applied for other weights.

Overall, the flow of computation for a typical MPC problem is represented in Figure 7.6.

7.2 Hybrid System Representation

During the last decades, Hybrid system arose naturally its interest in the scientific and research community. Many applications of hybrid system modelling in key areas were presented, such as automotive system [10] or power system [11].

A demonstration of considerable interest in hybrid system is the number of periodic conferences and entire session in major conferences completely devoted to them.

Moreover, this research field is relatively open to new advances. New approaches to mathematical representation of hybrid system have just appeared and a growing interest in applications is straightforward.

Hybrid systems are dynamic systems with both continuous states, discrete-states and event-variables. Consequently, a hybrid system provides a perfect structure to represent large plant of industrial process, which can be seen globally like an agglomeration of subsystems working in different modes, switching along the plant operation points. For example, the mathematical car's model with gear shift has different traction force curves related to selected gear [12]. To consider these different dynamics behaviour in a unique model, hybrid system modelling is mandatory. Moreover, hierarchical systems can be modelled as hybrid, in which lower components are described by continuous variables and higher-level blocks are governed by logic or decision modules.

Different kinds of models can be used to describe hybrid system. For control purpose, hybrid modelling techniques have to be descriptive enough to capture the behaviour of the interconnections between logic components (automata, switches, software code) and continuous dynamics (physical laws). Simultaneously, the model must to be simple enough to solve analysis and synthesis problems.

The state of the art of hybrid system modelling can be summarized in two main groups (Figure 7.7): the more used piecewise affine (PWA) system [13], mixed logical and dynamical (MLD) models [14] and hybrid automata (HA) [15]; and less used linear complementarity (LC), extended

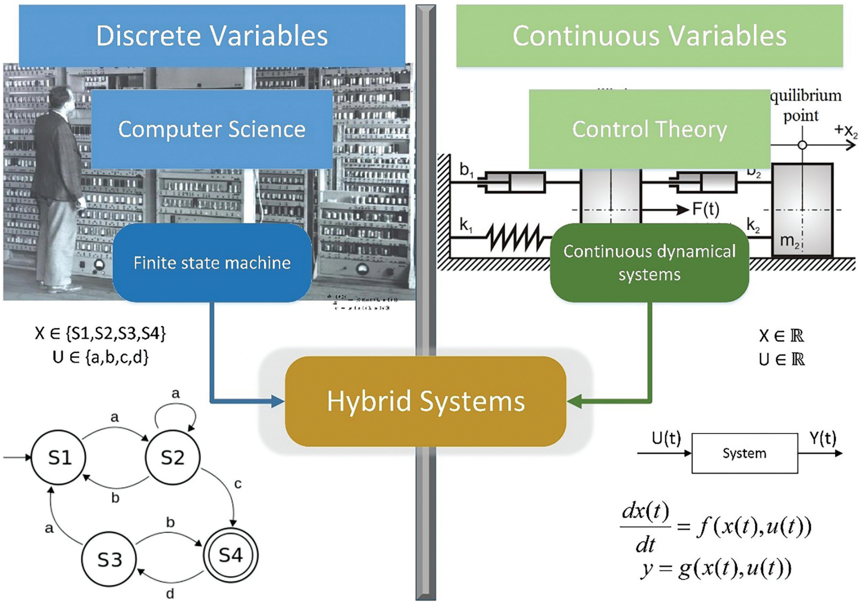


Figure 7.7 Schematic representation of hybrid system.

linear complementary (ELC) system and max-min-plus-scaling (MMPS) systems [16].

In detail, as proved in [16], all those modelling frameworks are equivalent and it is possible to describe the same system with models of each class. This characteristic is useful, for example, as each formulation offers some advantages in one particular situation: MLD framework is the best for the optimization of the system, while stability and robustness are more easily proved in a PWA formulation.

Hybrid system modelling allows to describe a variety of different kinds of systems, for example, it is possible to deal with complex system like switched dynamics system. Moreover, a hybrid model can describe the complete dynamics of the system and consider different aspects of the same system that works in different ways. For example, when a robot works in a cooperative environment, this type of modelling technique is able to consider each different dynamic, like free motion, contact with operator, different payloads applied at end-effector, etc.

Another kind of system that can be modelled as hybrid system is non-linear system. A common method to face non-linear system consists of piece-wise linearization around consecutive operating points. The output of this procedure is a PWA model (see Equation (7.3)).

The main advantage of using this kind of modelling system to synthesise a Model Predictive Control (MPC) is that the controller, when is calculating predicted outputs, is able to consider each different dynamics included in the model and optimize the control action in order to minimize the functional cost (i.e. minimize energy consumption, control action magnitude or tracking error).

7.2.1 Piece-Wise Affine (PWA) System

PWA systems representation is the most studied form of hybrid systems. A PWA system is defined as (Equation (7.3)):

$$\begin{cases} x(t+1)=A^i x(t)+B^i u(t)+f^i \\ y(t)=C^i x(t)+g^i \end{cases} \quad for\ [x(t),u(t)] \in \chi_i \tag{7.3}$$

where $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$ and $y(t) \in \mathbb{R}^r$ denote the state and the input and output vectors. $\{\chi_i\}_{i=1}^{s}$ is a convex polyhedral partition of the states and input space (i.e. see Figure 7.8). Each χ_i is given by a finite number of linear inequalities.

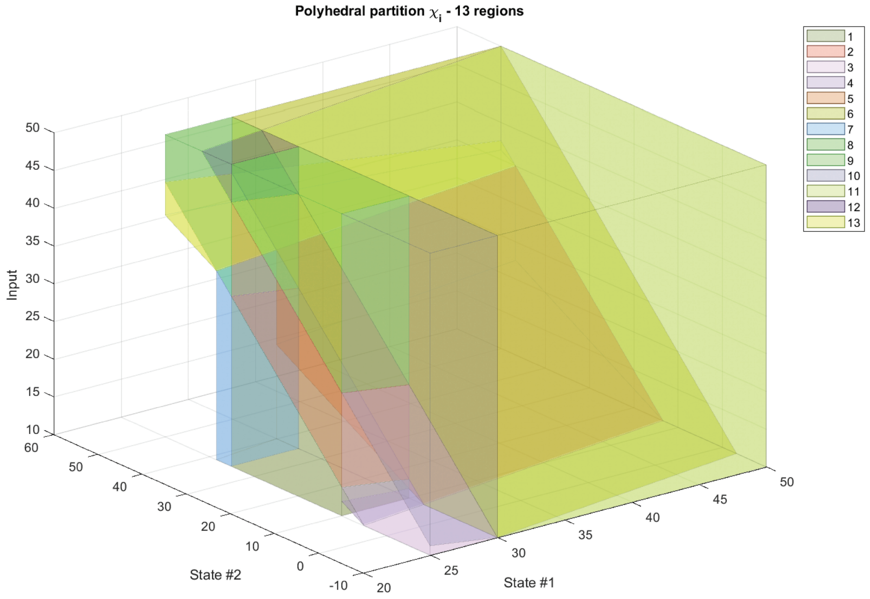


Figure 7.8 Polyhedral partition representation of a hybrid model. It is possible to see 13 partitions that divide the input state space into 13 pieces-wise sub-systems (using MatLab 2017b).

7.2.2 Mixed Logical Dynamical (MLD) System

In ref. [14], a new type of hybrid systems representation has been defined, in which logic, dynamics and constraints are integrated.

The MLD description is (Equation (7.4)):

$$\begin{cases} x(k{+}1){=}Ax(k){+}B_1u(k){+}B_2\delta(k){+}B_3z(k) \\ y(k){=}Cx(k){+}D_1u(k){+}D_2\delta(k){+}D_3z(k) \\ E_5 \;\geq E_1x(k){+}E_2u(k){+}E_3\delta(k){+}E_4z(k) \end{cases} \tag{7.4}$$

where $x(k) = [x_r^T(k), x_b^T(k)]$ with $x_r(k) \in \mathbb{R}^{n_r}$ and $x_b(k) \in \{0,1\}^{n_b}$; $y(k) = [y_r^T(k), y_b^T(k)]$ with $y_r(k) \in \mathbb{R}^{m_r}$ and $y_b(k) \in \{0,1\}^{m_b}$; $u(k) = [u_r^T(k), u_b^T(k)]$ with $u_r(k) \in \mathbb{R}^{q_r}$ and $u_b(k) \in \{0,1\}^{q_b}$. $z(k) \in \mathbb{R}^{r_r}$ and $\delta(k) \in \{0,1\}^{r_b}$are auxiliary variables that are used to represent the switching between different operating modes.

The inequalities have to be interpreted component-wise, and they define the switching conditions of different operating modes. The construction of this inequality is based on tools able to convert logical facts involving continuous variables into linear inequalities (for more details, see [17]). This tool will be used to express relations describing the evolution of systems where physical laws, logic rules and operating constrains are interdependent.

Equation (7.4) commits linear discrete-time dynamics for the first two equations. It is possible to build up another formulation describing continuous time version by substituting $x(k + 1)$ by $x(t)$ or a non-linear version by changing the linear equation and inequalities in (7.4) to more non-linear functions. However, in this way, the problem becomes hard tractable by a computational point of view, and more in general, the MLD representation allows to describe a wide range class of systems.

