POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
MASTER THESIS IN AUTOMATION AND CONTROL ENGINEERING

FAULT TOLERANT MODEL PREDICTIVE CONTROL OF A DE-MANUFACTURING PLANT

Candidate:
Marco Morescalchi
Matricola 854372

Supervisor:
Riccardo Scattolini

Co-Supervisor:
Andrea Cataldo

Academic year 2016-2017

To my Uncle and my Grandmother,
because with their curiosity and their love for knowledge
they have inspired me to achieve this goal

I would like to express my gratitude to my supervisor, Prof. Scattolini, for sharing his valuable suggestions and experience with me. Moreover, I am profoundly indebted to my co-supervisor, Ing. Cataldo, who was very generous with his time and knowledge and assisted me in each step to complete the Thesis.

I wish to express my gratitude to my parents, my grandparents and my brother, who always supported me during this long journey. Without them, I could have never reached this goal.

The writing of an MA thesis could have made me very annoying. So, I must reserve a particular mention to Francesco, who has endured me every day in the last six months, trying to save me from my *aimlessness*.

I would like to thank all of my friends who have supported me over the years, enduring me and encouraging me to give my best. In particular Andrea, Massimiliano, Matteo and Gianluca, who most of all had to deal with my anxieties and difficulties.

I need to also thank my colleagues with whom I have shared this two years. In particular Davide, Davide, Francesco, Lorenzo and Marco, that I have stressed so much in the last months and that I am proud to call *Friends*.

Finally I should thank Rachele. I cannot find the words to say how much she has helped me in these years and to describe how fantastic she is. Thanks, really.

Abstract

The diffusion of electronic board is constantly increasing due to the wide spreading of mechatronics products. End of life management of such type of products is not currently approached in terms of process efficiency and environmental effects in a global perspective. This is due both to the rapid technological evolution and to the absence of integrated, automated and flexible systems able to treat mechatronics components under sustainable conditions. This topic is critical from an environmental point of view and represents a big opportunity for the manufacturing industry. To respond to this lack, a de-manufacturing pilot plant has been developed by CNR ITIA, the Institute of Industrial Technologies and Automation of the National Council for Research, in Milan. The plant is designed for testing, repairing and disrupting electronic boards, and is currently composed by a multi-path transport line and four machines with different functions: testing, repairing, discharging and loading/unloading. In this context, a primarily role is played by the conveyor system, that has been designed not only to be a simple assemblage of sensors/actuators able to move the pallets following predefined trajectories. Indeed, it is able to guarantee an efficient path of the boards in the network allowing to avoid bottlenecks, starvation and maximizing throughputs. This is due to the implementation of a multi-level control strategy in which, at the higher level, an MPC controller manages the movement of the pallets in order to optimize the performances while, at the lower level, PLCs acquire the sensor signals and drive the actuators. The MPC algorithm has been implemented in a C++ control platform that uses Matlab to handle the optimization problems, while the logic control belonging to PLCs has been implemented in ISaGRAF.

The aim of this project is to increment the performance on the transport line of the pilot plant, making it more suitable for a possible use in the industrial field.

In the first part of the dissertation, the problem of the delays introduced by the complexity of the software is considered. A massive adjustment of the C++ structure has been implemented and Matlab has been completely removed from the control environment. This has allowed to parallelize the calculations of the optimal solutions and the activities of the low-level control system, with a remarkable gain in terms of time.

Then, the efforts have been concentrated in reducing the computational cost of the optimization problems, through the introduction of some heuristic rules and the modification of the dynamic model. Some features not strictly related to the improvements of the performances of the plant have been introduced in order to enhance the operator experience.

In the last part of the Thesis, a fault detection and recovery algorithm has been designed and implemented. Indeed, given the current system, a fault in sensors or actuators blocked the whole de-manufacturing process even if it was not a critical error. Thanks to the nature of MPC control (which allows to exclude the action of the damaged component by simply adding constraints in the optimization problem) and to the architecture of the control system, a residual based approach able to be completely robust to the sensors faults has been developed.

Through the implementation of advanced control techniques, such as MPC with control horizon, combined with the application of heuristic rules developed thanks to an in-depth analysis of the structure of the line, clear improvements have been obtained through the lowering of the computing power required. This resulted in a significant reduction in the average time needed to calculate the solution, which in some cases reached more than 96% compared to the basic version of the controller. Moreover, thanks to the parallelism introduced between the problem-solving activity and movements realization, even the total production time has been sensitively reduced and in the majority of cases the optimization problem is completely hidden. This means that the first issue faced in Thesis has been completely solved, and the bottleneck of the production time is now due to the implementation of the movements of the pallets, which is fixed.
The further limitation to work on is finding a more effective configuration for the Cplex environment to solve the bug of saturation of the workstation in which the main process of the solver runs.

Concerning the detection and the handling of the faults occurring on the plant, the case of single failures has been studied. A knowledge-based model for residual matrix building and an opportune method to active fault recovery have been implemented with outstanding results. The system is now able to react to failures with only two steps of delay, respecting the constraint of not wasting time in reading the state of the transport line by the sensors.

A possible follow-up on this Thesis could therefore be the attempt to reduce the steps necessary for the controller to detect a fault. Furthermore, fault recovery is limited by the configuration chosen for the transport modules. Studying another arrangement that keeps the distance between the machines unchanged, but adds connections to the graph could be proven to be fundamental.

Sommario

Ł′ampia diffusione di prodotti meccatronici ha comportato, e sta comportando, un aumento nella diffusione di schede elettroniche. La gestione di fine vita di questo tipo di prodotti non è attualmente affrontata in termini di efficienza dei processi. Ciò è dovuto sia alla rapida evoluzione tecnologica sia all′assenza di sistemi integrati, automatizzati e flessibili in grado di trattare i materiali speciali che compongono le schede elettroniche in condizioni sostenibili. Questo argomento risulta critico da un punto di vista ambientale e rappresenta una grande opportunità per l′industria manifatturiera. Per rispondere a questa mancanza, un impianto pilota di de-produzione è stato sviluppato dal CNR-ITIA, l′Istituto di Tecnologie Industriali e Automazione del Consiglio Nazionale per la Ricerca, a Milano. Ł′impianto è progettato per testare, riparare e distruggere questo tipo di prodotti ed è attualmente composto da una linea di trasporto modulare e da quattro macchine con funzioni diverse: testing, riparazione, distruzione e caricamento/scaricamento dalla linea. In questo contesto un ruolo principale è svolto dal sistema di trasporto, che è stato progettato in modo tale da garantire che le schede siano smistate tra le macchine in maniera efficiente, evitando colli di bottiglia e massimizzando il rendimento. Ciò è possibile grazie all′implementazione di una strategia di controllo multi-livello in cui, a livello superiore, un controller MPC gestisce il movimento dei pallet per ottimizzare le prestazioni mentre, a livello inferiore, i PLC acquisiscono i segnali dei sensori e guidano gli attuatori.
Ł′algoritmo MPC è stato implementato in una piattaforma di controllo $C++$ che utilizza Matlab per gestire i problemi di ottimizzazione, mentre il controllo logico dei PLC è stato implementato in ISaGRAF.
Lo scopo di questa Tesi, è quello di incrementare le prestazioni dell′impianto pilota lavorando sul sistema di controllo della linea di trasporto. Andando a migliorare la produttività e la affidabilità della linea in ottica di un possibile futuro utilizzo in ambito industriale.
Nella prima parte della Tesi, viene considerato il problema dei ritardi introdotti dalla complessità della struttura del software di controllo. La piattaforma di controllo $C++$ è stata completamente ripensata e Matlab è stato rimosso dall′ambiente di controllo. Ciò ha permesso di ridurre i tempi andando ad eseguire in parallelo la gestione della soluzione del problema di ottimizzazione legato all′algoritmo MPC e le attività del sistema di controllo di basso livello.
Quindi, gli sforzi si sono concentrati sulla riduzione del costo computazionale dei problemi di ottimizzazione, attraverso l′introduzione di alcune regole euristiche e la modifica del modello dinamico. Alcune funzionalità, non strettamente correlate ai miglioramenti delle prestazioni dell′impianto sono state introdotte per migliorare l′esperienza dell′operatore.

Nell′ultima parte della tesi è stato progettato e implementato un algoritmo di rilevamento e recupero dei guasti. Infatti, dato il sistema attuale, un guasto nei sensori o attuatori bloccherebbe l′intero processo di de-produzione anche se non si trattasse di un errore così grave da compromettere le funzionalità dello stesso. Grazie alla natura del controllo MPC (che consente di escludere l′azione del componente danneggiato semplicemente aggiungendo vincoli nel problema di ottimizzazione) e all′architettura del sistema di controllo, si è sviluppato un approccio basato sullo studio on−line dei residui.

Attraverso l′implementazione di tecniche di controllo avanzate, come MPC con orizzonte di controllo, combinate con l′applicazione di regole euristiche sviluppate grazie ad una analisi approfondita della struttura della linea, sono stati ottenuti chiari miglioramenti dal punto di vista dell′abbassamento della potenza di calcolo richiesta. Ciò ha comportato una significativa riduzione del tempo medio necessario per calcolare la soluzione, che in alcuni casi ha raggiunto miglioramenti quantificabili in più del 96 percento rispetto alla versione base del controllore.

Si può affermare che il problema legato all′abbattimento dei tempi affrontato in questa Tesi è stato completamente risolto e che il collo di bottiglia del tempo di produzione è ora da imputarsi all′implementazione degli spostamenti dei pallet sulla linea. Per quanto riguarda questo tema, quindi, l′unico aspetto su cui poter lavorare è trovare una configurazione più efficace per l′ambiente Cplex ed eliminare così il bug relativo alla saturazione della memoria RAM.
In relazione al tema del rilevamento e della gestione dei guasti che si verificano sull′impianto, è stato studiato il caso di guasto singolo. Un modello basato sulla conoscenza del sistema per la costruzione della sua matrici dei residui e un metodo per il recupero attivo dei guasti sono stati implementati con ottimi risultati. Il sistema è ora in grado di reagire ai guasti con solo due passaggi di ritardo, rispettando il vincolo di non perdere tempo nel leggere lo stato della linea di trasporto dai sensori.
Un possibile follow-up su questa tesi potrebbe quindi essere il tentativo di ridurre i passaggi necessari affinchè il controllore rilevi un guasto. Inoltre, il ripristino degli errori è limitato dalla configurazione scelta per i moduli di trasporto. Studiare un altra configurazione che mantenga inalterata la distanza tra le macchine, ma aggiunga connessioni al grafico, potrebbe dimostrarsi fondamentale per avere una strategia di correzione del guasto efficace.

Contents

Chapter 1

Introduction

The aim of this project is to increment the performance and the reliability of a de-manufacturing plant for electronic boards, by improving the performance of the control system of its transport line. The work developed concerns two main topics: the reduction of the time needed to compute the optimal trajectories for maximizing the number of elements processed, and the introduction of a mechanism of faults detection and recovery that allows guaranteeing the plant activity even in the presence of malfunctioning. All the solutions elaborated have been tested and implemented on a pilot project on the CNR-ITIA (Institute of Industrial Technologies and Automation of the National Council) plant, designed to represent a pilot project oriented to provide a second life to the electronic components, otherwise addressed to disruption. The handling of the end of life of such type of products is going to be one of the most relevant problems of our century [8, 16]. This issue is crucial both for the sustainable development of the mechatronic sector and for the possibility for European countries to become independent o countries rich of raw materials.
For a better understanding of the work reported in the Thesis it is therefore essential to briefly describe the operating principles of the de-manufacturing plant and to give an overview of the control software environment.

In the Thesis the structure and the control system of the pilot plant is described in a synthetic way, to provide the reader with the tools to understand the topics covered. For a more in-depth discussion, several papers are available; the interested reader is referred to [3, 4].

1.1 General Description of the Pilot Plant

The plant is designed for analysing, repairing and disrupting electronic boards, and is currently composed by a multi-path carriage line and four machines with different functions: testing, fixing, discharging and loading/unloading. Its structure is shown in figure 1.1, in which the main elements are highlighted.
Regarding the control system, each machine is represented as a three-state automa (empty, manufacturing and end manufacturing with pallet still loaded). Therefore, from the controller point of view, there is no difference between the three working cells. In this way, the re-usability and longevity of the developed system are guaranteed, as the

transport line is freed from the specific plant machinery.
For the sake of completeness, however, a brief description of the tasks and operations of each machine is presented.