MLD models are successful thanks to good performance in computation aspect. The main claim of their introduction was the easy handling of non-trivial problems, for the formulation of Model Predictive Control for hybrid and non-linear system. This formulation performs well when it is used together with modern Mixed-Integer Programming (MIP) solver for synthesizing predictive controller for hybrid systems, as described in Section 7.4.1.

Note that the class of Mixed Logical Dynamical systems includes the following important system classes:

- Linear systems;
- Finite state machines;
- Automata;
- Constrained linear systems;
- Non-linear dynamic systems.

In fact, the next section introduces the equivalence between different hybrid system representations and it underlines the potential of MLD models (in Figure 7.9, it is possible to see the interconnection between MLD and other system representation models).

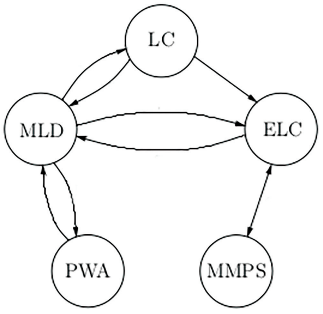


Figure 7.9 Graphic scheme of the links between the different classes of hybrid. The arrow from A to B classes shows that A is a subset of B.

7.2.3 Equivalence of Hybrid Dynamical Models

In ref. [16], there are different demonstrations of equivalence between each hybrid system model, summarized in Figure 7.9. For some transformations, additional conditions like boundedness of the state and input variables or well-posedness have to be made. Typically, the more frequent condition is that the polyhedral partition of input-state space must be univocally defined, i.e. with no overlapping between different χ_i. These requirements are fundamental in that case where, for example, in PWA or MLD, the modelling framework does not allow overlapping of sub-set of state-input space.

These equivalences are fundamental to demonstrate the properties of different hybrid models and commonly use stability analysis on a single representation, translating its effects on another modelling system.

7.3 Hybrid Model Predictive Control

Dealing with control of hybrid systems is an open field of research in both academia and industrial world. Model predictive control based its main advantage on the prediction of future outputs, which requires a model that considers the evolution of the system. In case of hybrid systems, discrete variables must be included. For this aim, the modelling frameworks described in Section 7.2 have to be considered.

7.3.1 State of the Art

Model predictive control was proposed for the first time in the late 1970s by Richalet et al. [9], who predicted future outputs in a heuristic manner. During that time, the application field of MPC was process industry, from chemical to oil and gas extraction through pharmaceutical industries.

Since then, model predictive control has been extended to a wide range of control problems. During the 1990s [18], the academics world was interested on stability analysis, because it is a very challenging problem not only for control engineers but also for mathematicians. Control engineers moved their focus to large systems, where both continuous and discrete variables describe the model of the system, therefore requiring a hybrid model predictive control solution [14]. HMPC consists in a repetitive solution of a Mixed-Integer Programming (MIP) problems, where variables could be both continuous and discrete. If the objective function is quadratic, these problems are classified as Mixed Integer Quadratic programming (MIQP) or Mixed Integer Linear programming (MILP), if a linear objective function is used.

MILP and MIQP problems are much more difficult to solve than a linear or quadratic programming problem (LP or QP), and some properties like convexity are lost (see ref. [19] for a more detailed description).

The computational load for solving an MIP problem is a key issue, as a brutal force approach consists of the evaluation of every possible combination. The optimal solution would be to solve every QP or LP related to all the feasible combinations of discrete decision variables. The solution is the minimum of all the computed solution of QP/LP problems. For example, if all the discrete decision variables are Boolean, then the number of possible LP/QP problems is 2^(n_b). Fortunately, there exists an entire research field on this topic and nowadays, there is a wide range of commercial solvers able to deal with MIP problem in a very fast way. These software are mainly based on branch and bound methods [20]; the most known and used are CPLEX (ILOG Inc. [3]), GLPK (Makhorin [21]) or GUROBI [2] for which APIs for many programming languages are available.

The application of the Model Predictive Control arose in the early 1990s. One of the first fundamental studies was made by Bemporad and Morari [14]: they proposed a rigorous approach to mathematical modelling of hybrid system where it is possible to obtain a compact representation of system called Mixed Logical Dynamical (MLD, see Section 7.3.2). Then, following the optimization step, it is possible to synthesize an optimal constrained receding horizon control. This methodology is helpful to optimize and orchestrate both large systems with mixed-variables and non-linear systems linearized around sequential operating points.

As in birth of MPC, the first implementation was in the field of refinery and chemical process. In these fields, Model Predictive Control was already a standard, and the possibility to build up a unique mathematical model that represents the whole system, like plant with all its components, and synthesize a unique controller able to find the optimal solution that respects every specified constrain was a revolution. In the next section, we deeply explore the issues and limits of Hybrid Model Predictive Control, which are roughly synthesizable in computational time and computational power. In that period, the solution of this problem was overcome by using off-line optimization, also called Explicit MPC. This control method is able to properly work only in a predetermined range of variable states: in fact, the on-line optimization was replaced by an off-line optimization, summarized in a lookup table. Using this methodology, the application of Hybrid MPC could be extended to mechanical and mechatronics system, where the cycle time can be very small. Some applications are summarized in refs. [10–12].

Indeed, in refinery and chemical process or more generally in process industry, the sampling time of the controller is in minutes-order. Since the solution of Mixed-Integer Programming problem is feasible, in these fields, it is used as industrial standard. However, in the last two decades, from ref. [14], the computational power of embedded micro-processor or Industrial PC has grown exponentially, as Moore's law said, and the commercial MIP solvers increase their "power" dramatically. These evolutions allow to rethink to Hybrid MPC with on-line optimization applied to fast system, with sampling time in the range of a few seconds. The aim of this study is to build a standard method to synthesize Model Predictive Control for hybrid system (aggregated CPS too) and have the opportunity to test a possible on-line execution of the controller, in order to understand the minimum sampling time of the controller. This possibility is a killer-feature in refinery and chemical process where Hybrid MPC already is in use, but there is not a powerful and standard tool able to help control's engineers to design HMPC for process industry. Otherwise, in the mechanical and mechatronics system control field, this tool can be revolutionary because it simplifies the design of the controller and standardizes it: in this way, the focus to realize a feasible controller is moved on MIP solving time. In addition, the designer can check in a meticulous, but fast, way the feasibility of the Hybrid Model Predictive Control and its performance.

7.3.2 Key Factors

In the last decades, since the introduction of MPC in control theory, a wide variety of application has been presented. All these applications are related to notable capabilities of fitting the control goals. Indeed, this methodology is able to realize very smooth and precise control. Moreover, MPC is capable of being tuned in a straightforward way in relation to desired performance of the system. As described in Section 7.2, a typical function cost contains different weights, which offer the possibility to tune the performance of the controller, easily to tune also for non-technical people. Moreover, the definition of constrains is direct in the optimization problem and it is simple to impose constraints on Manipulated Variables (MVs) and Output Variables (OVs), which means limits on actuator saturation, dynamical constrain on actuators and physic limits of the controlled system.

Summarizing the benefit of Model Predictive Control:

- Most widely used control algorithm in material and chemical processing industries [22];

- Increased consistency of discharge quality. Reduced off-specs products during grade changeover. Increased throughput. Minimizing the operating cost while meeting constrains (optimization, economic) [23];
- Superior for process with a large number of manipulated and controlled variables (multivariable, strong coupling) [24];
- Allows constraints to be imposed on both MVs and CVs. The ability to operate closer to constraints and over those (soft constraints);
- Allow time delays, inverse response, inherent non-linearities (difficult dynamics), changing control objectives and sensor failure (predictive);
- Optimal rejection to modelling error and disturbances;
- Multi-objectives control technique [25].

7.3.3 Key Issues

The basic issue of Hybrid MPC, and MPC in general, is related to the computational time needed to solve in real time the optimization problem. Indeed, when dealing with a large and fast system, the model of the system becomes really complex and the required closed loop time very precise and the online optimization is not achievable. In order to minimize the problem caused by large system, a pre-stored control allocation law can be used to avoid increased number of decision variables and increased solving time. This technique is known as Explicit Model Predictive Control [26], where the controller creates a look-up table during off-line simulation and uses it during the execution time. This method is able to avoid the main drawback of MPC removing the optimization procedure that is very time-consuming. This benefit enables the use of MPC, and mainly Hybrid MPC, inside application with very high sampling rates.

Another important issue is the difficulty to demonstrate the robustness of the control respect to the classical robust control technique like $H_{-\infty}$ [27]. A possible solution of this issue is to couple with the MPC controller an Online system identification tool, as it is shown in Errore. L'origine riferimento non è stata trovata., that is able to realize a more robust control. This is because the online system identification checks and tunes the system model recursively, compensating modellation errors.

7.4 Identification of Hybrid Systems

The design of a hybrid model predictive controller needs to describe the plant dynamics in terms of a hybrid linear model, which is used to simulate the plant behaviour within the prediction horizon. As known, there are basically two ways to construct a mathematical model of the plant:

- Analytic approach, where models are derived from first-principle physics laws (like Newton's laws, Kirchhoff's laws, balance equations). This approach requires an in-depth knowledge and physical insight into the plant, and in the case of complex plants, it may lead to non-linear mathematical models, which cannot be easily expressed, converted or approximated in terms of hybrid linear models;
- System identification approach, where models are derived and validated based on a set of data gathered from experiments. Unlike the analytic approach, the model constructed through system identification has a limited validity (e.g., it is valid only at certain operating conditions and for certain types of inputs) and it does not give physical insights into the system (i.e., the estimated model parameters may have no physical meaning). Nevertheless, system identification does not need, in principle, in-depth physical knowledge of the process, thus reducing the modelling efforts.

In this project, hybrid linear models of the process of interested will be derived via system identification, and physical insights into and knowledge of the plant will be used, if needed, to assist the whole identification phase, such as choosing the appropriate inputs to perform experiments, choosing the structure of the hybrid model (defined, for instance, in terms of number of discrete states and dynamical order of the linear subsystems), debugging the identification algorithms and assessing quality of the estimated model.

The following two classes of hybrid linear models will be considered, which mainly differ in the assumption behind the switches among the (linear/affine) time-invariant sub-models:

- Jump Affine (JA) models, where the discrete-state switches depend on an external signal, which does not necessarily depend on the value of the continuous state. The switches among the discrete states can be governed, for instance, by a Markov chain, and thus described in terms of state transition probabilities. Alternatively, in deterministic jump models, the mode switches are not described by a stochastic process, but they are triggered by or associated to determinist events (e.g. gear or speed selectors, evolutions dependent on if-then-else rules, on/off switches and valves). In this chapter, we will focus on the identification of deterministic jump models. Stochastic models might be considered at a later stage, only if necessary.

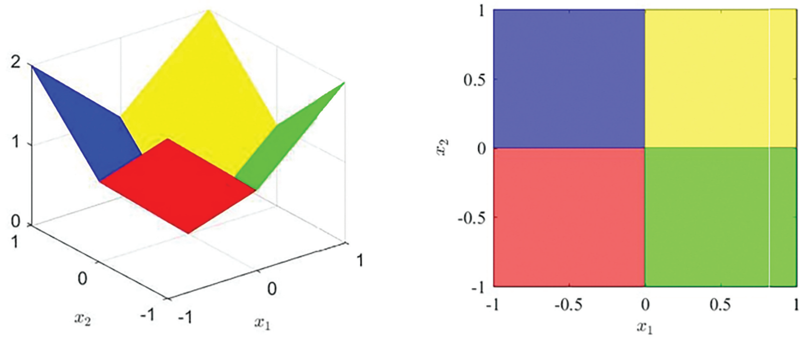


Figure 7.10 Example of a three-dimensional PWA function $y = f(x_1, x_2)$.

- Piece-Wise Affine (PWA) models, where the active dynamic affine sub-model at each time instant only depends on the value of the continuous state. More specifically, in PWA models, the (continuous) state space is partitioned into a finite number of polyhedral regions with non-overlapping interiors, and only one dynamical affine model is associated to each polyhedron. PWA models can be used to accurately describe dynamical systems that evolve according to different dynamics depending on the specific point in the state-input space (e.g. a bouncing ball or switching feedback control laws where the switches between the controllers depend on the state of the system). Furthermore, thanks to the universal approximation property of PWA maps, PWA models can be also used to approximate non-linear/non-smooth phenomena with an arbitrary degree of precision [28]. For the sake of visualization, an example of a three-dimensional PWA function, defined over four polyhedral regions of the state space, is plotted in Figure 7.10.

Note that Jump models and PWA models can be also combined to describe, for instance, finite state machines (with linear dynamics at each mode), where the mode transition depends on both an external event and the current value of the continuous state, input and output.

In the following, we formalize the hybrid system identification problem and discuss its main challenges. Finally, we provide an overview of the algorithm that will be used and implemented in the DAEDALUS platform, for the identification of both Jump Affine and PWA models.

7.4.1 Problem Setting

Let us consider a training dataset of input/output pairs $\mathcal{D} = \{u(t), y(t)\}_{t=1}^{N}$ (generated by the plant we would like to model), where t denotes the time index, $u(t) \in Rnu$ and $y(t) \in Rny$ are the input and output of the system at time t, respectively, and N is the length of the training set. Our goal is to estimate, from the considered training set D, a hybrid linear dynamical model approximating the input/output relation of the system and described in the input/output Auto-Regressive with Exogenous input (ARX) form (Equation (7.5)):

$$\hat{y}(t)= \Theta_{s(t)}x(t) \tag{7.5}$$

where $\hat{y}(t) \in Rny$ is the output of the estimated model, $s(t) \in \{1, \ldots, \overline{s}\}$ is the active mode at time t (i.e. the value of the discrete state at time t) and $x(t) \in X \subset Rnx$ is the regressor vector containing past values of the input and of the output (Equation (7.6)), i.e.

$$x(t) = [1\; y(t-1)' \ldots y(t-n_a)'u(t)'u(t-1)'u(t-n_b)']' \tag{7.6}$$

for some fixed values of n_a and n_b, and $\Theta_s \in Rny, nx$ (with $s = 1, \ldots, \overline{s}$) is the parameter matrix describing the linear sub-model associated to the discrete state s.

The identification of a hybrid linear dynamical model (Equation (7.5)) thus requires: (i) choosing the number $\overline{s}$ of modes (i.e. size of the discrete state); (ii) computing the parameter matrices Θ_s (with $s = 1, \ldots, \overline{s}$) characterizing the affine sub-models; (iii) finding the hidden sequence of discrete states $\{s(t)_{t=1}^{N}\}$ and (iv) in the case of PWA model identification, finding the polyhedral partition of the regressor space X where the affine sub-models are defined.

When choosing the dimension $\overline{s}$ of the discrete state, one must take into account the trade-off between data fitting and model complexity. For small values of $\overline{s}$, the hybrid model cannot accurately capture the non-linear and time-varying dynamics of the system. On the other hand, increasing the number of modes also increases the degrees of freedom in the description of model, which may cause overfitting and poor generalization to unseen data (i.e., the final estimate is sensitive to the noise corrupting the observations), besides increasing the complexity of the estimation procedure and of the resulting model. In the identification algorithms, which will be developed during the project, we will assume that $\overline{s}$ is fixed by the user. The value of $\overline{s}$ (as well as the values of the parameters na and nb defining the dynamical order of the affine sub-models) will be chosen through cross-validation, with a possible upper-bound dictated by the maximum tolerable complexity of the estimated model or by some physical insight into the system.

Fitting-Error Minimization

The hybrid linear model structure in Equation (7.5) suggests to formulate the identification of the hybrid models as the following fitting-error minimization problem

$$\min_{\substack{\{\Theta_s\}_{s=1}^{\overline{s}} \\ \{s(t)\}_{t=1}^{N}}} \frac{1}{N} \sum_{t=1}^{N} \|y(t) - \Theta_{s(t)} x(t)\|_2^2 \tag{7.7}$$

which aims at minimizing, over the parameter matrices Θ_s (with $s = 1, \ldots, \overline{s}$) and the discrete state sequence $\{s(t)\}_{t=1}^{N}$, the power of the error between the measured output $y(t)$ and the model output $\hat{y}(t){=}\ \Theta_{s(t)}x(t)$.

In the cases where the discrete state sequence $\{s(t)\}_{t=1}^{N}$ is exactly known (e.g. when $s(t)$ is associated to the gear number in a car or to an external switching signal controlled by the user, or, for PWA models, the partition of the regressor space X is fixed a priori), the fitting-error minimization problem (7.7) becomes a simple linear regression problem, and the parameter matrices Θ_s (with $s = 1, \ldots, \overline{s}$) defining the affine sub-models can be easily estimated through standard least squares, i.e.

$$\hat{\Theta}_s = arg \min_{\Theta_s} \frac{1}{N} \sum_{t=1}^{N} \mathbb{I}\{s = s(t)\} \|y(t) - \Theta_{s(t)} x(t)\|_2^2 \tag{7.8}$$

with $I\{s{=}s(t)\}$ denoting the indicator function, i.e.

$$I\{s{=}s(t)\}{=} \begin{cases} 1 & if\ s{=}s(t) \\ 0 & otherwise \end{cases} \tag{7.9}$$

Namely, in computing an estimate of Θ_s through Equation (7.8), only the regressor/output pairs $(x(t), y(t))$ such that $s{=}s(t)$ are considered.

In the more general case, where the discrete state sequence $\{s(t)\}_{t=1}^{N}$ is not available, the identification of hybrid models becomes NP hard (strictly speaking, Equation (7.8) is a mixed-integer quadratic programming problem, which might be computationally intractable, except for small-scale problems). Furthermore, besides reconstructing the discrete1 state sequence $\{s(t)\}_{t=1}^{N}$ and estimating the parameter matrices Θ_s (with $s = 1, \ldots, \overline{s}$), the identification of PWA models also requires to compute a polyhedral partition of the regressor space X.