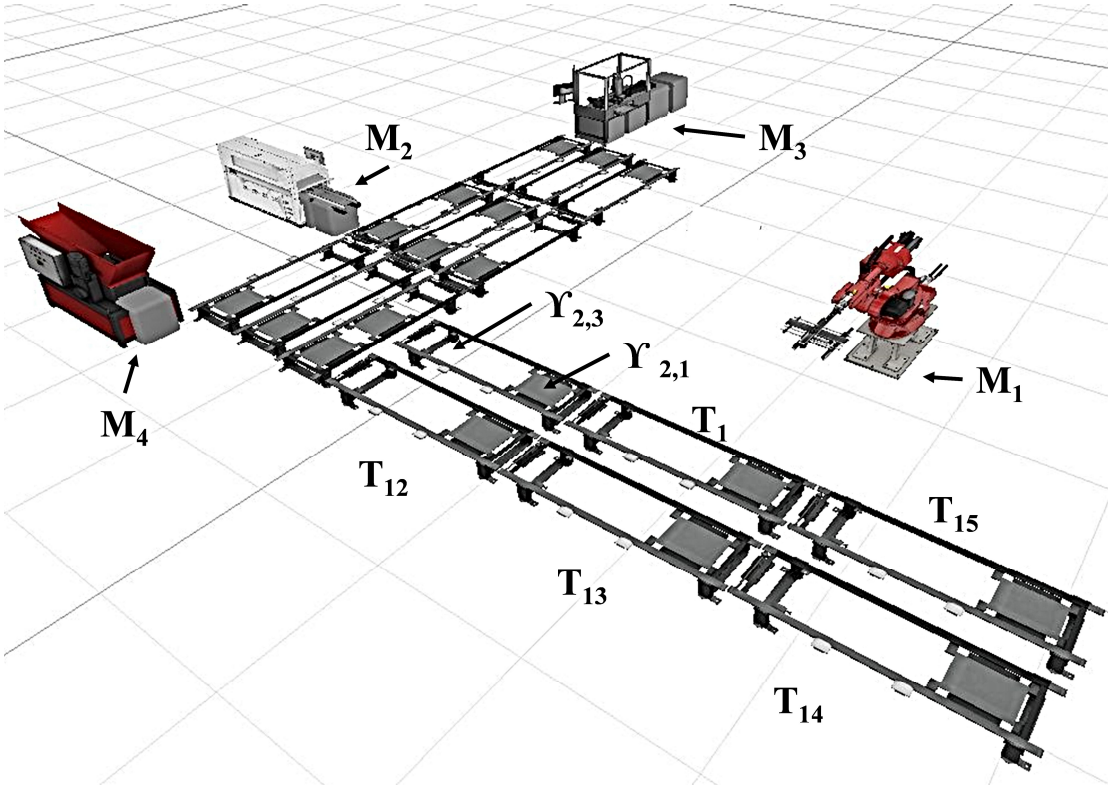

Figure 1.1: The de-manufacturing plant structure.

- Collaborative Robot Cell (M1). The electronic boards can be loaded or unloaded from the transport line. Concerning the loading phase, each board is extracted from the case of the wasted mechatronic products and inserted into a pallet, which is then loaded on the transport line. In the unloading phase, the reusable boards are removed from the pallets and stored.
- Testing Machine (M2). It examines the board received from the transport line to identify the presence of failures and, at the end of the operation, communicates to the controller a code corresponding to the failure detected.
- Reworking Machine (M3). The board previously analysed is repaired with a specific program for each type of malfunction.
- Discharge Machine (M4). The board impossible to fix is unloaded from the pallet and destroyed.

The main element of interest in this project is the pallet transport line. As in the majority of the manufacturing sites, the transport line has the essential role of allowing the exchanging of elements between the machines of the plant without the need of human intervention. The system has been designed to guarantee a high re-usability since it has

a modular structure, thanks to which it is easy to implement any change to the topology of the plant or substitute broken elements. The modularity is guaranteed by the fact that it consists of fifteen modules ($T_1, .., T_{15}$ in figure 1.1) designed in to have the same physical structure, on which the same components can be mounted. The basic idea is therefore to have elements that are easily interchangeable, keeping the possibility of customizing them according to specific needs. Given this, the transport line is easily adaptable to the changes required in a modern production line. In the specific case, the fifteen modules allow to appropriately configure to adapt the line to the conformation of the CNR laboratory. It is interesting to note that even within the same laboratory, the structure of the transport line has been modified several times depending on the changes introduced over time.
So, each module has a specific configuration depending on its position in the plant, but it is possible to present a general scheme useful to understand the working principles. Generally speaking, each element of the line is endowed with a conveyor belt, on which the pallets are placed and with a system of motors and sensors able to command the belt and then move the pallets along the possible directions imposed by the physical structure of the line. Furthermore, any module is ideally divided into up to three different areas called buffer zones (in the following they will be named BZs), where the pallets could rest their movements.
Each module is marked with a number, so that it is possible to localize a pallet in the network, using this number and the buffer zone occupied as parameters ($\gamma(i, j)$ will denote the jth position of the ith transport module). As an example [2], in figure 1.2, a transport module with three BZs is reported, together with the possible movements of the pallet. From the figure is easy to notice that from a BZ a pallet can be moved in different directions (forward, backward, lateral) depending on the specific combination of actuators and sensors installed on the module. So, different paths are possible and the problem of managing the routing on the transport line is not trivial.

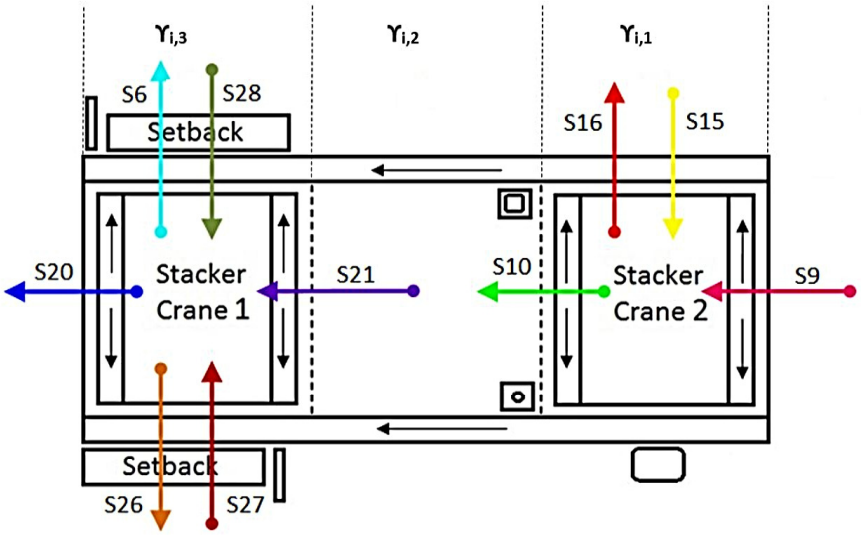

Figure 1.2: Schematic of the transport module

1.2 The control system

The controller of the plant has been developed according to the hierarchical, multi-level structure [22], shown in figure 1.3.
At the higher level, an MPC controller manages the movement of the pallets to optimize the plant performances based on the current status of the plant, i.e. the number and the position of pallets on the conveyor and the state of the machines.
At the lower level, a set of Programmable Logic Controllers (PLCs) solve the task of handling the sensors and the actuators. In fact, for every module a PLC acquires the data transmitted by the sensors and consequently turns on/off the actuators to move the pallets.
The division of the problem in two different sub-problems has allowed to completely separate the solution of the routing problem from the one regarding the implementation of the pallets movements.

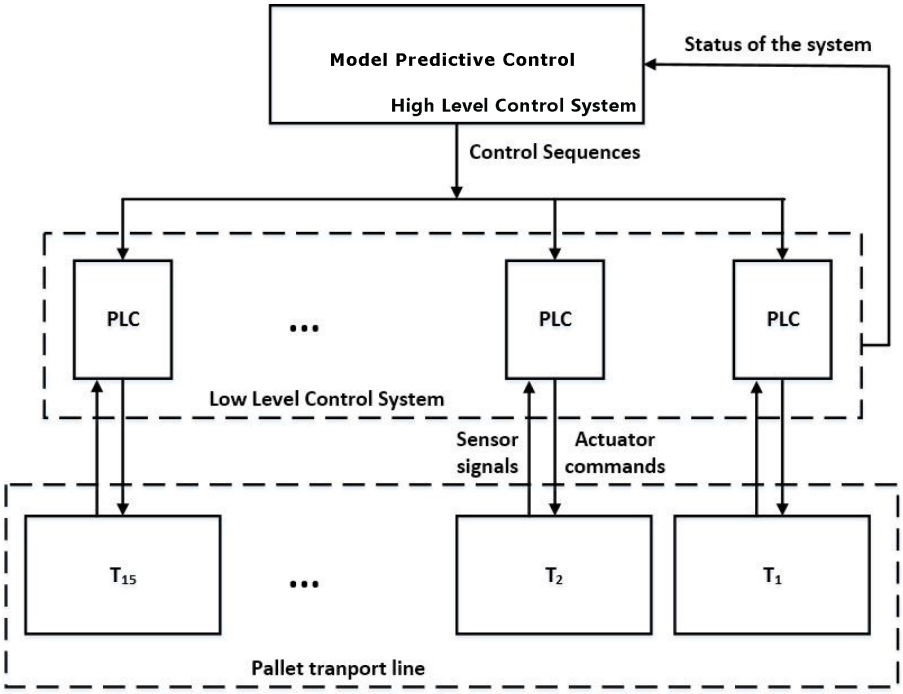

Figure 1.3: De-manufacturing pilot plant control architecture

Currently, this control architecture is distributed on three computers: a workstation and two supporting PCs. The workstation runs the the controller, that implements the high-level logics entrusted to a set of software with different tasks coordinated by a C ++ platform. On the same machine is handled also the low-level logic. They are implemented on ISaGRAF [11]. It is essential because, among other things, it compensates for the lack of physical PLCs by creating a virtual copy on the support computers connected to the system network. So, ISaGRAF sends the commands to the supports PC that elaborate them substituting the PLCs and managing the actuators and the sensors. The software architecture is schematised in figure 1.5.

Low level control system

The low-level control has been designed to handle the pallets movements through the different buffer zones according to the requests coming from the high-level control. It is composed of a set of logic sequences that specify how a PLC has to manage the actuators and the sensors of the module, which has to deal with. Thanks to the structure of the transport line, it is possible to classify a set of thirty-six control sequences (identified by arrows and labeled S_n, in figure 1.2) capable of achieving all possible pallet moves on the conveyor. Then, it has not been necessary to develop distinct control software for each PLC, but just to download the control sequences required to perform the pallet movement for each transport module, with a notable gain in terms of re-usability, maintenance and efficiency of the software.

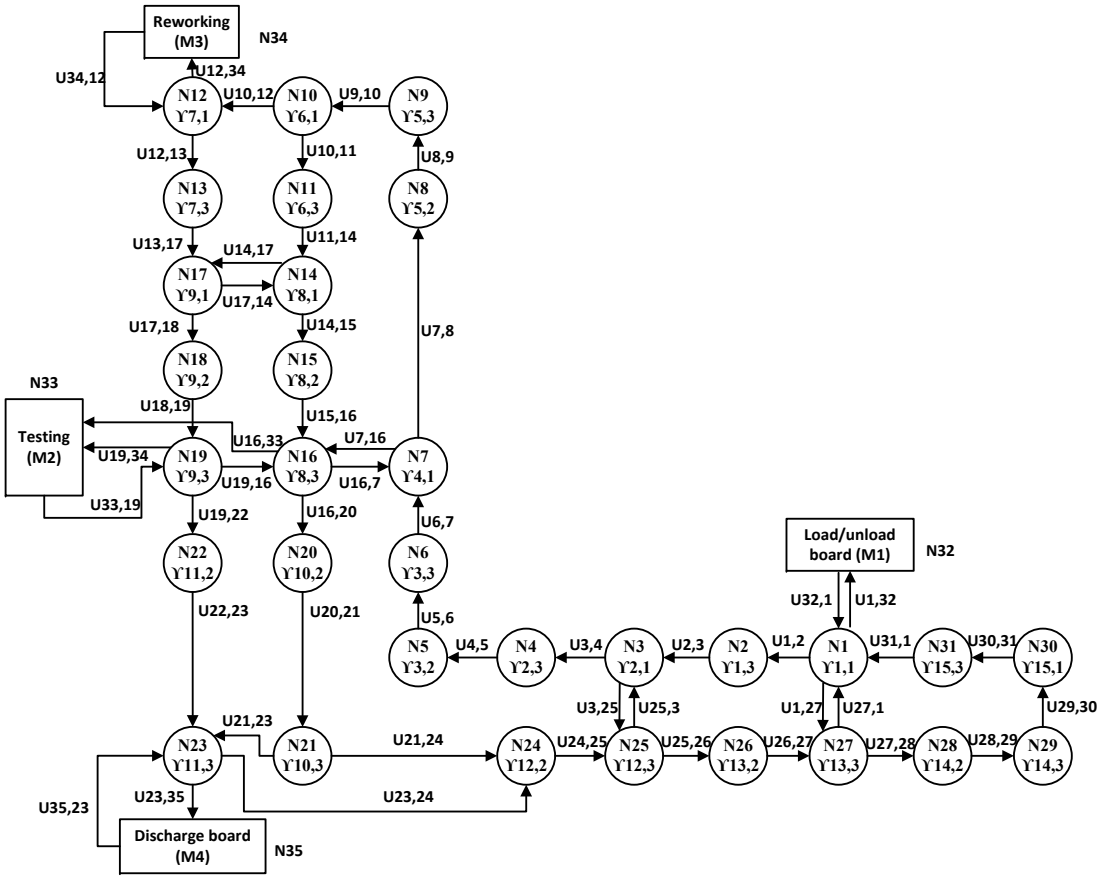

Figure 1.4: Graph representation of transport line.

Hence , for each movement between two buffer zones or two transport modules, there are one or more fixed prearranged control sequences to invoke. The coordinator does not need to know the commands to the actuators necessary to move the pallet from a position to another, but it only needs to know the two buffer zones involved in the movements. Consequently, to reduce the number of variables necessary to describe the moves, it is possible to define specific variables uniquely associated with the pallet motion between two particular buffer zones. Then, these variables could be considered the control actions computed by the HLCS, which have to be sent to the LLCS and to be

implemented to manage the plant.
The transport line can thus be described and represented by a direct graph, as in figure 1.4. The nodes represent both the BZs and the machines where the pallets can lay, while the arcs are associated with the HLCS commands used to move the pallet from a node to an adjacent one. The modelling of the plant in the form of a graph allows the development of advanced control techniques for the management of the routing on the transport line. In fact , several mathematical formalisms can be used to derive an accurate model that is entirely abstract with respect to the actual plant, but very reliable in the controller design.