7.4.2 State-of-the-Art Analysis

Several heuristics have been proposed in the literature to overcome the challenges encountered in hybrid system identification (see [29, 30] for an exhaustive overview of algorithms for identification of Jump Affine and PWA models). Among the proposed algorithms, we have analyzed:

- the bounded-error approach [31], which addresses the identification of Jump Affine models under the assumption that the noise corrupting the output observations $y(t)$ is norm-bounded (with known bound). The goal is to estimate the set of all model parameters Θ_s, which are compatible with the a-priori assumptions on the noise bound, the chosen model structure and the observations. A polynomial optimization problem is formulated, whose solution is approximated through convex-relation techniques based on the theory of moments [32]. This approach turns out to be very sensitive to outliers (i.e. noise outside the supposed bounds) and conservative if a large bound on the noise is assumed. Furthermore, it suffers from high computational complexity because of the high computational burden of the employed theory-of-moment-based relaxation;
- the sparse optimization-based approaches [33] and [34], which address the segmentation of linear models by formulating an optimization problem penalizing the fitting error and the number of switches among the affine sub-models. Therefore, these methods are suited only for Jump Affine systems with infrequent switches;
- the mixed-integer quadratic programming approach [35], which addresses the identification of PWA systems using hinging-hyperplane ARX models and piecewise affine Wiener models. A mixed-integer quadratic programming problem is formulated (similar, but not exactly equal to (6.3)) and solved through brunch-and-bound. Unfortunately, the number of integer variables increases with the number of training samples, limiting the applicability of the method to small-/medium-scale problems;
- the two-stage clustering based approach [36], which can be used for both Jump Affine and PWA model identification. At the first stage, the regressor observations are clustered by assigning each data-point to a sub-model through a k-means-like algorithm, and the affine sub-model parameters Θ_s are estimated at the same time. In the case of PWA identification, a second stage is performed to compute a partition of the

regressor space X. Although ref. [36] is able to handle large training sets, poor results might be obtained when the affine local sub-models are over-parameterized (i.e. large values of the parameters na and nb in the definition of the regressor (6.2) are used), since the distances in the regressor space (namely, the only criterion used for clustering) turns out to be corrupted by redundant, thus irrelevant, information;

- the recursive two-stage clustering-based approach [37], which is based on the same two-stage clustering philosophy of [36], is suited for both Jump Affine and PWA model identification. The proposed approach consists of two stages: (S1) simultaneous clustering of the regressor vector and estimation of the model parameters Θ_s ($s = 1, \ldots, \overline{s}$). This step is performed recursively by processing the training regressor/output pairs sequentially; (S2) computation of a polyhedral partition of the regressor space through efficient multi-class linear separation methods. This step is performed either in a batch way (i.e. offline) or recursively (i.e. online). Note that stage S2 is required only for PWA system identification. Because of its computational efficiency and the possibility to be used both for batch and recursive identification, we have decided to use and implement this algorithm in the DAEDALUS project. Further details on this algorithm are discussed below.

7.4.3 Recursive Two-Stage Clustering Approach

The main ideas behind the recursive two-stage clustering approach proposed in ref. [37] are presented in this section. As mentioned in the previous paragraph, the hybrid system identification problem is tackled in two stages: S1 (iterative clustering and parameter estimation) and S2 (polyhedral partition of the regressor space, necessary only for PWA model estimate).

Stage S1 is carried out as described in Algorithm 1, where clusters and sub-model parameters are updated iteratively, making the algorithm suitable for online applications, when data are acquired in real time.

Algorithm 1 Recursive clustering and parameter estimation

Input: Observations $\{x(t), y(t)\}_{t=1}^{N}$, desired number $\overline{s}$ of affine submodels, initial condition for model parameter matrices $\Theta_1, \ldots, \Theta_{\overline{s}}$.

```
1. let $\mathcal{C}_s \leftarrow \emptyset$, $s = 1, \ldots, \overline{s}$;
```

```
2. **for** $t = 1, \dots, N$ **do**
    2.1. **let** $e_s(t) \leftarrow y(t) - \Theta_s x(t)$,
    2.2. **let** $s(t) \leftarrow \arg\min_{s=1,\dots,\overline{s}} \|e_s(t)\|_2^2$;
    2.3. **let** $\mathcal{C}_{s(t)} \leftarrow \mathcal{C}_{s(t)} \cup x(t)$;
    2.4. **update** $\Theta_{s(t)}$ using recursive least-squares;
3. **end for**;
4. **end.**
```

Output: Estimated matrices $\Theta_1, \dots, \Theta_{\overline{s}}$, clusters $\mathcal{C}_1, \dots, \mathcal{C}_{\overline{s}}$, sequence of active modes $\{s(t)\}_{t=1}^{N}$.

The main idea of Algorithm 1 is to compute, at each time instant t, the fitting error $es(t) = y(t) - \Theta_s x(t)(s \in \{1, \dots, \overline{s}\})$ achieved by all the $\overline{s}$ local affine sub-models, and select the local model that "best fits" the current output observation $y(t)$ (Steps 2.1 and 2.2). The regressor $x(t)$ is then assigned to the cluster $C_s(t)$ (Step 2.3) and the parameter matrix $\Theta_s(t)$ associated to the selected submodel is updated using recursive least squares (Step 2.4).

Due to the greedy nature of Algorithm 1, the estimates of the model parameters Θ_s and the clusters Cs are influenced by the initial choice of the parameters Θ_s. A possible initialization for the parameter matrices is to take $\Theta_1, \dots, \Theta_{\overline{s}}$ all equal to the best linear model, i.e.

$$\Theta_s = \arg\min_{\Theta} \frac{1}{N} \sum_{t=1}^{N} \|y(t) - \Theta x(t)\|_2^2, \quad s = 1, \dots, \overline{s}.$$

Moreover, the estimation quality can be improved by reiterating Algorithm 1 multiple times, using its output as an initial condition for the following iteration. This can be performed only if the algorithm is executed in a batch mode (offline). Alternatively, a subset of data can be processed in a batch mode to find proper initial conditions. Then, Algorithm 1 is executed in real time to iteratively process data streaming.

7.4.4 Computation of the State Partition

If a PWA identification problem is addressed, besides estimating the model parameters $\{\Theta_s\}_{s=1}^{\overline{s}}$ and the sequence of active modes $\{s(t)\}_{t=1}^{N}$, also a polyhedral partition of the regressor space X should be found. More specifically, let X_s (with $s = 1, \dots, \overline{s}$) be a collection of polyhedra which form a complete polyhedral partition[1] of the regressor space X. Each polyhedron Xs is defined as:

$$X_s = \{x \in Rnx : Hsx \leq B_s\}, \tag{7.10}$$

for some matrix Hs and vector Bs of proper dimensions. The goal is thus to estimate Hs and Bs (with $s = 1, \ldots, \overline{s}$) defining the polyhedron Xs, where the s-th local affine submodel is active. Two approaches can be followed:

- according to the idea discussed in [11], the Voronoi diagram generated by the clusters' centroids can be used as a polyhedral partition of the regressor space X. Specifically, let cs be the centroid of cluster Cs. Then, the polyhedron Xs associated to cluster Cs (Equation (7.11)) is the set of all the values of the continuous state x such that cs is the closest centroid to x among all the other centroids c_j (with $j \neq s$), i.e.,

$$\mathcal{X}_s = \{x \in \mathbb{R}^{n_x} : \|x - c_s\|_2 \leq \|x - c_j\|_2, \quad j = 1, \ldots, \overline{s}, j \neq s\}, \tag{7.11}$$

Through simple algebraic manipulations, Xs can be expressed in a form like Equation (7.10), i.e.

$$\mathcal{X}_s = \{x \in \mathbb{R}^{n_x} : -2(c_s' - c_j')x \leq c_j' c_j - c_s' c_s, \quad j = 1, \ldots, \overline{s}, j \neq s\}. \tag{7.12}$$

Note that the definition of the polyhedron Xs (Equation (7.12)) only depends on the clusters' centroids, which can be easily updated recursively once the cluster Cs is updated (Step (2.3) of Algorithm 1). This makes the use of the Voronoi diagram particularly suited for real-time applications, where data are processed iteratively. However, a limitation of the Voronoi diagram is that it does not take into account how much the points are spread around the clusters' centres, making the state-space partition less flexible than general linear separation maps. In order to overcome this limitation, the approach described below can be followed.

- separate the clusters $\{\mathcal{C}_s\}_{i=1}^{\overline{s}}$ provided by Algorithm 1 via linear multi-category discrimination (see, e.g. [37–39]). In the following, we briefly describe the algorithm used in [37], which is suited for both offline and online computations of the state partition.

  The linear multi-category discrimination problem is tackled by searching for a convex piecewise affine separator function φ: $Rnx \rightarrow$

[1]A collection $\{\mathcal{X}_s\}_{s=1}^{\overline{s}}$ is a complete partition of the regressor domain $\mathcal{X}$ if $\bigcup_{s=1}^{\overline{s}} \mathcal{X}_s = \mathcal{X}$ and $\mathcal{X}_s^\circ \cap \mathcal{X}_j^\circ = \emptyset, \forall s \neq j$, with $\mathcal{X}_s^\circ$ denoting the interior of Xs.