Regarding the implementation, each control sequence is coded in the SFC programming language following the guidelines imposed by the IEC 61131-3 standard.

High level control system

The high-level control decides which control actions must be activated at any moment. Several control algorithms could be used to determine the evolution of the system. In particular, the controller considered in this Thesis can choose whether to use an advanced technique such as MPC, or a sequence of predefined control actions in a cyclic manner. To accomplish both techniques, different programs are needed. Indeed, the tasks to be fulfilled are different: it is necessary to store the information on the prediction model, solve optimization problems, manage the communication with the system, translate the commands sent from the top level to control commands compatible with PLCs, etc. In this context, a primary role is played by the DCPIP platform, which runs the control algorithm of the plant to be controlled. It is a C++ software that constitutes the kernel of the control system managing the communication between the principal software packages of the environment: ISaGRAF and MATLAB, used respectively as interfaces with the PLCs (and then with the plant) and for storing the prediction model and carrying out the operations necessary to calculate the control actions.

To understand how the high-level logic works it is essential to understand that the upper layer of the controller is based on a simple information-exchange mechanism. The various software, running on the same machine, are able to exchange information by writing and reading text files. The information transferred are vectors of integers thanks to which it is possible to define the system status at any time. For example, it is possible to describe the position taken by the pallets on the transport line through a vector of numbers from zero to five; zero indicates that a certain buffer zone is empty, and the numbers from one to five indicate the associated destination of each pallet on the line. Similarly, the control actions can be described by a vector of zeros and ones, where one indicates that the control action must be activated. To summarize, the operations performed are the following: in an initialization phase, the DCPIP platform starts the system by initializing the data structures necessary for communication with the other programs, while Matlab builds the model to be used to implement the MPC algorithm using HYSDEL; after this step, the DCPIP passes the state of the system to Matlab cyclically. Matlab solves the optimization problem associated with the control problem by using a toolbox provided by Cplex; when the solution is ready, it is stored into a text file, which is read by the DCPIP and passed to ISaGRAF, which in turn translates

it into a language that is understandable to the PLCs and starts the logic of the low level control. When the movement of the pallets ends, ISaGRAF communicates to the DCPIP the new state of the system which in turn passes it to Matlab and so on.

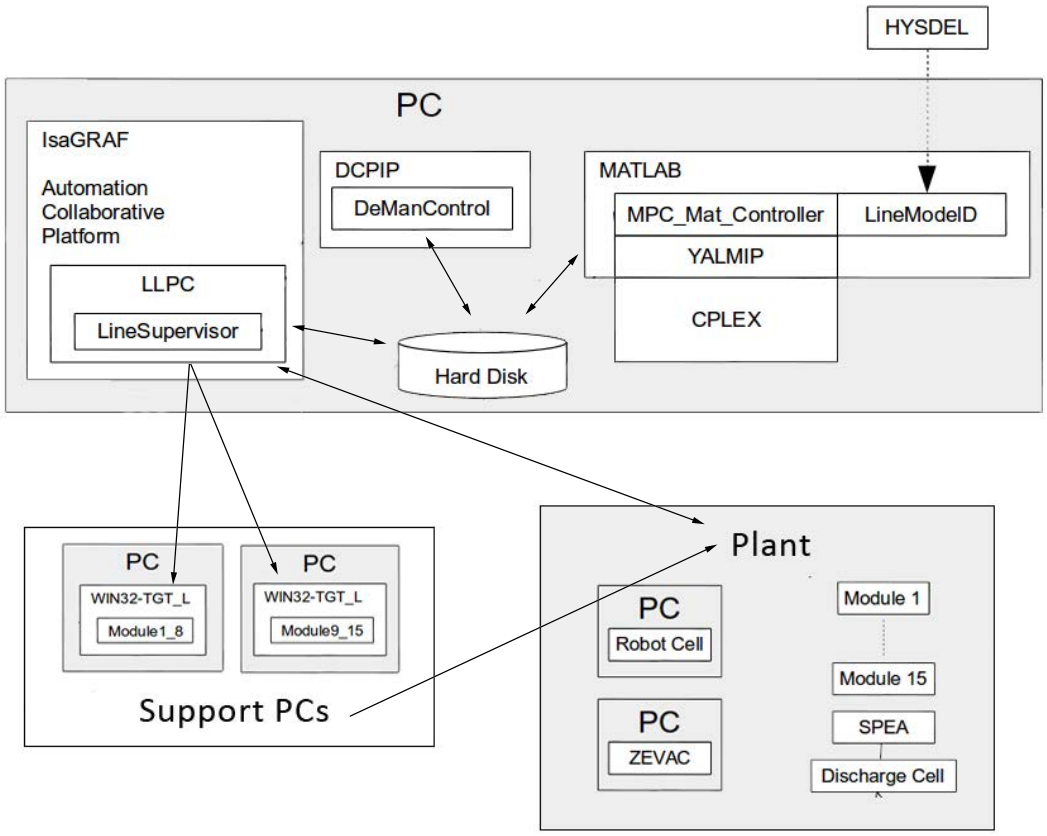

Figure 1.5: Software architecture

1.3 Low Level Control Implementation

To fully understand the operation of the plant we need to deepen the functioning of *ISaGRAF*. It is an environment that allows simulating and programming the PLCs of the plant and implements the line supervisor which handles the operations of the plant and the communication of the control system with the machines. Moreover, it works as an observer of the plant processes since, at each step, it stores the information about the state of the transport line and, through its mathematical model, he is able to calculate his future status starting from the control actions to be implemented.

It is very important to understand how communication with the low level is managed and how the project has been organized since a massive modification of this platform have been done during the thesis.
The workflow can be summarized as follows: an ISaGRAF program runs on the workstation carrying out the activity of plant supervisor while, on another computer, a different program simulates the operation of the PLCs. When the DCPIP writes the control

actions to the dedicated text file, it notifies that the data are available to the supervisor that reads the file and translates the control actions into control sequences. In the end, these sequences are activated and the secondary process manages the inputs/outputs of the corresponding module of the transport line. Once the operation associated with the sequences has been completed, this process returns an "all ok" signal to the supervisor; this estimates the system status starting from the previously read control actions and passes it to the DCPIP writing it on text. It is important to underline two main aspects:

- If the supervisor does not receive a feedback from the virtual PLCs, the whole control system is stuck. It means that even a non critical fault could block the operations of the plant.
- Both the DCPIP and ISaGRAF, and therefore the high-level control, are not aware of the data communicated by the sensors. The system status is calculated starting from the control actions through the knowledge of the mathematical model of the plant by the line supervisor. The choice of not reading the state of the plant from the sensor at each step is due to the fact that for this plant this operation is very time consuming.

The software structure is divided into three parts, which correspond to the three distinct programs generated during compilation. Two parts are related to the modules, and for each module the SFC sequences that implement the low level control are defined. The third part defines the logic of the line supervisor and the management of the machines, its elements are reported in figure 1.6.

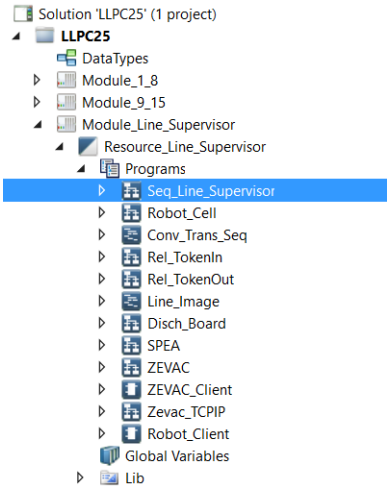

Figure 1.6: Module Line Supervisor

Each module of the software has local variables that are accessible from the other modules only through a one-way binding process, whereby one module variable is assigned the value of another in another module. This technique is used to allow PLCs to communicate to the line supervisor that they have finished their work. To enable virtual PLCs to communicate with the actuators and sensors, it is possible to configure the input and output ports connected to the network by associating the values of some variables.

1.4 Contents of the Thesis

During the thesis work, various issues related to both the design and the implementation phases have been addressed. Working on a pilot plant has allowed addressing problems of different nature and to experiment in a safe mode new features and ideas, even the ones not strictly related to the thesis topics. For example, the momentary failure of the robot cell has required to develop a new temporary way to unload/load the pallets from the plant.
So, many aspects of the de-manufacturing pilot plant have been taken into account; the main ones are briefly described below.

The first aspect analyzed regards the simplification of the software architecture. In particular, the Matlab environment has been removed to implement the MPC control algorithm entirely in C++ with a direct call to CPLEX. This operation required an appropriate manipulation of the matrices of the linear MLD model of the transport line, in particular, the unrolling over time of the dynamic, output and constraints matrices. Moreover, the operator interface has been changed to improve the human work use.

Then, the attention has been focused on studying the topology of the plant to individuate heuristic rules for reducing the complexity of the optimization problem derived from the MPC algorithm. Some set of buffer zones, called tunnels (in which there is only one possible path to follow from the pallets) have been selected and, for each of them, a mechanism able to exclude these areas from the computations has been developed. The pallets that enter in a tunnel are automatically pushed, without the need of computing the best path. Moreover, the cost function of the MPC problem has been modified to have an 'accumulation zone', close to the robot cell, where the empty pallets are pushed on.

Finally, taking advantage of the hierarchical control structure, a residual-based approach for the faults detection has been developed. Once a fault is identified, if it is not critical, the system automatically acts on the LLCS stopping the critical control sequence and adds constraints to the MPC controller to make the HLCS able to avoid the broken path and in this way to be able to restore the working activity. Otherwise, the whole plant is stopped, and the fault is signaled to the operator. Thanks to this approach, the controller has now become robust with respect to sensor and actuator faults.

1.5 Thesis structure

The thesis is organized as follows.
Chapter 2 describes the new software architecture and develops the optimization problem requested from the MPC technique starting from the MLD model of the system.
Chapter 3 considers the problem of the reduction of the computational time requested by the above optimization problem and describes the others changes to the control system to improve the user experience use.
Chapter 4 addresses the challenge of the design of fault tolerant controller in the case of sensors and actuators fault.
In Chapter 5, a possible solution on how the plant must respond after a fault occurs is presented.
Chapter 6 presents the conclusions of the work and some hints for future works.

1.6 List of publications

The research activity developed during the Thesis development has lead to the following publication:

International conferences proceedings

Cataldo A., Lanzarone E., Morescalchi M., Scattolini R. *Complexity reduction of Model Predictive Control for a de-manufacturing plant.* In *Proc. of the 16th IFAC Symposium on Information Control Problems in Manufacturing*. Bergamo, Italy, 11-13 June 2018.

Chapter 2

MPC Controller: algorithm formulation and software implementation

In this chapter, the MPC algorithm developed for controlling the pilot plant is formulated and a brief description of its implementation into the Dynamic Control Platform for Industrial Plants (DCPIP) is given. At the end, it is shown the configuration chosen for the platform delegated to solve the MILP problem.

Starting from the graphical representation of the plant (see figure 1.4) it has been possible to obtain a dynamic model of the transport line, which successively has been translated into a Mixed Logical Dynamic (MLD) formulation by linearising the non-linear terms and the logic propositions. The fundamental idea behind this technique is that the logical expressions can be rewritten into algebraic inequalities. The operating principles have been stated as simple predicates that have been combined using connectives and modifiers such as *and* ($\wedge$), *or* ($\vee$), *not* ($\neg$), *implies* ($\rightarrow$) or *if and only if* ($\leftrightarrow$). Thus, by exploiting the properties of boolean algebra, these conditions have been combined and translated into algebraic constraints. There are several techniques to do that and, depending on the one used, the model presents a different number of auxiliary variables and constraints. At the end, one can obtain a representation in which both continuous and discrete dynamics coexist and interact with each other. The ideas of MPC can be applied to this kind of systems, and it is called Hybrid MPC, but in this case, the mathematical problem associated is more computationally demanding than in the case of processes with only real variables. Once the expressions to derive the model of the transport line have been obtained, the objective function of our problem has been defined, and then it has been possible to start the implementation of the algorithm in C ++ code. The implementation process consists in developing the functions that, at each generic instant k, allow to automatically calculate the matrices used for solving the optimization problem and manage the interaction with the calculation software.

The modeling phase is not the subject of this thesis, so just an example of how the different elements of the system have been modeled will be provided to the reader. For more details, the complete model is available in [2]

It is important to underline that, at the beginning of this Thesis, the implementation of the MPC algorithm was delegated to the Matlab environment. Thanks to the help of the Yalmip and HYSDEL tools the matrices of the system were transformed into the constraints of the optimization problem; then the issue was solved by invoking Cplex, a very powerful optimization solver. Changing the control structure, eliminating Matlab and making sure that its role is incorporated into the DCPIP, has been essential in order to work on the improvements presented in the next chapters and has made it possible to eliminate the need to buy a rather expensive license with a view to a future industrial application.

2.1 Dynamic Model

In this section, the dynamical model of the system is derived as in [4].

Model of the Nodes

Each pallet is associated with an integer that denotes the destination, also called target, of the loaded electronic board. It is possible to define $\Gamma_i(k)$ as the value correspondent to the pallet target present in node (remember the nodes are the mathematical representation of buffer zones and machines) $N_i,\ i = 1,...,35$; it can takes the following values:

- $\Gamma_i(k) = 0$ if node N_i is not occupied by a pallet at instant k.
- $\Gamma_i(k) = m,\ m = 1,2,3,4$ if the node N_i contains a pallet in which is loaded a board to be sent, respectively, to the machines $M1, M2, M3, M4$.
- $\Gamma_i(k) = 5$ if node N_i is occupied by a pallet that is empty at instant k (typically when pallets exit from the discharge station they assume the value $\Gamma = 5$ until they are called by the robot cell to load a new board).

As concerns the not structurally null commands, the following variables are defined:

- $u_{i,j}(k) = \begin{cases} 0 & if\ the\ command\ is\ not\ active\ at\ k \\ 1 & otherwise \end{cases}$
- $I_{i,in}$ is defined as the set of indices j associated with $u_{j,i}$, which allow to move a pallet to the node N_i from an adjacent node N_j.
- $I_{i,out}$ is defined as the set of indices j associated with $u_{j,i}$, which allow to move a pallet from the node N_i to an adjacent node N_j.

To complete the model of the nodes correctly, it is advisable to define some constraints:

1. For each node N_i and at any time instant k, only one control input can be allowed

$$\sum_{j \in I_{i,in}} u_{i,j} \leq 1 \quad i = 1,..,35$$

2. For each node N_i and at any time instant k, only one control output can be allowed

$$\sum_{j \in I_{i,out}} u_{i,j} \leq 1 \quad i = 1,..,35$$

3. If a node is empty, no commands of output can be given

$$\Gamma_i(k) = 0 \rightarrow \sum_{j \in I_{i,out}} u_{i,j} = 0 \quad i = 1,..,35$$

Model of the Buffer Zones

All the following equations are defined for $i = 1,..,31$.
The dynamic equation that describes the pallet movement for the generic BZ and its target propagation is

$$\Gamma_i(k+1) = \Gamma_i(k) + \sum_{j \in I_{i,in}} \Gamma_j(k) u_{i,j}(k) - \sum_{j \in I_{i,out}} \Gamma_j(k) u_{i,j}(k)$$

A node N_i can contain at most one pallet, so, if it full, it is possible to activate an input control action ($u_{i,k} \in I_{i,in}$) only if at the same time an output control action ($u_{i,k} \in I_{i,out}$) is set to one so as to free N_i. It can be expressed as a constraint by imposing:

$$\Gamma_i(k) \geq 0 \ \wedge \sum_{j \in I_{i,out}} u_{i,j}(k) = 0 \ \rightarrow \sum_{j \in I_{i,in}} u_{i,j}(k) = 0$$

It is useful to define $\gamma_i(\Gamma_i(k))$ referred to a pallet occupying N_i as the minimal distance to its target. $\gamma_i(\Gamma_i(k))$ is equal to zero if N_i is empty or is occupied by an empty pallet. In all the other cases, it is equal to the length of the minimal path from node i to the target machine.
Another aspect to consider is the fact that the permanence of the pallet on the transport line should be penalized to force its movement toward the target machine and avoid deadlocks. To this end, a counter η_i for each buffer zone is defined and, at each time instant, its value is increased by one so as to it represent the number of instants in which the corresponding pallet has been on conveyor. When it reaches a machine, the counter is reset. For modelling a counter, the following variables are needed:

- $\delta_i(k)$ that is a boolean variable that indicates if at least one control action regarding N_i has been activated (in this case it is equal to one) or not (equal to zero).
- $\theta_i(k)$ that is a boolean variable that indicates if N_i contains a pallet with $\Gamma \neq 1 \; or \; 5$.

Then, the dynamic equation of the counter for the $i-th$ BZ is given by

$$\eta_i(k+1) = \eta_i(k) + \delta_i(k)\,\theta_i(k) + \sum_{j \in I_{i,in}} [\eta_j(k)+1]\,\theta_j(k)\,u_{i,j}(k) - \sum_{j \in I_{i,out}} \eta_i(k)\,\theta_i(k)\,u_{i,j}(k)$$

Model of the Machines

The generic machine M_i is described by a finite state machine (FSM) with three boolean states:

1. x_{i1} idle and empty machine
2. x_{i2} manufacturing
3. x_{i3} end manufacturing with pallet still loaded

To model its dynamic behaviour, the following rules are set.

1. Switch from x_{i1} to x_{i2}

$$x_{i1}(k) \wedge \sum_{j \in I_{i,in}} u_{i,j}(k) = 1 \rightarrow \begin{cases} x_{i1}(k+1) = 0 \\ x_{i2}(k+1) = 1 \end{cases}$$

   In x_{i1} the counter is kept at zero. When the control action u_{ij} (the action that moves the pallet from the BZ j to the machine i), the FSM changes its state from x_{i1} to x_{i2}.

2. Switch from x_{i2} to x_{i3}

$$x_{i2}(k) \wedge (n_i(k) \geq \tilde{n}_i) \rightarrow \begin{cases} x_{i2}(k+1) = 0 \\ x_{i3}(k+1) = 1 \end{cases}$$

   The counter n_i is increased at every step and when it reaches a certain threshold $\tilde{n}_i$, the changes passes from x_{i2} to x_{i3}.

3. Switch from x_{i3} to x_{i1}

$$x_{i3}(k) \wedge \sum_{j \in I_{i,out}} u_{i,j}(k) = 1 \rightarrow \begin{cases} x_{i3}(k+1) = 0 \\ x_{i1}(k+1) = 1 \end{cases}$$

   In x_{i3}, n_i is kept constant at $\tilde{n}_i$, and a new target Γ is assigned to the pallet. When the output control action $u_{i,j}$ is activated, the pallet is moved to the adjacent BZ of the transport line.

2.2 MPC Problem Formulation

The Model Predictive Control [17, 18] technique is a control method that is becoming more and more popular with the increase of computing power. It consists in formulating the control problem as a mathematical optimization issue, built on the dynamic model of the system to be controlled, and, in particular, on the prediction of the evolution of the system in a certain future horizon [5]. The derived optimization problem is then solved on-line at each sampling instant, obtaining the sequence of optimal inputs to be supplied to the system. In other words, at any time instant k, the controller, relying on the available information, solves the optimization problem with respect to the future control sequence $[u(k), .., u(k+N-1)]$ and applies only the first element $u^0(k)$. Then, at time instant k+1, a new optimization problem is solved, based on the new information acquired at instant k+1, along the prediction horizon $[k+1, n+N]$ [23].
Its success is mainly due to the possibility to include different goals in the optimization problem and the fact it explicitly include in the control problem formulation state and input constraints.

Generally speaking, an MPC controller can be described by two elements: the prediction model and the objective function (also called performance index).

Prediction Model

The MLD system used in this Thesis is described as follows.

$$\begin{cases} x(k+1) = Ax(k) + B_u u(k) + B_{aux} w(k) + B_{aff} \\ y(k) = Cx(k) + D_u u(k) + D_{aux} w(k) + D_{aff} \\ E_x x(k) + E_u u(k) + E_{aux} w(k) \leq E_{aff} \end{cases}$$

Where $x(k) = [x_r^T(k) x_b^T(k)]$ with $x_r(k) \in R^n$ and $x_b(k) \in \{0,1\}^{n_b}$ is the vector of the state variables, $u(k) = [u_b^T(k)]$ with $u_b(k) \in \{0,1\}^{l_b}$ is the vector of the control actions, $w(k) = [w_r^T(k) w_b^T(k)]$ with $w_r(k) \in R^r$ and $w_b(k) \in \{0,1\}^{r_b}$ is the vector of boolean auxiliary variables.

- In the model of the transport line, the vector of the state variables consists of 82 elements useful to describe the distribution of the pallets on the plant. Of these, 35 are integer variables (Γ_i) that indicate the final destination of the pallet associated to the $i-th$ BZ, 35 are integer variables (η_i) that store the counter associated to each pallet and 12 (x_{i1}, x_{i2}, x_{i3}) are boolean variables that describe the behaviour of the machines.
- The vector of the inputs (control actions) is composed by 51 boolean variables ($u_{i,j}$) which allow the pallet movement from a buffer zone to another one.
- The vector of auxiliary variables contains 399 variables, of these 302 are boolean and 97 are integer. They are used to implement the cost function, to set the physical constraints due to the topology of the network and to add some features like the off-limits area (it is not allowed to the pallet to move close to a machine when it is working) to the plant.

Objective function

It is possible to state the expression of the linear performance index J as:

$$J = \sum_{h=1}^{N} \Bigg\{ \underbrace{\sum_{i=1}^{35} \gamma_i(\Gamma_i(k+h))}_{(1)} + \underbrace{\sum_{i=32}^{35} q_{xi} x_{i3}(k+h)}_{(2)} + \underbrace{\sum_{i=1}^{31} q_{\eta_i} \eta_i(k+h)}_{(3)} + \underbrace{\sum_{(i,j)\in I_u} q_{ui,j} u_{i,j}(k+h-1)}_{(4)} + \underbrace{\sum_{(m,r,i,j)\in \Psi} \lambda_{m,r} \sigma_m(k+h-1) u_{i,j}(k+h-1)}_{(5)} \Bigg\}$$

For the problem addressed in this Thesis, J must be minimized with respect to the future control actions defined over the prediction horizon specified by the positive integer N.

To fully understand the elements considered in the performance index, it is appropriate to remind the meaning of some variables:

- $\gamma_{i,j}$ $j-th$ position of the $i-th$ transport module
- $\Gamma_i(k)$ target state of the pallet in node N_i
- $u_{i,j}$ control input which moves a pallet from N_i to N_j
- $x_{i,3}$ state of the machine M_i
- η_i counter associated to node N_i

and define some new one as:

- q_{xi},q_{η_i} weights in the performance index
- $q_{u_{i,j}}$ weight on the control action in the performance index
- λ_m weight on the off-limit zone in the performance index
- σ_m auxiliary binary variable for the off-limit zone performance index penalty
- I_u set of the pair (i,j) associated with all commands $u_{i,j}$ not structurally null
- $\Psi(m,r,i,j)$ set of the buffer zones included in the off-limit area

So, in the objective function the following terms are weighted: the number of steps necessary for a pallet to reach its target destination, whose contribution in the cost function is highlighted in the term (1) in the cost function; the permanence of already finished pallets into the machines, whose contribution in the cost function is highlighted in the term (2); the counters associated with the staying of the pallets on the transport line, whose contribution in the cost function is highlighted in the term (3); the control actions, whose contribution in the cost function is highlighted in the term (4); the permanence of a pallet in the nodes adjacent to the machines, to allow the manufactured pallets to exit the working cells and move towards their new target, whose contribution in the cost function is highlighted in the term (5).

The choice of the values of the weights of the performance index is not trivial. Some qualitative rules can be defined, but a proper tuning on the real plant must be performed to obtain optimal performances. Even the choice of the prediction horizon is not trivial. It must be selected big enough to avoid deadlocks due to contrasting paths on the pallets but not too large to slow down the system due to the high resolution times. Moreover, N should not exceed the minimum number of steps $\tilde{n}_i$ required to the machine M_i to finish its work (in this case N < 9), otherwise the controller would not activate the control action to load the machines due to the high penalty on their state x_3.

At the end, MPC results in an optimization problem with a set of linear constraints, a linear objective function and with real and integer decision variables. These types of optimization problems are known as Mixed Integer Linear Programming (MILP) [24] problems. They are much more difficult problems to solve than the classical ones because, for each of the possible combinations of the discrete decision variables, a LP

problem (with the remaining continuous decision variables) should be solved. Due to the high complexity of this mathematical problem it has been necessary using an optimization tool [15]. This tool is not able to understand the problem directly from the model of the system, so it has been necessary rewrite the problem in suitable form.

2.3 MILP Problem Derivation

The goal is to formulate explicitly the MILP problem starting from the dynamic model of the system. This means that starting from the MLD matrices, appropriately manipulated, we derive the matrices of the equations and inequalities representing the constraints over the prediction horizon (N) under which the MPC problem must be solved. It is possible to define the vectors:

$$X(k)=\begin{bmatrix}x(k+1)\\x(k+2)\\...\\x(k+N)\end{bmatrix}_{(n_x*N),1} \quad Y(k)=\begin{bmatrix}y(k)\\y(k+1)\\...\\y(k+N-1)\end{bmatrix}_{(n_y*N),1} \quad U(k)=\begin{bmatrix}u(k)\\u(k+1)\\...\\u(k+N-1)\end{bmatrix}_{(n_u*N),1} \quad W(k)=\begin{bmatrix}w(k)\\w(k+1)\\...\\w(k+N-1)\end{bmatrix}_{(n_w*N),1}$$

Where n_x is the number of the state variables, n_u the number of the control actions, n_y the number of the outputs and n_w is the number of the auxiliary variables, e is the number of the constraints.