R discriminating between the clusters $C_1, \ldots, C_{\overline{s}}$. The separator φ (Equation (7.13)) is defined as the maximum of $\overline{s}$ affine functions $\{\phi_i(x)\}_{i=1}^{\overline{s}}$, i.e.

$$\phi(x) = \max_{s=1,\ldots,\overline{s}} \phi_s(x) \tag{7.13}$$

with $\varphi_{s(x)}$ described as (Equation (7.14))

$$\varphi_s(x) = x'\omega^s \tag{7.14}$$

where $\omega s \in Rnx$ ($s = 1, \ \ldots, \overline{s}$) are the parameters to be computed.

For $s = 1, \ldots, \overline{s}$, let Ms be an $ms \times nx$ dimensional matrix (with ms denoting the cardinality of cluster Cs) obtained by stacking the regressors $x(t)'$ belonging to Cs in its rows. If the clusters $\{C_s\}_{s=1}^{\overline{s}}$ are linearly separable, the piecewise-affine separator φ satisfies the conditions:

$$Msss \geq Ms\omega j + 1ms, \quad s, j = 1, \ldots, \overline{s}, s \neq j \tag{7.15}$$

where $1ms$ is an ms-dimensional vector of ones.

The piecewise-affine separator φ thus satisfies the conditions (Equation (7.16)):

$$\begin{cases} \varphi\,(x) = x'\omega^s & \forall x \in C_s, \ s = 1, \ldots, \overline{s} \\ \varphi\,(x) \geq x'\omega^j + 1 & \forall x \in C_s, \ s \neq j \end{cases} \tag{7.16}$$

From (7.16), the polyhedra $\{\mathcal{X}_s\}_{s=1}^{\overline{s}}$ are defined as

$$\mathcal{X}_s = \{x \in \mathbb{R}^{n_x} : (\omega^s - \omega^j)'x \leq -1, \quad j = 1, \ldots, \overline{s}, j \neq s\}.$$

The condition (7.15) thus suggests computing the parameters $\{\omega^s\}_{s=1}^{\overline{s}}$ by minimizing the convex cost

$$\min_{\omega^1,\ldots,\omega^{\overline{s}}} \sum_{s=1}^{\overline{s}} \sum_{\substack{j=1 \\ j\neq s}}^{\overline{s}} \frac{1}{m_s} \|([M_s - \mathbf{1}_{m_s}](\omega^j - \omega^s) + \mathbf{1}_{m_s})_+\|_2^2, \tag{7.17}$$

with $(\cdot)+$ defined as $f+ = \max\{0, f\}$. Problem (7.17) minimizes the averaged squared 2-norm of the violation of the inequalities in Equation (7.15). The solution of the convex problem (7.17) can be then computed numerically in two ways: (i) offline through a Regularized Piecewise-Smooth Newton method or (ii) online through a Stochastic Gradient Descent method, as explained in [10].

7.5 Integration of Additional Functionalities to the IEC 61499 Platform

The DAEDALUS automation platform is built on top of the IEC-61499 standard and makes it the main core technology to enable the implementation of industrial grade applications in distributed control scenarios. The function block (FB) is one of the base elements of this standard. Function blocks are a concept to define solid, reusable software components in industrial automation systems. They allow the encapsulation of algorithms in an easy, understandable, even for newcomer, and usable form. Each function block has defined inputs, which are read and processed from an internal algorithm. The result will be outputted at defined outputs. Whole applications can be created out of various function blocks by connecting their inputs and outputs. Concretely, each function block consists of a head, a body, input/output events and input/output data.

The IEC 61499 standard defines various kinds of function blocks:

- Basic Function Blocks. Basic function blocks are used to implement basic functionalities of applications. Basic function blocks include internal variables, one or more algorithms and an "Execution Control Chart", to define the processing of the algorithms;
- Service Function Blocks. Service function blocks represent the interfaces to the hardware;
- Composite Function Blocks. Several basic, service or composite function blocks as well can be grouped to form a composite function block. The composite FB presents itself as a closed function block with a clearly defined interface.

7.5.1 A Brief Introduction to the Basic Function Block

Basic function blocks are the atomic units of execution in IEC 61499. A basic FB consists of two parts, i.e. a function block interface and an execution control chart (ECC) that operates over a set of events and variables. The execution of a basic FB entails accepting inputs from its interface, processing the inputs using the ECC and emitting outputs.

A basic FB is encapsulated by a function block interface, which exposes the respective inputs and outputs using ports. These input and output ports may be classified as either event or data ports.

Figure 7.11 shows the interface of the function block that implements a valve control logic. This interface exposes input events (INIT,

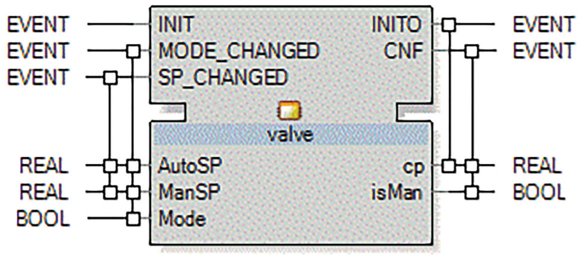


Figure 7.11 Valve: an example of basic function block.

MODE_CHANGED, SP_CHANGED), output events (INITO, CNF), as well as input variables (AutoSP, ManSP, mode) and output variables (cp, isMan).

Event ports are specialized to accept or emit events, which are pure signals that represent status only, i.e. they are either absent or present. On the other hand, data ports can accept or emit valued signals that consist of a typed value, such as integer, string or Boolean. Variable ports of a special type Any can accept data from a range of typed values. In addition, a concept of multiplicity is also applicable to data ports, which allows accepting or emitting arrays of values. A data port can be associated with one or more event ports.

As shown in Figure 7.11, for example, Mode is associated with MODE_CHANGED.

However, this association can only be defined for ports of the matching flow direction, e.g. input data ports can only be associated with input event ports. This event–data association regulates the data flow in and out of a basic FB, i.e. new values are loaded or emitted from the data ports on the interface when an associated event is present.

The behaviour of a basic FB is expressed as a Moore-type state machine, known as an ECC. An ECC reacts to input events and performs actions to generate the appropriate outputs.

Figure 7.12 shows the ECC of the valve basic function block, which consists of four states: START, INIT, exec_SPChange and exec_ModeChange.

States in ECCs have provision to execute algorithms and emit output events upon ingress, which are represented as ordered elements in their respective action sets.

As an example, in Figure 7.12, the algorithm exec_SPChange is executed (represented as a gray label), and the CNF event is emitted upon entering the exec_SPChange state (represented as a blue oval).

The execution of an ECC starts from its initial state (START in Figure 7.12) and progresses by taking transitions, which are guarded by an

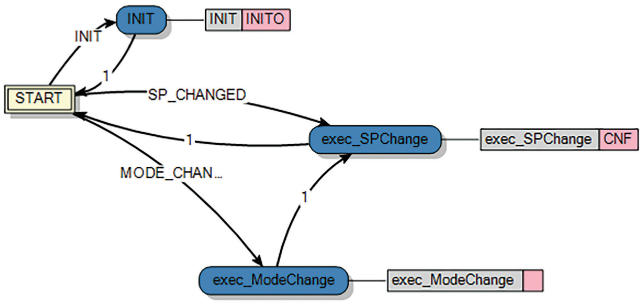
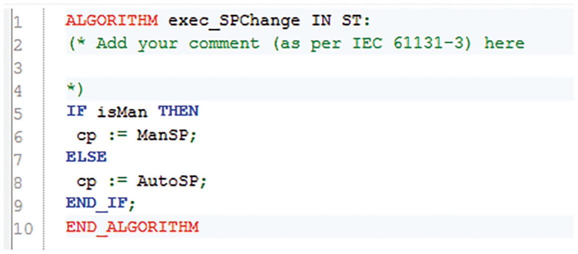


Figure 7.12 Example of execution control chart (ECC).



```
ALGORITHM exec_SPChange IN ST:
(* Add your comment (as per IEC 61131-3) here

*)
IF isMan THEN
 cp := ManSP;
ELSE
 cp := AutoSP;
END_IF;
END_ALGORITHM
```

Figure 7.13 exec_SPChange algorithm from the valve basic FB.

input event and an optional Boolean expression over input and/or internal variables. Upon evaluation, a transition is considered to be enabled if the respective guard condition evaluates to true. The ECC will then transition to the next state by taking the enabled egress transition from the source state to the corresponding target state.

An algorithm is a finite set of ordered statements that operate over the ECC variables. Typically, an algorithm consists of loops, branching and update statements, which are used to consume inputs and generate outputs. The IEC 61499 standard allows algorithms to be specified in a variety of implementation-dependent languages. As an example, the implementation from nxtControl allows the development of custom algorithms in Structured Text (ST).

The exec_SPChange algorithm from the valve basic FB is presented in Figure 7.13 that uses the ST language as defined in IEC-61131-3. Here, the IF–THEN–ELSE construct is used to update the output value of cp based on the value of the input isMan.

7.5.2 A Brief Introduction to the Composite Function Block

Composite function blocks facilitate the representation of structural hierarchy. Composite FBs are similar to basic FBs in the sense that they too are encapsulated by function block interfaces. However, unlike a basic FB, the behaviour of a composite FB is implemented by a network of function blocks.

Basic and composite function blocks characterize different types of specifications, which are referred to as function block types (FBTypes). A function block network may consist of instances of various FBTypes, where any given FBType may be instantiated multiple times. This concept is very similar to the object-oriented programming paradigm, which contains classes (analogous to FBTypes) and their instances, namely objects (analogous to FB instances). These FB instances connect and communicate with each other using wire connections, and with external signals via the encapsulating function block interface. This facilitates the structural hierarchy, i.e. a given function block network may contain instances of other composite FBs that encapsulate sub-FBNs.

Figure 7.14 shows a function block network with three function block instances that communicate with each other using wire connections, e.g. a Real output value SetPoint of the AutoCommand instance can be read as AutoSP by the valve instance.

Furthermore, some signals directly flow from the interface of the top-level composite FB into the encapsulated function block network, e.g. the event MODE_UPDATED is read from an external source and made available to the MODE_CHANGED input event of both the AutoCommand and valve instances. However, only compatible signals flow in this manner, meaning that an input event on a composite FB interface can only flow into an input event of nested FB interfaces. Similarly, data flow in this manner must also conform to data-type compatibility, e.g. a Boolean input on the composite FB interface cannot flow into a string type input of the nested FB interface. One exception to this rule is the Any type, which, as the name suggests, can accept any data type. This mode of signal flow is thus directly responsible for effecting the interface definition of a composite FB, i.e. if a nested FB needs an input from an external source, there must be an input defined on the composite FB interface, which flows into the said nested FB. This encapsulation of nested FBs from external sources simplifies the reuse of FBTypes.

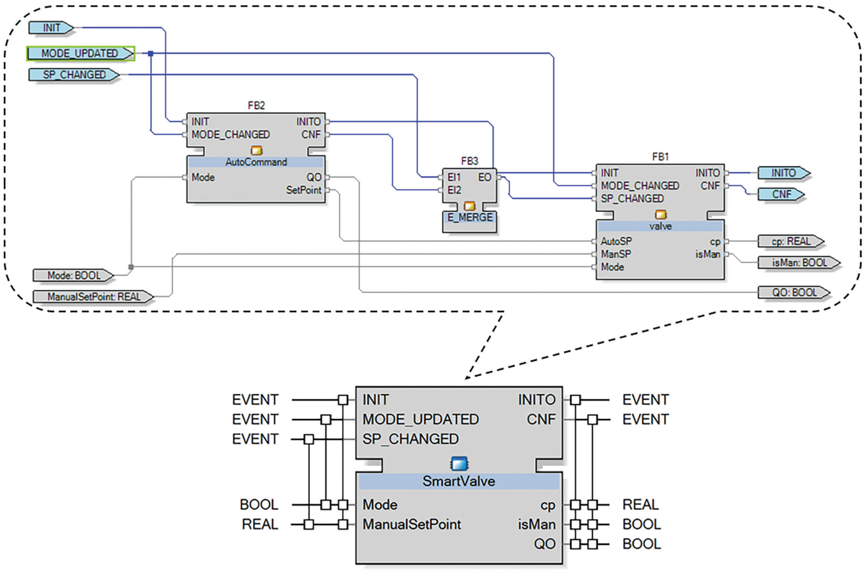


Figure 7.14 A composite function block with an encapsulated function block network.

7.5.3 A Brief Introduction to the Service Interface Function Block

Service interface function blocks (SIFB) can be considered as device drivers that connect the external environment with function block applications. These blocks are used to provide services to a function block application, such as the mapping of I/O pin interactions to event and data ports and the sending of data over a network.

There are two categories of SIFBs described in the standard, namely communication function blocks and management function blocks. While composite FBs capture centralized entities, resources are reminiscent of tasks and devices represent PLCs. Hence, both resources and devices need specific entities that facilitate either task-level (inter-resource) or distributed (inter-device) communication.

Communication function blocks are SIFBs providing interfaces that enable communication between IEC 61499 resources. Within the context of IEC 61499, a resource is a functional unit contained in a device that has independent control of its operations, so it may be created, configured, parameterized, started up, deleted, etc., without affecting other resources. The goal of a resource is to accept data and/or events from one or more interfaces, elaborate them and return data and/or events to some interfaces.

For the sake of completeness, it is worth mentioning that an IEC 61499 device contains one or more interfaces and those interfaces can be of two different types: communication and process. While communication interfaces provide a mapping between resources and the information exchanged via a communication network, a process interface provides a mapping between the physical process (e.g. analog measurements, discrete I/O, etc.) and the resources. Different types of communication function blocks may be used to describe a variety of communication channels and protocols.

On the other hand, management function blocks are SIFBs that are used to coordinate and manage application-level functionalities by providing services, such as starting, stopping, creating and deleting function block instances or declarations. They are somewhat analogous to a task manager in a traditional operating system. Unlike basic FBs, where the behaviour is specified using an ECC, SIFBs are specified using time-sequence diagrams.

7.5.4 The Generic DLL Function Block of nxtControl

The IEC 61499 software tool engineered by nxtControl provides a mechanism to integrate custom code in an IEC 61499 application. The mechanism is called Generic DLL function block and enables the exploitation of custom IEC 61499 function blocks interfaced by means of an abstract interface layer.

It provides the possibility to implement basic or service IEC 61499 function blocks in a custom programming language that are compiled in a dynamical loadable library (DLL) and then loaded and bound to the IEC 61499 runtime at the execution phase.

The Generic DLL function block mechanism builds on top of two components:

- a DLL that exposes a C interface where a predefined number of functions and data structures (embedded in a prototype which follows a well-defined template) implement the custom functionalities to be integrated in the distributed control application;
- a graphical representation of the custom function block, whose FBType is FB_DLL, and which is used in the nxtControl's engineering software environment to instantiate as many FBs as needed.

Such a mechanism enables the development of customized FB, providing:

- a representation of the IEC 61499 simple data types (as well as one-dimensional arrays of them) and plain C types;
- an input/output interface for passing these data between the IEC 61499 runtime software and the DLL implementation;

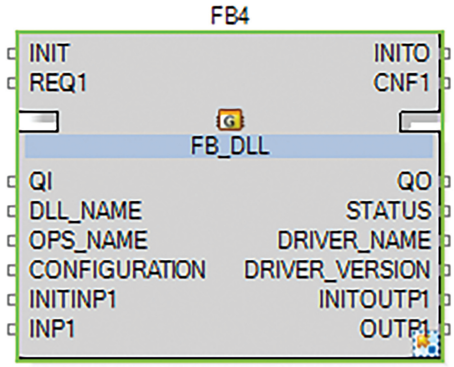


Figure 7.15 Example of FB_DLL function block.

- an interface for a custom function block where one initialization event and an arbitrary number of input events can be fed;
- the possibility to generate output events asynchronously;
- an interface to register and unregister a function block with the custom DLL;
- a way to query the provided data interface, so it is possible to implement consistency checks or to implement operations on different data types by one implementation;
- the possibility to implement several function blocks through a single DLL.

More than one instance of the Generic DLL function block (FB_DLL, Figure 7.15) can be instantiated in an IEC 61499 application, and the parameters provided as input to those FBs are exploited to select the appropriate DLL. All the FB_DLL instances are characterized by an INIT input event that is used to load the DLL: in particular, when the INIT event of any FB_DLL is received for the first time, the associated DLL is loaded and the IEC 61499 runtime registers the function block with that DLL. Furthermore, if the constructor is implemented in the custom code, then it is run afterward.

To leverage this flexible customization mechanism for implementing distributed automation applications, the custom code has to expose a data structure whose specification is detailed in the nxtControl's documentation material. That interfacing structure defines different elements that characterize the generic DLL function block, like:

- the number of input and output events;

- the number of data values that are associated to the input and output events;
- the data type associated to data values.

In addition to the description of the input/output events and data, the custom code used in a generic DLL function block has to define a precise set of functions that the IEC 61499 runtime uses to interact with the DLL when the distributed control application needs to execute the custom code. The most relevant of such functions are those used to register/unregister an FB_DLL with the appropriate DLL, the one used to execute the code associated to a specific input event, as well as the one dedicated to signal the triggering of an output event. In addition to those, there is also a function dedicated to the log information that can be used by the code in the DLL to report diagnosis information to the IEC 61499 runtime.

7.5.5 Exploiting the FB_DLL Function Block as Interfacing Mechanism between IEC 61499 and External Custom Code

Leveraging the generic DLL function block it is possible to extend functionalities available in the nxtControl automation platform with additional features that can be integrated in a seamless manner into an IEC 61499 control application.

That possibility opens the opportunity to integrate in an engineering software tool, designed to develop IEC 61499 applications, features that are not strictly related to the standard itself but that are interesting for implementing advanced distributed control applications. Actually, this can be leveraged to integrate the advanced functionalities that characterize a CPS that conforms to the DAEDALUS' vision, as for example, the integration of the "simulation dimension" and advanced MPC algorithms.

The possibility to extend the type of elaborations that can be performed within a function block in a distributed control application based on IEC 61499 enables the possibility to introduce new functionalities. Furthermore, it enables to test new features while respecting the normative rules and constraints of the standard and, as a consequence, allows to keep a high level of portability of the solution developed by means of this mechanism.

Since the DLL code is developed and compiled outside the classic development toolchain that is normally used for a plain IEC 61499 application (i.e. leveraging the development environment from nxtControl), the DLL has to be compiled by means of appropriate software tools to address the specific platform where the DLL will run. This means that an appropriate software toolchain is needed to generate a binary code that can run on the controller platform selected.