Starting from the original system, by recursion we obtain:

$$\begin{cases}X(k+1)=\tilde{A}X(k)+\tilde{B}_uU(k)+\tilde{B}_{aux}W(k)+\tilde{B}_{aff}\\Y(k)=\tilde{C}X(k)+\tilde{D}_uU(k)+\tilde{D}_{aux}W(k)+\tilde{D}_{aff}\end{cases}$$

Where:

$$\tilde{B}_{aux}=\begin{bmatrix}B_{aux} & 0 & 0 & .. & 0\\AB_{aux} & B_{aux} & 0 & .. & 0\\.. & .. & .. & .. & ..\\.. & .. & .. & B_{aux} & 0\\A^{n-1}B_{aux} & A^{n-2}B_{aux} & .. & AB_{aux} & B_{aux}\end{bmatrix}_{(n_x*N),(n_w*N)} \quad \tilde{B}_u=\begin{bmatrix}B_u & 0 & 0 & .. & 0\\AB_u & B_u & 0 & .. & 0\\.. & .. & .. & .. & ..\\.. & .. & .. & B_u & 0\\A^{n-1}B_u & A^{n-2}B_u & .. & AB_u & B_u\end{bmatrix}_{(n_x*N),(n_u*N)}$$

$$\tilde{D}_u=\begin{bmatrix}D_u & 0 & 0 & .. & 0\\CB_u & D_u & 0 & .. & 0\\CAB_u & CB_u & D_u & .. & 0\\.. & .. & .. & .. & 0\\CA^{n-2}B_u & CA^{n-1}B_u & .. & CB_u & D_u\end{bmatrix}_{(n_y*N),(n_w*N)} \quad \tilde{B}_{aff}=\begin{bmatrix}B_{aff}\\AB_{aff}+B_{aff}\\A^2B_{aff}+AB_{aff}+B_{aff}\\...\\A^{n-1}B_{aff}+A^{n-2}B_{aff}+..+B_{Aff}\end{bmatrix}_{(n_x*N),1}$$

$$\tilde{D}_{aff}=\begin{bmatrix}D_{aff}\\CB_{aff}+D_{aff}\\CAB_{aff}+CB_{aff}+D_{aff}\\...\\A^{n-2}B_{aff}+..+CB_{Aff}+D_{aff}\end{bmatrix}_{(n_y*N),1} \quad \tilde{A}=\begin{bmatrix}A\\A^2\\...\\A^n\end{bmatrix}_{(n_x*N),n_x} \quad \tilde{C}=\begin{bmatrix}C\\C^2\\...\\C^n\end{bmatrix}_{(n_y*N),n_x}$$

$$\tilde{D}_{aux} = \begin{bmatrix} D_{aux} & 0 & 0 & .. & 0 \\ CB_{aux} & D_{aux} & 0 & .. & 0 \\ CAB_{aux} & CB_{aux} & D_{aux} & .. & 0 \\ .. & .. & .. & .. & 0 \\ CA^{n-2}B_{aux} & CA^{n-1}B_{aux} & .. & CB_{aux} & D_{aux} \end{bmatrix}_{(n_y*N),(n_w*N)}$$

The same holds for the matrices of the constraints, that are rewritten as:

$$\tilde{E}_x\tilde{X}(k) + \tilde{E}_u U(k) + \tilde{E}_{aux} W(k) \leq \tilde{E}_{aff}$$

Where:

$$\tilde{E}_x = \begin{bmatrix} \begin{bmatrix} E_x & \cdots & 0_{e,n_x} \\ \vdots & \ddots & \vdots \\ 0_{e,n_x} & \cdots & E_x \end{bmatrix} & \begin{bmatrix} 0_{e,n_x} \\ \vdots \\ 0_{e,n_x} \end{bmatrix} \end{bmatrix} \quad \tilde{E}_u = \begin{bmatrix} \begin{bmatrix} 0_{e,n_u} \\ \vdots \\ 0_{e,n_u} \end{bmatrix} & \begin{bmatrix} E_u & \cdots & 0_{e,n_u} \\ \vdots & \ddots & \vdots \\ 0_{e,n_u} & \cdots & E_u \end{bmatrix} \end{bmatrix} \quad \tilde{E}_{aff_c} = \begin{bmatrix} E_{aff} \\ \vdots \\ E_{aff} \end{bmatrix}$$

$$\tilde{E}_{aux} = \begin{bmatrix} \begin{bmatrix} 0_{e,n_w} \\ \vdots \\ 0_{e,n_w} \end{bmatrix} & \begin{bmatrix} E_{aux} & \cdots & 0_{e,n_w} \\ \vdots & \ddots & \vdots \\ 0_{e,n_w} & \cdots & E_{aux} \end{bmatrix} \end{bmatrix} \qquad \tilde{X}(k) = \begin{bmatrix} x(k) \\ x(k+1) \\ \cdots \\ x(k+N-1) \end{bmatrix}_{nx*N,1}$$

To be resolvable from Cplex, the system must be written in the form:

$$\begin{aligned} &min\ (f \cdot x_{cplex}) \\ &s.t. \\ &\quad \begin{cases} A_{eq}\ x_{cplex} = b_{eq} \\ A_{ineq}\ x_{cplex} <= b_{ineq} \\ l_b <= x_{cplex} <= u_b \end{cases} \end{aligned}$$

where:

- A_{eq}, b_{eq} are respectively a matrix and a column vector related to the equalities of the MILP problem defined starting from the original model.
- A_{ineq}, b_{ineq} are respectively a matrix and a column vector related to the inequalities of the MILP problem defined starting from the original model.
- x_{cplex} is the vector resulting from the solution of the optimization problem
- f is a column vector containing the weights of all the variables of the system.
- l_b, u_b are two vectors containing the lower and upper bounds of the variables taken into account in the optimization problem

For what concerns f, l_b, u_b , they are all design parameters set in the modelling phase. For this reason, there is not need to determine them. On the contrary, it is necessary to define A_{eq}, b_{eq}, A_{ineq}, b_{ineq} as functions of the vector x_{cplex}.

Let's define the vector to be passed to Cplex as:

$$x_{cplex} = \begin{bmatrix} X(k) & Y(k) & W(k) & Y(k) \end{bmatrix}^T$$

Regarding the matrix of the equalities A_{eq}, it must contain both the dynamic and the output equations. Since the vector $X(k)$ starts from $x(k+1)$, it is needed to isolate the initial state $x(k)$ to be able to rewrite the system in the desired form explicating x_{cplex} as :

$$A_{eq}\ x_{cplex}\ =\ b_{eq}$$

So, it is rewritten as:

$$\begin{cases} \tilde{A}x(k) + \tilde{B}_{aff} = X(k) - \tilde{B}_u U(k) - \tilde{B}_{aux} W(k) \\ \tilde{C}x(k) + \tilde{D}_{aff} = Y(k) - \tilde{D}_u U(k) - \tilde{D}_{aux} W(k) \end{cases}$$

The expressions of A_{eq} and b_{eq} are:

$$A_{eq} = \begin{bmatrix} I & -\tilde{B}_u & -\tilde{B}_{aux} & 0 \\ 0 & -\tilde{D}_u & -\tilde{D}_{aux} & I \end{bmatrix} \quad b_{eq} = \begin{bmatrix} \tilde{A}x(k) + \tilde{B}_{aff} \\ \tilde{C}x(k) + \tilde{D}_{aff} \end{bmatrix}$$

About the inequality constraints, attention must be paid to the fact that every constraint is determined for $\tilde{X}(k)$ and not $X(k)$, like in the previous case. It means that these constraints are defined at time k also for the state, whereas the vector of the state is defined starting from $k+1$. So, the first element of the $\tilde{X}(k)$ must be eliminated to rewrite the system as:

$$A_{ineq}\ x_{cplex}\ \leq\ b_{ineq}$$

To do that sub-matrices of zero elements are added to the A_{ineq} to delete the multiplication between the matrix and $x(k)$. In this way, there is a subset of n_c constraints concerning the initial state and it is possible to write the expressions of A_{ineq} and b_{ineq}. The only "price to pay" for this operation is that by using $\tilde{X(k}$ to define the state, an element is lost. It has elements defined in the range from (k) to $(k+N-1)$ while the original state vector is defined from $(k+1)$ to $(k+N)$.
The expressions of A_{ineq} and b_{ineq} are:

$$A_{ineq} = \begin{bmatrix} \begin{bmatrix} 0_{e,(nx*N)} & \begin{bmatrix} E_u \end{bmatrix} & 0_{e,(nu*(N))} & \begin{bmatrix} E_{aux} \end{bmatrix} & 0_{e,(nw*(N))} & 0_{e,(ny*N)} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} E_{x_c} \end{bmatrix} & \begin{bmatrix} E_{u_c} \end{bmatrix} & \begin{bmatrix} E_{aux_c} \end{bmatrix} & 0_{e*(NRH-1),p*(NRH)} \end{bmatrix} \end{bmatrix}$$

$$b_{eq} = \begin{bmatrix} E_{aff} - E_x x(k) \\ \tilde{E}_{aff} \end{bmatrix}$$

The expressions obtained are used to calculate the matrices to be passed to the optimizer at each sampling step starting from the matrices of the original MLD system acquired by HYSDEL.

2.4 Controller Implementation

Once the expressions to derive the matrices of the MILP problem are computed, it is possible to invoke the solver directly from the DCPIP platform eliminating the contribution of Matlab. To make the platform able to implement the MPC algorithm, some libraries has been added to the working environment:

1. The C ++ language does not support by default a data structure to store matrices of the size of those that describe the model of the plant. For this reason has been necessary to take advantage of Armadillo [21], an Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments, that not only allows to easily perform computations on very large matrices but it also has a syntax very similar to the one of Matlab, which makes it very easy and intuitive to use.

2. In order to implement the advanced techniques of operational research needed to solve a MILP problem characterized by a large number of variables to consider in a small amount of time, it has been necessary to use IBM Cplex Optimizer [10] *v*12.61, that is a high-performance mathematical programming solver for linear programming, mixed integer programming and quadratic programming. The interaction with this software is handled using the C++ API provided by the software house.

In order to understand how it has been modified, it is important to know some basic concepts related to the object-oriented programming. The basic idea behind this kind of programming languages is the concept of the class. A C ++ class represents an abstract data type that can contain elements closely related to each other and share the same attributes and the same functions. In particular the characteristics of the class are called *attributes*, while its functions are called *methods*.
When one speaks about an object is relating to an instance of a class. Therefore, an object has the same properties of the class to which it belongs and can recall its functions. It is important to underline that the data structures associated to an object are kept in memory during the whole life of the same, unlike the local variables of the functions that are destroyed when the effect of the function is terminated. Moreover, when an object is created, the function that creates it can be customized to initialize its parameters as desired.
A very important feature of the object-oriented programming is the inheritance, i.e. the possibility for an object son to acquire the characteristics (attributes and member functions) of another object father.

The DCPIP platform consists of various classes, and it is structured as shown in figure 2.1. Its main class is the *Task Manager*, which initializes the other ones and handles all the operations. An explanation of the role of each class of the platform is presented below:

- Machine: It defines the data structure associated to the plant to be controlled. So, it defines parameters like the number of BZs or the number of control possible actions.

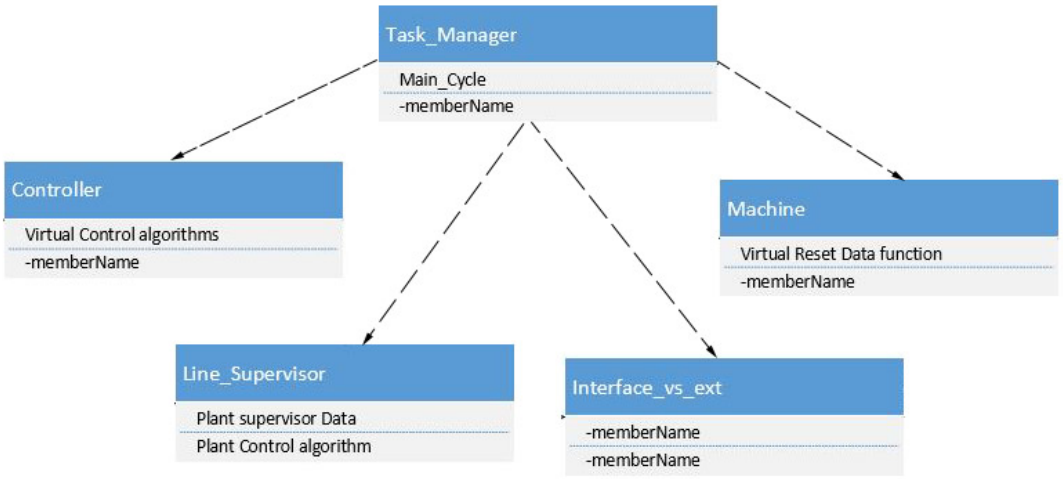

Figure 2.1: DCPIP class structure.

- Line Supervisor: it contains the variables and methods used to create the object machine data structure used to get information about the machine data and to read and write from/to the plant.
- Task Manager: it contains the main control cycle which scans the input acquisition, the control algorithms execution and the output updating.
- Controller: the module Controller is a class which implements different control methods. Since each machine is characterized by specific control functionalities then they will be implemented as specific classes. By means of the inheritance some common methods will be used from all the derived classes.
- Interface vs ext: it implements the communication among the Machine controllers, the Line Supervisor controller and the plant PLC. This class has got methods to execute the read/write input/output data, according to the different communication methods needed.

The DCPIP platform is a tool that allows to implement different types of control techniques, once they have been added as new classes of the project. So, the changes made to the platform mainly concern the addiction of a new class to it, son of the Controller one. The *deman control* class has been created, inheriting the attributes and the methods of its father. It contains the logic and data structures of the MPC algorithm.
For sake of clarity, its most important functions and attributes are briefly described:

Attributes:

- **Aeq, Aineq**. Matrices containing the inequalities (Aeq) and the disequations (Aineq) of the sytem to be passed to the math optimizer. They, **as all the unrolled matrices** (and the weight matrix **MP**) , are computed during the initialization phase and stored as a private member of the class. In this way we avoid to compute at every step the matrices (always the same) and it is possible to modify the constraints and the weights (and so the model of the plant) on-line.

- **NRH**. It defines the prediction horizon over which the MPC algorithm must be performed. It can be chosen in the initialization phase by means of a very simple GUI (graphic user interface).