The main constraints that characterize this approach are:

- All the algorithms that define the behaviour of the FB_DLL have to be compiled as a dynamic loadable library (DLL) with a binary format compatible with the architecture of the controller, where the DLL will have to be installed;
- The DLL has to expose a C interface corresponding to the template imposed by the generic DLL function block mechanism;
- In the case where the FB_DLL is conceived to provide an output event to confirm the completion of the elaboration performed by the FB before a new input event can be processed by the FB, the elaboration performed by the DLL has not to take too much time before generating the output event. Otherwise that elaboration can affect negatively the controller's real-time performance;
- When the elaboration to be performed takes many computational resources and a lot of time to generate a result from the elaboration, another approach should be used: for example, the approach to run elaborations in parallel and generate output events asynchronously is a valid alternative;
- One of the aspects that needs to be considered at design is that a DLL can be shared by all the FB_DLL instances that make use of that library. This means, as a consequence, that the current number of function blocks registered with a DLL have to be managed appropriately, in order to keep track of the code portions that need to be executed for each FB_DLL instance.

The compact approach

The first approach enabled by the use of the generic DLL function block consists in exploiting the mechanism to implement a basic function block fully customized, where the constraint of using an execution control chart (ECC) is no more effective. In this case, the developer can freely design the finite state machine for government of the function block's logic states by exploitation of any preferred development tool (Figure 7.16).

By means of this approach, the logic algorithms that need to be executed when the associated input events are received by the FB_DLL instance can be designed and implemented following a customized approach that satisfies the developer's preferences and needs. At the same time, this mechanism enables

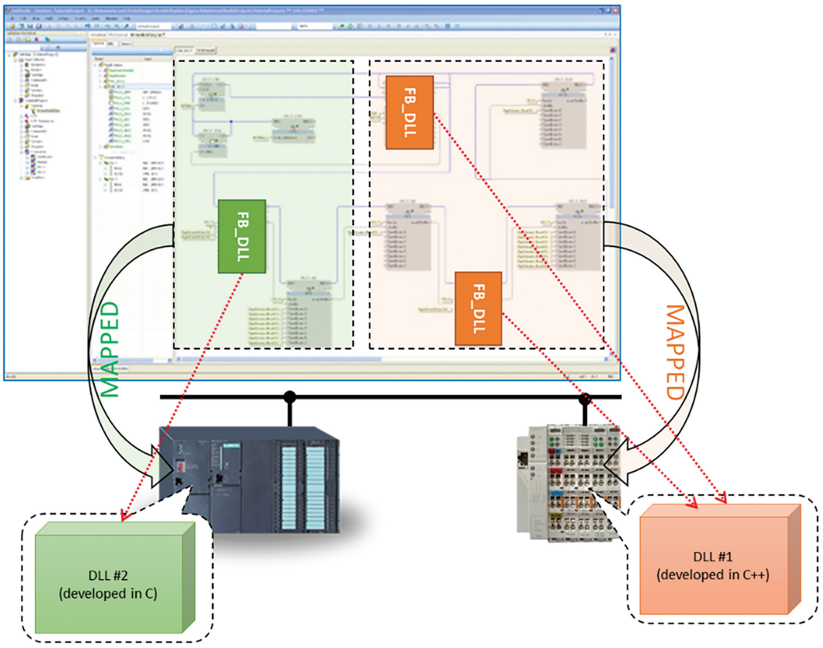


Figure 7.16 Illustration of the compact approach based on exploitation of generic DLL FBs.

to leverage other programming languages to implement the algorithms of the basic function block, in addition to the structured text (ST) language currently supported by the nxtControl software development tool.

The extended approach

A generalization of the previous approach consists in leveraging one or more additional DLLs when implementing the code associated to the FB_DLL instance. This basically means that the dynamic loadable library associated to the generic DLL function block is linked, in turn, to one or more other DLLs (Figure 7.17).

In such a case, it is possible that the exploitation of third parties' libraries implements customized function blocks usable in an IEC 61499 distributed control application. In this way, it is possible to develop custom service interface function blocks, making use of operating system function calls to access low-level hardware features or input/output data via interfacing devices.

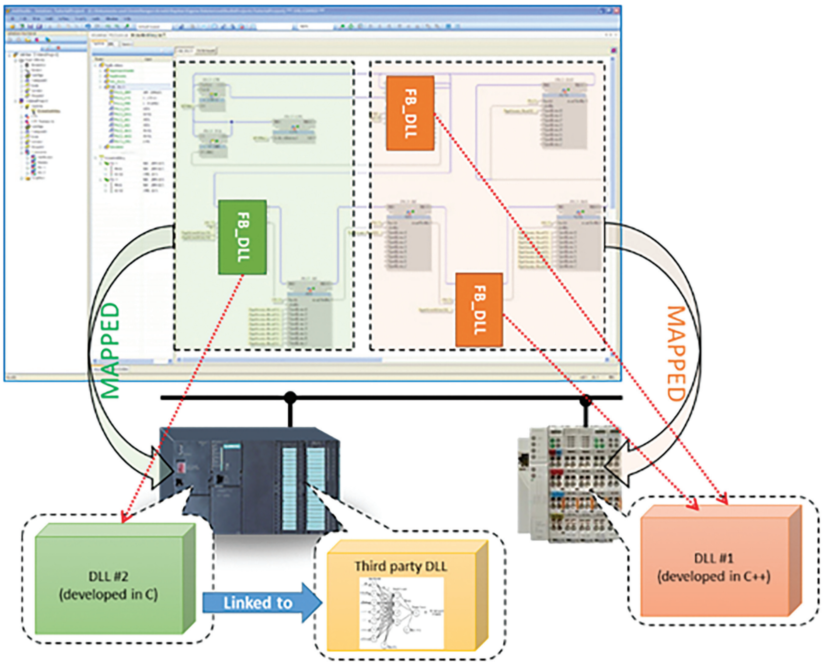


Figure 7.17 Illustration of the extended approach based on exploitation of generic DLL FBs.

In order to make this approach applicable, all the DLLs that are going to be exploited within the code of a general DLL function block have to be compiled for the specific architecture of the controller that will run that code.

That constraint can be limiting in certain scenarios, where the DLLs referenced by the custom code are not available for the platform selected and therefore it makes the use of those libraries impossible in such a scenario. On the other hand, that limitation has not to be ascribed to the generic DLL function block mechanism but to the lack of a compatible version of third party's libraries.

All the considerations that have been done for the basic approach of exploiting the FB_DLL are valid also for this extended case.

The distributed approach

The most general and flexible exploitation approach of the generic DLL function block mechanism consists not only in leveraging the FB_DLL FBs to integrate custom made and/or third-party software algorithms, but also in expanding the distributed computational network with additional

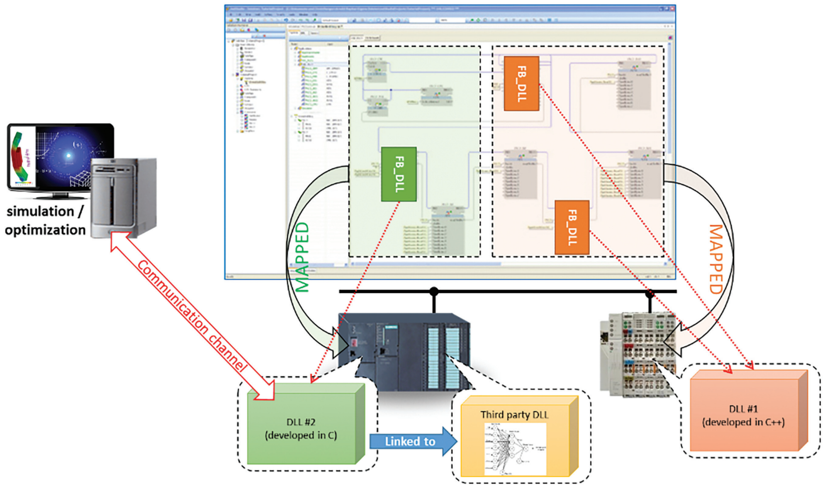


Figure 7.18 Illustration of the distributed approach based on exploitation of generic DLL FBs.

elaboration devices via interfacing mechanisms that can co-exist in parallel to the IEC-61499 communication interface (Figure 7.18).

This means that in addition to custom and advance algorithms embedded in DLLs that run locally in the controller where the FB_DLL instance is mapped, we can leverage the computational resources of other devices, in which specific data processing is allocated.

In such a scenario, the dynamic loadable library associated to an FB_DLL instance is used to open appropriate communication channels toward other computational nodes of the network where the data elaboration is actually performed. The FB_DLL has to leverage the asynchronous generation of events and appropriate mechanism to accept new requests in order to manage appropriately the elaboration and communication time without affecting negatively on the responsiveness of the IEC 61499 controller where the FB-DLL instance is running.

7.6 Conclusions

A deeply review of state of the art regarding solutions for controlling aggregated CPS has been carried out: the focus has been pointed on Model Predictive Control, especially on Hybrid Model Predictive Control. The analysis delves into Hybrid System representation and modelling, showing different techniques, mainly PWA and MLD. The advantages of PWA representation is related to the presence of numerous tools developed in the control system and identification fields, which are able to perform stability proof of system, convergence analysis. Moreover, PWA allows to build up an easier-to-use interface for SDK Interface in future development step. On the other hand, the MLD representation allows a deeply computational cost reduction for the solver as shown in Section 7.2.2. Both PWA and MLD are used in the SDK of Daedalus and they will work synergistically to improve the performance and the usability of the toolbox (SDK).

A review of the literature on data-driven modelling of hybrid systems has been carried out, with emphasis on PieceWise Affine (PWA) models and Jump Affine models, where the switches among the discrete state are triggered by deterministic events (e.g. if–then–else rules). These two models will be combined in the future stages of the project to arrive at Jump PieceWise Affine (JPWA) models, where the PieceWise Affine part will be used to describe the non-linear dynamics of the continuous (physical) states of the CPS, while the Jump part will be used to describe the time-evolution of the discrete (logical) states.

As a next step, a user-friendly software toolbox for identification of hybrid systems will be developed and the software functions will be integrated in the Daedalus' platform. This toolbox for on-line identification will contain the algorithm in ref. [37]. If necessary, improvements and/or extensions of this identification algorithm will be proposed and implemented in the toolbox. Benchmark examples available in the literature and case studies proposed by the project's partners will be used to test the implemented identification algorithms.

The IEC-61499 standard defines a technology for the implementation of distributed control applications applicable on several industrial scenarios. Many are the key aspects that make such a technology a valid solution for the development of the new generation of industrial control systems, leveraging networks of interacting CPSs.

The modularity that characterizes the control software design approach, which builds on the concept of function block, and the event-based execution paradigm are, just as an example, two of the core architectural aspects of the IEC-61499 standard that provide an effective development tool for complex control applications.

Advanced control software can be implemented exploiting the hierarchical development approach based on nesting of different types of function blocks. Custom algorithms can be implemented both through the composition of function blocks and by the development of Basic Function Blocks, leveraging the programming languages supported by the selected software development toolkit.

Acknowledgements

This work was achieved within the EU-H2020 project DAEDALUS, which has received funding from the European Union's Horizon 2020 research and innovation programme, under grant agreement No. 723248.

References

[1] V. Vyatkin, "The IEC 61499 standard and its semantics", IEEE Industrial Electronics Magazine, vol. 3, 2009.

[2] G. Optimization et al., "Gurobi optimizer reference manual", URL: http://www.gurobi.com, vol. 2, pp. 1–3, 2012.

[3] I. L. O. G. CPLEX, Reference Manual, 2004, 2011.

[4] B. a. T. M. Meindl, "Analysis of commercial and free and open source solvers for linear optimization problems", Forschungsbericht CS-2012-1, 2012.

[5] J. Lunze, F. Lamnabhi-Lagarrigue, Handbook of hybrid systems control: theory, tools, applications, Cambridge University Press, 2009.

[6] S. A. Nirmala, B. V. Abirami, and D. Manamalli, "Design of model predictive controller for a four-tank process using linear state space model and performance study for reference tracking under disturbances", in Process Automation, Control and Computing (PACC), 2011 International Conference on, 2011.

[7] J. G. Ortega, E. F. Camacho, "Mobile robot navigation in a partially structured static environment, using neural predictive control", Control Engineering Practice, vol. 4, pp. 1669–1679, 1996.

[8] M. Kvasnica, M. Herceg, L. irka, M. Fikar, "Model predictive control of a CSTR: A hybrid modeling approach", Chemical papers, vol. 64, pp. 301–309, 2010.

[9] J. Richalet, A. Rault, J. L. Testud, J. Papon, "Model predictive heuristic control: Applications to industrial processes", Automatica, vol. 14, pp. 413–428, 1978.

[10] F. Borrelli, A. Bemporad, M. Fodor, D. Hrovat, "An MPC/hybrid system approach to traction control", IEEE Transactions on Control Systems Technology, vol. 14, pp. 541–552, 2006.

[11] G. Ferrari-Trecate, E. Gallestey, P. Letizia, M. Spedicato, M. Morari, M. Antoine, "Modeling and control of co-generation power plants: a hybrid system approach", IEEE Transactions on Control Systems Technology, vol. 12, pp. 694–705, 2004.

[12] D. Corona, B. De Schutter, "Adaptive cruise control for a SMART car: A comparison benchmark for MPC-PWA control methods", IEEE Transactions on Control Systems Technology, vol. 16, pp. 365–372, 2008.

[13] E. Sontag, "Nonlinear regulation: The piecewise linear approach", IEEE Transactions on automatic control, vol. 26, pp. 346–358, 1981.

[14] A. Bemporad, M. Morari, "Control of systems integrating logic, dynamics, and constraints", Automatica, vol. 35, pp. 407–427, 1999.

[15] R. Alur, C. Courcoubetis, T. A. Henzinger, P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems", in Hybrid systems, Springer, pp. 209–229, 1993.

[16] W. P. M. H. Heemels, B. De Schutter, A. Bemporad, "Equivalence of hybrid dynamical models", Automatica, vol. 37, pp. 1085–1091, 2001.

[17] H. P. Williams, Model building in mathematical programming, John Wiley & Sons, 2013.

[18] J. H. Lee, "Model predictive control: Review of the three decades of development", International Journal of Control, Automation and Systems, vol. 9, no. 3, pp. 415–424, 2011.

[19] C. A. Floudas, Nonlinear and mixed-integer optimization: fundamentals and applications, Oxford University Press, 1995.

[20] R. Fletcher, S. Leyffer, "Numerical experience with lower bounds for MIQP branch-and-bound", SIAM Journal on Optimization, vol. 8, pp. 604–616, 1998.

[21] A. Makhorin, "GLPK", GNU Linear Programming Kit, 2004.

[22] E. F. a. C. B. Camacho, Model predictive control in the process industry, Springer Science & Business Media, 2012.

[23] E. B. E. Y. a. I. E. G. Perea-Lopez, "A model predictive control strategy for supply chain optimization", Computers & Chemical Engineering 27.8 (2003), vol. 27, no. 8–9, pp. 1201–1218, 2003.

[24] J. e. a. Łirokʅ, "Experimental analysis of model predictive control for an energy efficient building heating system", Applied energy, vol. 89, no. 9, pp. 3079–3087, 2011.

[25] S. e. a. Li, “Model predictive multi-objective vehicular adaptive cruise control”, IEEE Transactions on Control Systems Technology, vol. 19, no. 3, pp. 556–566, 2011.

[26] A. A. a. A. Bemporad, “A survey on explicit model predictive control”. Nonlinear model predictive control, in Non Linear Model Predictive Control, Springer Berlin Heidelberg, pp. 345–369, 2011.

[27] S. Y. Xu, T. W. Chen et al., “Robust H-infinity control for uncertain stochastic systems with state delay”, IEEE Transactions on Automatic Control, vol. 47, pp. 2089–2094, 2002.

[28] L. Breiman, “Hinging hyperplanes for regression, classification, and function approximation”, IEEE Transactions on Information Theory, vol. 39, pp. 999–1013, 1993.

[29] S. Paoletti, A. L. Juloski, G. Ferrari-Trecate, R. Vidal, “Identification of hybrid systems a tutorial”, European journal of control, vol. 13, pp. 242–260, 2007.

[30] A. Garulli, S. Paoletti, A. Vicino, “A survey on switched and piecewise affine system identification”, in 16th IFAC Symposium on System Identification, Brussels, 2012.

[31] N. Ozay, C. Lagoa, M. Sznaier, “Set membership identification of switched linear systems with known number of subsystems”, Automatica, vol. 51, pp. 180–191, 2015.

[32] J. B. Lasserre, “Global optimization with polynomials and the problem of moments”, SIAM Journal on Optimization, vol. 11, pp. 796–817, 2001.

[33] H. Ohlsson, L. Ljung, S. Boyd, “Segmentation of ARX-models using sum-of-norms regularization”, Automatica, vol. 46, pp. 1107–1111, 2010.

[34] D. Piga, R. Tth, “An SDP approach for 0-minimization: Application to ARX model segmentation”, Automatica, vol. 49, pp. 3646–3653, 2013.

[35] J. Roll, A. Bemporad, L. Ljung, “Identification of piecewise affine systems via mixed-integer programming”, Automatica, vol. 40, pp. 37–50, 2004.

[36] G. Ferrari-Trecate, M. Muselli, D. Liberati, M. Morari, “A clustering technique for the identification of piecewise affine systems”, Automatica, vol. 39, pp. 205–217, 2003.

[37] V. Breschi, D. Piga, A. Bemporad, “Piecewise Affine Regression via Recursive Multiple Least Squares and Multicategory Discrimination”, Automatica, vol. 73, pp. 155–162, 2016.

[38] K. P. Bennett, O. L. Mangasarian, "Multicategory Discrimination via Linear Programming", Optimization Methods and Software, vol. 3, pp. 27–39, 1994.

[39] Y. J. Lee, O. L. Mangasarian, "SSVM: A Smooth Support Vector Machine for Classification", Computational Optimization and Applications, vol. 20, pp. 5–22, 2001.