Methods:

- **MPC_dynamic_matrices, MPC_constr_matrices, MPC_output_matrices**. These are the functions that compute the unrolled matrices starting from the MLD system.
- **CopyMat**. The cplex API does not recognize matrices as data structure, so it is fundamental to translate the data to pass to the optimizer in a suitable format.
- **SolveMQP**. It handles the communication with the Cplex software through its C++ API.

A UML representation of the Deman Control class is given in figure 2.3.

Behaviour of Deman Control

The DCPIP working life can be summarized into three phases:

- Initialization phase, in which it reads the type of the control strategy to be performed, then starts dynamically to build the correspondent data structures.
- Activity phase, in which it cyclically reads the inputs, processes the control algorithm and writes the outputs. The entire cycle is handled by the Task Manager, while the control strategy is performed by the specific controller chosen in the initialization phase.
- End phase, in which the data structures are destroyed.

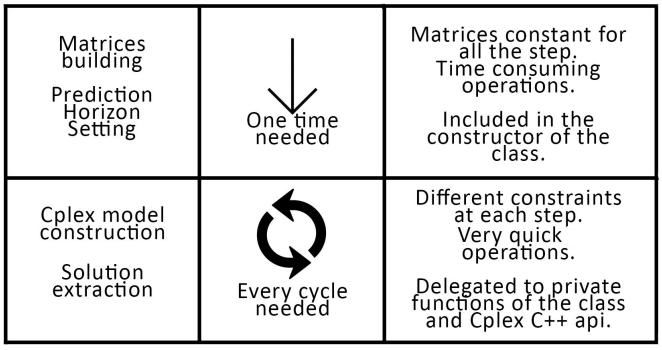

Figure 2.2: Deman Control tasks scheme.

In the specific case of this Thesis, the task manager invokes the main function of the Deman Control class (DCc) to solve the control problem. An object of the DCc is created by the Task Manager at the beginning of the program life, then its main function is called every step. Its operating principles (figure 2.2) can be described in this way:

- In the initialization phase, the matrices of the MLD model are acquired and used to compute A_{eq} and A_{ineq}. This operation is included in the builder of the class because treating these matrices entails a considerable loss of time due to their large dimensions.
- After that, at each cycle, the data structures are updated and the methods to compute the control actions are invoked. First of all, using the most recent state of the plant acquired, b_{eq} and b_{ineq} are computed. Differently from the other two, these change every time because they are strictly related to the distribution of the pallets over the network. However, it is important to underline how their dimensions are notably smaller than the ones in the previous case, and also the time required for the computations is smaller. In a second moment, A_{eq}, b_{eq}, A_{ineq} and b_{ineq} are translated into a format compatible with the Cplex environment that does not recognize a matrix as data structure. Through the API, these are transmitted to the solver and the solution is saved into a vector. At the end, the outputs are passed to ISaGRAF to be executed.

2.5 Cplex configuration

Cplex uses a branch and cut algorithm for the solution of MIP problems. IT consists of the application of a branch and bound search combined with the cutting planes method to tighten the linear programming relaxations. Setting up the work environment to obtain good performances has taken a long time because of the combinatorial nature of resolution method. In fact, Cplex has many parameters that allow users to customize the way in which the branch and bound algorithm operates and the user cannot realistically try all the possible combinations of settings. For this reason, IBM LOG provides an automatic tool that allows configuring the solver quickly for the specific system to deal with. It allows us to obtain a list of parameters to be changed to improve the performances of the solver starting from a model of the system to be optimized in '.lp' format, a proprietary one of Cplex. It must contain an example of our MILP problem indicating, for example, if it is a minimization or maximization problem, the number and the type of our variable or the constraints to be considered. However, the use of this tool did not prove to be useful and therefore some parameters had to be modified by trial, as indicated on the IBM website.
The customization of the parameters has been necessary not only to reduce the calculation time, much higher than the one obtained with the Matlab toolkit, of the solution but, above all, because the initial configuration of the solver sometimes provides unacceptable solutions for the control problem. It means that wrong actions could be passed to the system, that it is not acceptable in our context. For this reason, the boundaries relaxation has been forbidden, whereas to improve the computational performances the following parameters have been changed:

- ILOCPLEX::PARAM::MIP::STRATEGY::HEURISTICFREQ, which defines how often to apply the periodic heuristic.
- ILOCPLEX::PARAM::MIP::STRATEGY::VARIABLESELECT, which establishes the

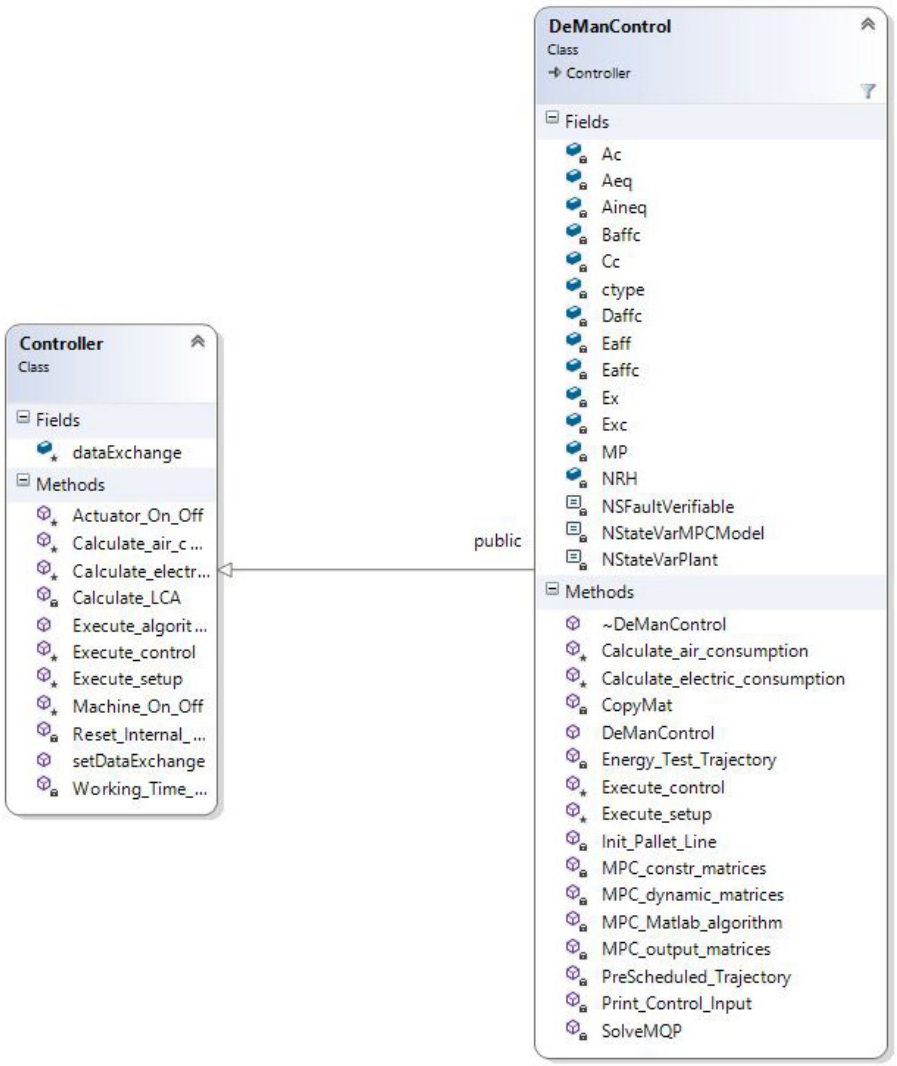

Figure 2.3: UML scheme

rule for selecting the branching variable at the node which has been chosen for branching

- ILOCPLEX::PARAM::MIP::LIMITS::CUTPASSES, which sets the upper limit on the number of cutting plane iterations done solving the root node.
- ILOCPLEX::PARAM::PREPROCESSING::BOUNDSTRENGTH, which decides whether to apply bound strengthening in mixed integer programs. Bound consolidation tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from the analysis.

A problem has been detected concerning the saturation of the RAM that has not been solved. After each call to Cplex, the volatile memory is not cleaned up by the data of the previous call. So, after a certain number of optimizations, the work-station is slowed down until it gets stuck. Furthermore, part of the memory must be reserved for other programs currently running on the same machine.

The solution chosen to solve the problem was not found to be effective. In fact, it was decided to limit the number of threads and the percentage of memory available to the optimizer, but after prolonged usage of the plant, there were problems of slowing down on the workstation and consequently on the plant.

The current configuration leads to improvements in the optimizer performance compare to those obtained with Matlab. However, it is not optimized yet. Indeed, this mathematical problem would require more in-depth skills on advanced operational research topics and more time to be dealt with appropriately.

Chapter 3

Improvements of the Control Algorithm

This chapter presents the work done to improve the performance of the control system regarding the reduction of the time needed to calculate the control actions and the increase of the productivity of the controlled plant.

In order to understand the advantages of the various techniques developed, a simulation script has been defined in Matlab. It emulates the behaviour of the plant without communicating with the lower level. More precisely, the MLD model and the control algorithm have been implemented in such a way that after an initial phase in which the matrices of the system are built, the optimization problem is solved, cyclically, exploiting at each step the prediction of the state obtained at the previous cycle (the initial state at the initial time instant is given). In this way, it is possible to simulate the evolution of the plant without resorting to simulators. All the simulations have been verified on the real plant with a much shorter simulation interval. However, note that the machining operation times used for each machine Mi, that is parametrized in the MLD model in terms of number of events k, are different from the real one that are basically unpredictable, so the result of the simulations aiming to measure the number of pallets worked are not comparable with the one performed on the real plant.
The behaviour of the machines is simulated so that the new target assigned to each worked pallet is chosen randomly between the ones reachable from that machine (for example, from M1 there is only one destination possible, while from M2 three targets are available depending on the outcome of the test executed the pallet). Direct consequence of this is the fact that the evolutions of the system for each simulation are different each other. For this reason, the only possible way to perform a comparison is to reason in terms of average times of execution considering a long interval of optimization.

For each implemented improvement, two experiments have been carried out[1]. In the first one, different tests, under different starting conditions and considering 100 simulation steps, have been performed in order to compute the corresponding average of the

[1]Simulations run on a computer with Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz, 16.0 GB of installed RAM, system type 64-bit operating system, x64-based processor, Windows 8.1 Pro., MATLAB R17a, CPLEX R12.6 (settings: Parallelmode = 0, Threads = 0).

computational times. Both the number of pallets loaded on the transport line, at time instant $k = 0$, and the prediction horizon have been varied to understand the behaviour of the different control strategies starting from different initial states. In the second experiment, considering the initial condition of five pallets and NRH equal to 6, the number of machined pallet within 1000 simulations steps has been estimated.
To allow the comparison of the results obtained with the different solutions, in all the tests performed the initial state (at the instant $k = 0$) is the following:

pallet 1 in N28 with Target 4;
pallet 2 in N29 with Target 2;
pallet 3 in N30 with Target 3;
pallet 4 in N31 with Target 5;
pallet 5 in N16 with Target 2;
pallet 6 in N19 with Target 3;
pallet 7 in N23 with Target 5.

Since in some tests it had been necessary to consider a starting condition in which a smaller number of pallets were on the transport line, they have been taken into account as needed in numerical order. So, depending on the number n_p of pallets they were placed in position described above for the pallet from 1 to n_p.

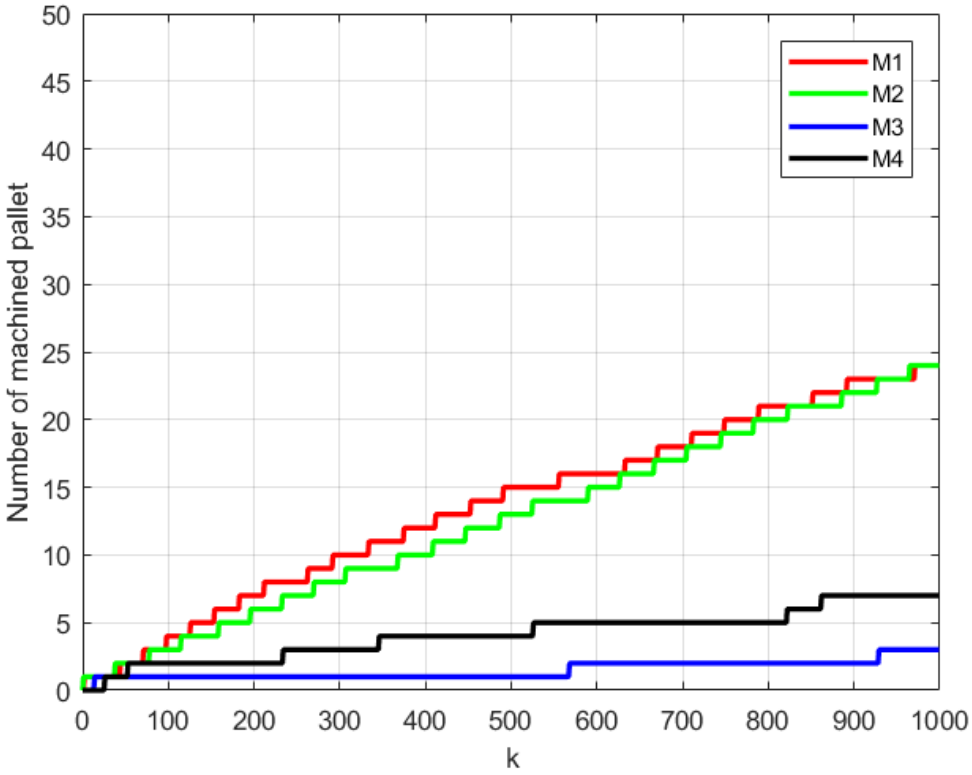

Figure 3.1: Machined pallets for time unit (Basic Controller).

The same experiments has been performed on the basic MPC controller developed in the Chapter 2. The results are reported in Table 3.1 and in figure 3.1. These results are

Table 3.1: Average computational time [s] per optimization step with standard MPC

Num. Pallets	$N_{RH} = 5$	$N_{RH} = 6$	$N_{RH} = 7$
3	0.32	0.87	2.62
4	0.47	1.22	3.41
5	0.59	1.29	6.70
6	1.34	6.30	56.35
7	4.17	>100.00	>100.00

taken as reference and used as terms of comparison to understand the actual usefulness of the improvements introduced. In fact, the goal of the first test is to understand if there is a significant saving of time due to the improvements developed during the Thesis work, while the second one aims to understand if the modifications lead to a significant change in the number of pallets machined.

3.1 Hiding Optimization Process

The original controller has been designed to reserve different time slots for calculating the optimal solution and executing the commands reserved for the PLCs. It means that, at each time step, the optimal solution is derived, the control actions to be executed are passed to the PLCs and, at the end of the last one of these sequences, the high control reads the new state of the plant and calculates the next move. This approach introduces a useless delay from the moment that these actions being carried out by two distinct players. So, it was decided to rewrite the controller structure in such a way to carry out the calculations and the movements in parallel.
The idea behind the first feature introduced is based on the fact that the DCPIP can compute the solutions while the LLC is working, with an evident saving of time and without significant drawbacks. To do this, the properties of the MPC are exploited. In fact, one of the advantages of Model Predictive Control is that it provides information on the entire control horizon chosen. The solution taken from the solver is the vector:

$$x_{cplex} = [\ X(k)\ U(K)\ Y(K)\ W(K)\ ]^T$$

$$X(k) = [\ x(k+1)\ x(k+2)\ x(k+2)\ ..\ x(k+N-1)\ ]$$

Thus, at each iteration k, when Cplex is called to determine the control actions, the predicted state $x(k+1)$ is also received. It is possible to use the predicted state to start a further optimization while the system is in motion, instead of waiting and read it from the LLCS.
At the generic instant k, the new sequence of operations is:

1. The controller computes the optimal control input $u^o(k)$ and predicts the future state of the plant $x_p(k+1)$.
2. The lower layer of the control structure starts running to implement this input and to move the pallets, while, at the same time, a new optimization process can run based on $x_p(k+1)$ to compute the new optimal control input $u^o(k+1)$ in advance.
3. As the pallet movements due to $u^o(k)$ are completed, the predicted state $x_p(k+1)$ is compared to $x_i(k+1)$ given by the LLCS.

4. If they are equal, the computed optimal control $u^o(k+1)$ is applied.
   Otherwise, the optimization problem is restarted by setting $x_i(k+1)$ as the new initial state.

There are two main effects of this implementation: there is a substantial saving of times, and the pallets move on the network more fluidly. Indeed, thanks to the Cplex configuration described in the previous chapter, very often the resolution of the optimal problem requires less time than that needed for the plant to move the pallets. Therefore as soon as one control action ends, another one starts.
As an example of the advantages given by this feature, consider the case in which six pallets are moved over the transport line and the prediction horizon $N = 6$. The average time requested to solve the optimization problem is very similar to the one needed for performing the movements of the pallets. In figure 3.2, the difference between the new and the old implementations is shown and, in particular, the times characterizing the communication protocol are compared. Whit the red lines it is represented the time needed to compute the control actions, whit the blue lines the time in which the actuators implement them and with the green ones the time required by the network to propagate the information from PCs to the plant and vice-versa.
To implement this feature, it has been necessary to work on the main function of the *Deman Control* class. A vector has been obtained to store the state of the plant between two consecutive iterations it has been defined as a private member of the class and the structure of its main function has been modified in order to work as required by means of the pseudo-code:

Main control cycle
If low level control is still {
state= readState();
if (alreadyComputed *equal to* false) OR (state *different from* stateSav){
[solution,stateSav]=computeSolution(state); }
passToLLC(solution);
alreadyComputed = FALSE;
} **Else** {
solution= computeSolution(stateSav);
alreadyComputed = TRUE; }
End control cycle

The code reported is very simplified, but it is useful to understand the idea behind the changes implemented. Since only the helpful information is extracted from the solution returned from Cplex, the method delegated to compute the control actions is modified to store the vector of the state corresponding to $x(k+1)$ (*stateSav*). It is passed to the same function while the LLC is acting instead of waiting and reading it from the plant. With this mechanism, especially with a limited number of pallets and/or a small predictive horizon, the state read by the plant is very rarely used for the optimization process because the time necessary to do the calculations hardly exceeds the time to move the pallets. It could become a problem in the case in which an external input acts on the plant modifying its state. Speaking of external information is not different to speak of signals from the machines of the plant with which they describe the progress of their activity. The exogenous factors are unpredictable because they do not depend on the MPC algorithm (it is not possible to know the time requested to a machine to perform

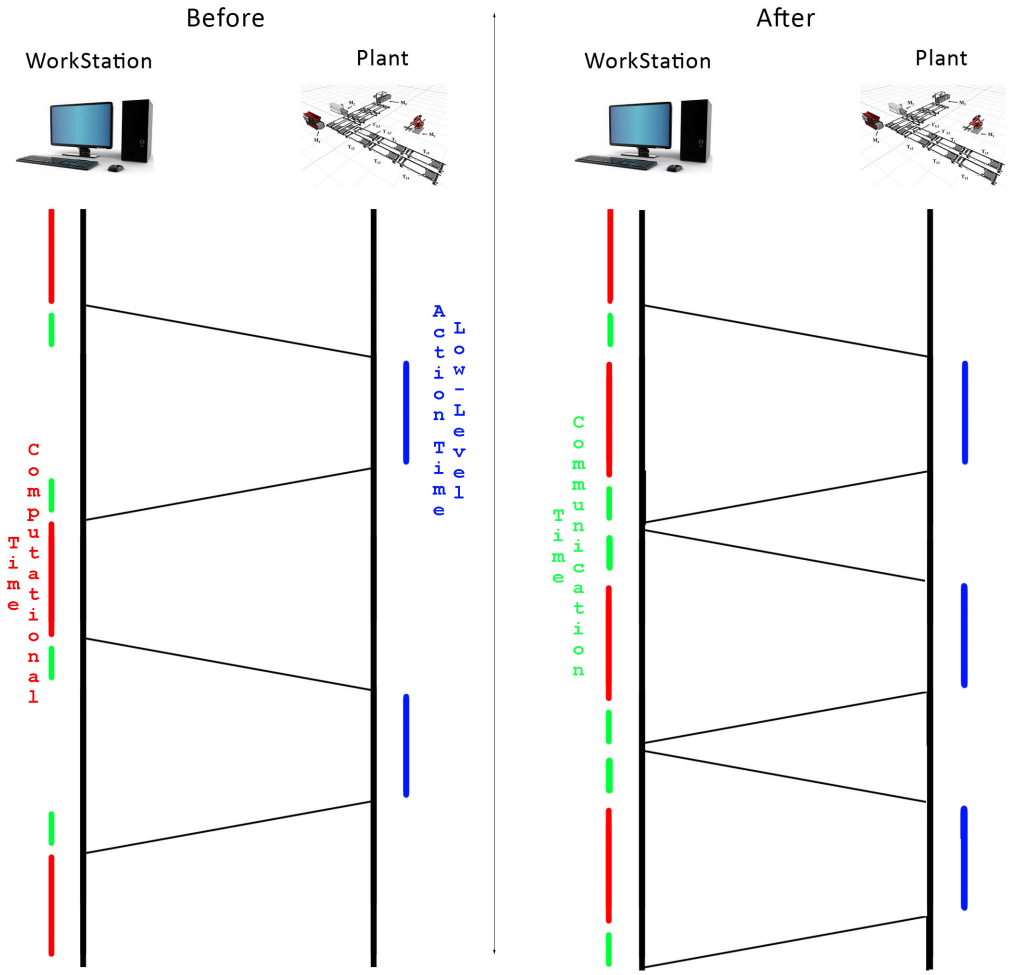

Figure 3.2: Communication protocol comparison

its operation since it depends from the specific board and fault detected on it). For this reason, they could be ignored for a very long time by the controller, with significant performance degradation. Indeed, when a machine ends up working, it sends a message to the line supervisor that consequently changes the state of the machine in its vector of the state, that is the one passed to the high-control level to compute the next control actions. So, if the controller never uses the state of the plant passed from the LLCS, and more specifically the machines state, it always continues to see the machine, that has finished to work, in the state x_2, i.e. in the working phase. The machined pallet is never loaded on the transport line and the production has a drastic decay of the performances. Then, it is essential for the controller to compare its prediction of the state with the one acquired by the LLCS and, if they are different, to drop the control actions computed with the wrong state and call Cplex to obtain the new vector to be passed to the PLCs.

Table 3.2: (A) Total single step time [s] *before-after* the hiding implementation; (B) Percentage of the time saved

Num. Pallets	$N_{RH} = 5$	$N_{RH} = 6$	$N_{RH} = 7$
3	5.32 - 5.00	5.87 - 5.00	7.62 - 5.00
4	5.47 - 5.00	6.22 - 5.00	8.41 - 5.00
5	5.59 - 5.00	6.29 - 5.00	11.70 - 6.70
6	6.34 - 5.00	11.3 - 6.30	61.35 - 56.35
7	9.17 - 5.00	>100.00	>100.00

(A)

Num. Pallets	$N_{RH} = 5$	$N_{RH} = 6$	$N_{RH} = 7$
3	6.00	14.80	21.30
4	8.60	19.60	40.50
5	10.50	20.50	42.70
6	21.10	44.20	8.14
7	45.40	< 0.01	< 0.01

(B)

To understand the effects of performing the two operations in parallel, it is possible to define the time requested to complete a step (total single step time) of the plant activity, before implementing it, as:

$$T_{tsst} = T_m + T_c + 2 \cdot T_p$$

Where:

- T_p, it the time required to exchange information between the HLC and the LLC. Since it depends on the network structure, and it is not the object of this Thesis, it will be neglected in the following analysis. However, it is relatively small with respect to the others.
- T_c, it is the average time required by the calculator to complete control actions
- T_m, it is the average time required by the LLC to complete a control action. It has been experimentally measured on the real plant and it is about *five* seconds. Obviously, this time does not change depending on the type of control, or on the chosen prediction horizon, since it depends on the settings of the motors (and they cannot be modified).

At the same time we can express the total single step time after the parallelization:

$$T_{st} = \begin{cases} T_m & if\ T_c \leq T_m \\ T_c & if\ T_c > T_m \end{cases}$$

So, it is possible to define the saving of time as the difference between these two measures:

$$S_t = (T_m + T_c) - max\{T_m, T_c\}$$
$$S_{t\%} = \frac{S_t}{T_{tsst}} \cdot 100$$

Taking as reference the times obtained from the simulation of the basic MPC, a qualitative estimation of the time saved has been done. The data are reported in Table 3.2.

Generally speaking, it is possible to state that with this technique significant results (>10%) can be obtained in terms of time saved when the resolution times of the optimization problem belong to the interval [0.6 , 55] seconds. Otherwise, the advantages is minimum (case with seven pallets and NRH bigger than five, or six pallets and NRH equal to 6). Note that for a prediction horizon N smaller than 6, the optimization process is completely hidden and when a control action ends another one starts.
Assuming to re-run the second experiment with the same MPC algorithm, the total processing time of the plant operations would change from 105 minutes to 83, that is the minimum time required by the low-level system to implement the pallets movements.

3.2 Tunnel Implementation

There are some paths in the plant, called tunnels, in which there is no routing decision to be taken because the pallets have only one way to follow. The controller, at the beginning of this Thesis, was not able to individuate these situations, so it wasted time at every step to find the control actions to perform even in the case in which there was only feasible possible solution. This could be a severe problem as the number of pallets on the transport line increases, limiting the scalability of our control solution considerably. For this reason, a mechanism to force the movements of the pallets when they are in these areas without wasting computational power has been developed.

First of all, the plant topology has been studied to find the buffer zones suitable to form the tunnels. The analysis has been carried out systematically, and once the rules indicated below were defined, the graph (see figure 1.4) has been analysed applying those to each node. The rules are:

- If a BZ has only one control action as input and only one as output, it can be added to the tunnel.
- If a BZ has more than one control actions as input, it can not be added to the tunnel.
- If a BZ has more than one control actions as output, it can not be added unless it is the last BZ of the tunnel.
- A tunnel cannot be composed by only one BZ.

These four simple rules allow to define a logical structure, relying on which it is possible to set heuristic rules to move the pallets without wasting time in the optimization process. In general, a tunnel is composed by BZ that have only one control action to move-in and one to move-out. However, a particular rule applies to the last element: indeed, it could have more than one outputs because the controller does not hide the pallet present in that position. This is because it is supposed that the BZ adjacent to the last element has more than one input and so it is necessary to make a reasoned choice to avoid crushing between pallets.

An example of acceptable and non acceptable BZ to be inserted in a tunnel is shown in figure 3.3.

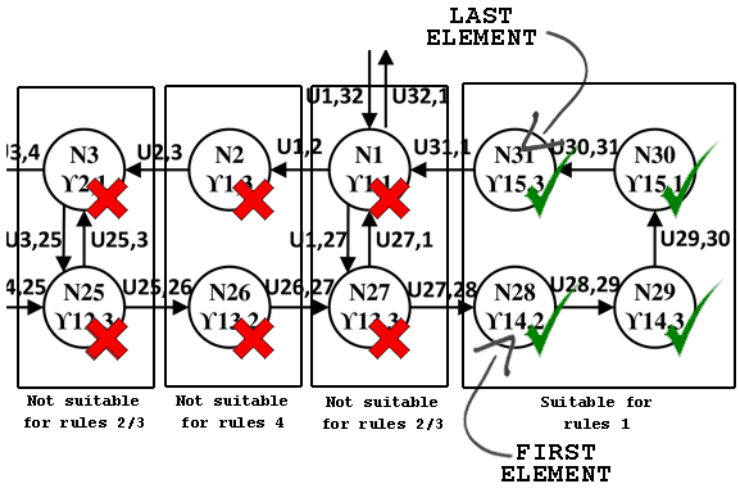

Figure 3.3: Example of admissible buffer zones.

Following these rules, four groups have been individuated for our configuration: a set of four nodes composed by 28-29-30-31 in the graph representation of figure 1.4, two formed by three nodes (4-5-6 and 8-9-10) and a smaller one composed by only the nodes 20-21.
This mechanism has been implemented in such a way as to modify the information exchanged between the optimizer and the control platform, acting on the state vector and on the vector x_{cplex} given back as the solution of Cplex. It recalls the "man-in-the-middle attack", typical problem of the computer security where the attacker secretly alters the communication between two parties who believe they are directly communicating with each other. In this case, the "attacker" is a specific function, *ComputeXnTarget*, of the DemanControl class and the two parties are Cplex and the function dedicated to handling the communication with it, called *SolveMQP*. In this way, it was possible to introduce this change without modifying the basic operation of the DCPIP platform.
Specifically, *ComputeXnTarget* removes the elements corresponding to the pallets in the tunnels from the vector of the state before passing it to the optimizer. After that a solution is given back, it adjusts the vector of the inputs (u_{ij}) by adding the control actions necessary to move them since they have been not considered from the coordinator in its computations.
Particular attention must be given to the fact that in this phase, not just the elements indicating the target associated to each BZ (Γ_i) is hidden, but their counters (η_i) too. The management of the counter (η_i) associated with each pallet to avoid starvation is not trivial for mainly two reasons. Firstly, since it is not possible to read them from the plant, they must be stored at each step and merged to the subset of the state readable at following one. Secondarily, there are several rules to follow to update correctly them, since it is needed translating from constraints of the MLD model to C++ code.

To better describe how this function works, its structure is simplified, divided into the three parts (preliminary operations on the state, a posteriori operations on the state and control actions) and reported as pseudo-code:

Pseudo-code of preliminary state modification

```
For each tunnel:
for (int i= 0; i < tunnelLength; i++) {
supportVectorState(i) = vectorOfState(numberOfNode+i);
supportVectorCounter(i) = vectorOfCounter(numberOfNode+i);
if (i < tunnelLength - 1) {
vectorOfState(numberOfNode+i) = 0;
vectorOfCounter(numberOfNode+i) = 0; }
}
```

The information characterizing the network is saved in two vectors, one for Γ_i and one for η_i, for each tunnel before the removal in order to be able to correctly update the vectors later. These $n = 2 \cdot n_{tunnels}$ vectors are volatile data structures that are created and immediately destroyed at each call to the solver, with enormous benefits from the computational point of view with respect to solutions in which permanent data structure are used. As it could be easily seen from the pseudo-code, the last element of the vector is not overwritten. It is copied to the support vector to implement the internal logic of the tunnel, but at the same time, it is not hidden to the coordinator.
Once the preliminary operations are done, the modified state is passed to the function *Solve MQP*, devoted to communicate with the solver, that gives back the vector of the predicted state, the vector of the predicted counters, and the one contains the control actions. These vectors need to be modified.

Pseudo-code of a posteriori operations on control actions

```
if (controlAction that moves pallet in last position has been chosen by solver) {
supportVectorState(tunnelLenght-1) = 0; }
for (int i= tunnelLenght; i > 0; i- -) {                    (all the elements are examined in pairs)
if ((supportVectorState(i) equal to 0) AND (supportVectorState(i-1) different from 0)){
controlAction(startIndex+i) = 1; }
}
```

In this way, the missing control actions are added to the corresponding vector. The support vector is inspected with a top-down approach. Starting from the last element hidden, a check on the following is done and if the position is free, the control action that moves the pallets from the position $i-1$ to the position i is set to 1. Note that, the position are not updated after this control, because due to the configuration of the system the following rule must be followed:
Two pallets lying on two adjacent BZs in a tunnel cannot be moved at the same time step.
This rule had to be introduced because in some points the mechanical structure of the module limits the way of operating. The constraints implementing it are obviously included in the MLD model, and concern the specific modules in which it is not possible to move two pallets at the same time, whereas in the tunnels it is implemented as a general law. Although this slightly reduces performance, it helps to define tunnels as a mechanism that can be easily applicable to a wide range of routing problem based on mathematical graph representation without loss of generality. Once the vector of the control actions has been adjusted, using it, it is possible to modify the vector of the counters.

Pseudo-code of a posteriori operations on counters

```
if (controlAction to move the pallet from the i-1 position to i one is set to 1) {
supportVectorState(i)= supportVectorState(i-1);
supportVectorState(i-1)= 0;
updateCounter(supportVectorCounter, VectorCounter, 'move', supportVectorState(i));
} else {
updateCounter(supportVectorCounter, VectorCounter, 'still', supportVectorState(i-1));
}
```

Once the control actions have been determined, it is possible to establish the position of the pallets at the next step. Considering the ones able to move the pallet between the BZs composing the tunnels, a check to understand what control actions are activated is done. If the control action $U_{i-1,i}$ is activated, the support vector of the state is updated, copying the element in $i-1$ to the i position, and the element in the position $i-1$ is set to zero. As far as the counters are concerned, the elements of the support counters vector are updated even if the corresponding pallets remains still. Considering the node i, at each iteration, η_i must be increased by one if Γ_i is bigger than one. So, once the new position of the pallets at the step k+1 has been determined, it is possible update the counter vector by analysing the target corresponding to each BZs.
Moreover, at this point, another check on the first element of the tunnel is needed. In fact, it is possible that the optimizer pushes a pallet into the tunnel even if the entrance of this is actually occupied by a hidden pallet. In this case, since a wrong control action has been activated, it must be cancelled. Setting it equal to zero is not enough, as the prediction of the state (and used to calculate the next step) has been calculated taking into account an action that will not actually be done. For this the state vector and the corresponding counter must be updated too.

At the end of the operations, the vector x_{cplex} is re-built merging the support vectors and the modified control actions vector together. It is then passed to *SolveMQP* which extracts the necessary information as before without being aware of the tunneling process.

To test the effectiveness of this technique, the same experiments performed for the basic algorithm has been done. Regarding the first experiment, the data concerning the computation times are shown in Table 3.3. In this case, there is a reduction in terms of computation time taking as reference the basic controller, which is more evident as the number of pallets on the system and the prediction horizon increase. From the result of the second test, figure 3.4, it is possible to assume as expected the performances in terms of pallets machined for unit of time are more or less the same with respect to the basic controller case (shown in figure 3.1).

Table 3.3: Average computational time [s] per optimization step adopting tunnels (A); Percentage of the time saved with respect to the basic controller (B)

Num. Pallets	$N_{RH} = 5$	$N_{RH} = 6$	$N_{RH} = 7$
3	0.23	0.29	0.61
4	0.36	0.98	1.77
5	0.38	0.77	1.87
6	0.49	3.44	8.04
7	1.65	4.98	18.49

(A)

Num. Pallets	$N_{RH} = 5$	$N_{RH} = 6$	$N_{RH} = 7$
3	28.2	66.6	76.7
4	23.4	19.6	48.1
5	35.5	40.3	72.1
6	63.4	45.3	85.7
7	60.5	>96	>96

(B)

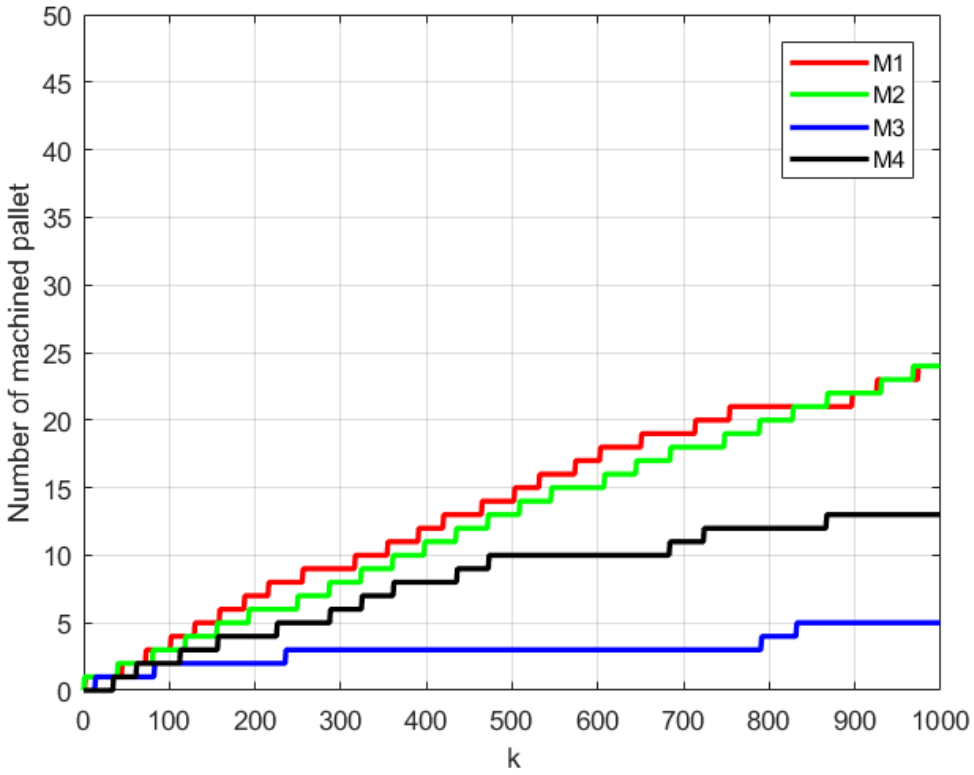

Figure 3.4: Machined pallets for time unit (Tunnel).

3.3 Control Horizon

The definition of tunnels proved to be very effective in terms of reducing the required computing power and therefore of computational time reduction. On the other hand, it is a technique applicable to control problems based on mathematical models such as oriented graphs. For this reason, it has been tried to reduce the time of computation exploiting a technique applicable to any MPC controller: the control horizon. Note that, actually, the two techniques are compatible since the control horizon is a modification of the model predictive control algorithm, while the tunnels method is an improvement to the high-level control system that does not regard the type of control strategy implemented.

Typically, a large value of N is often chosen to include in the prediction horizon all the main process dynamics. The problem is that the more the prediction horizon increases the more the number of optimization variables is large and the computational load of the optimization problem to be solved on-line is heavy. For this reason, in order to lighten up the work of the solver, it is possible to define a new interval $0 < N_u < N$, called control horizon, in which additional constraints are considered [23]. The variables computed in the minimization of the cost function are the commands $u_{i,j}(k+l)$ over the control horizon, while some Heuristic Rules are used from $k + N_u$ onwards. In this way the number of optimization variable to be computed at each k is reduced without shortening the prediction horizon N. Choosing $N_u < N$ leads to a suboptimal solution with respect to the stated optimization problem, but it allows to apply the proposed technique in problems otherwise intractable due to their inherent computational complexity.
In our system, the HR have been developed by forcing each pallet that comes out of any machine to follow a prescribed path not interlaced with the others and depending on the final target machine of the pallet itself [4]. From the practical point of view, with this technique new constraints are added to the MILP problem derived by the MPC algorithm by concatenating A_{eq} and A_{ineq} of the basic controller with A_{eq} and A_{ineq} derived from another MLD model. In this model, that is the same of the other except for some additional constraints, some precautions are taken into account to reduce the complexity of the optimization problem, for example some control actions, that create alternative routes with respect to the shortest one, are imposed to 0.
To implement this control technique, it is possible to use the same function for the basic controller both for the first interval (from 0 to N_u) and the second interval (from $N_u + 1$ to N) of the horizon and then concatenate the resulting matrices. The same functions for handling the communication with Cplex and the LLC can be used.

Even in this case, the algorithm has been tested with the two experiments. For both of them, the considered horizon are: $N_u = 2$ and $N = 6$.
The data collected are shown in table 3.4. Even in this case, it has been possible to notice a significant reduction of the computational time at the price of an acceptable deterioration of the performances (as it can be seen from figure 3.5) due to the fact that in the control horizon interval, it is not possible to use all of the available degrees of freedom, due to the presence of feasible paths that are not considered in the HRs.

Table 3.4: Average computational time [s] per optimization step with MPC with control horizon (A); Percentage of the time saved with respect to the basic controller (B).

Num. Pallets	$N_{RH} = 5$	$N_{RH} = 6$	$N_{RH} = 7$
3	0.28	0.63	1.45
4	0.39	0.91	2.12
5	0.43	1.12	4.46
6	1.12	3.66	19.13
7	2.54	>100.00	>100.00

(A)

Num. Pallets	$N_{RH} = 5$	$N_{RH} = 6$	$N_{RH} = 7$
3	13.18	27.92	44.69
4	17.26	25.69	33.78
5	26.80	13.08	33.42
6	16.03	41.08	66.05
7	38.98	<0.01	<0.01

(B)

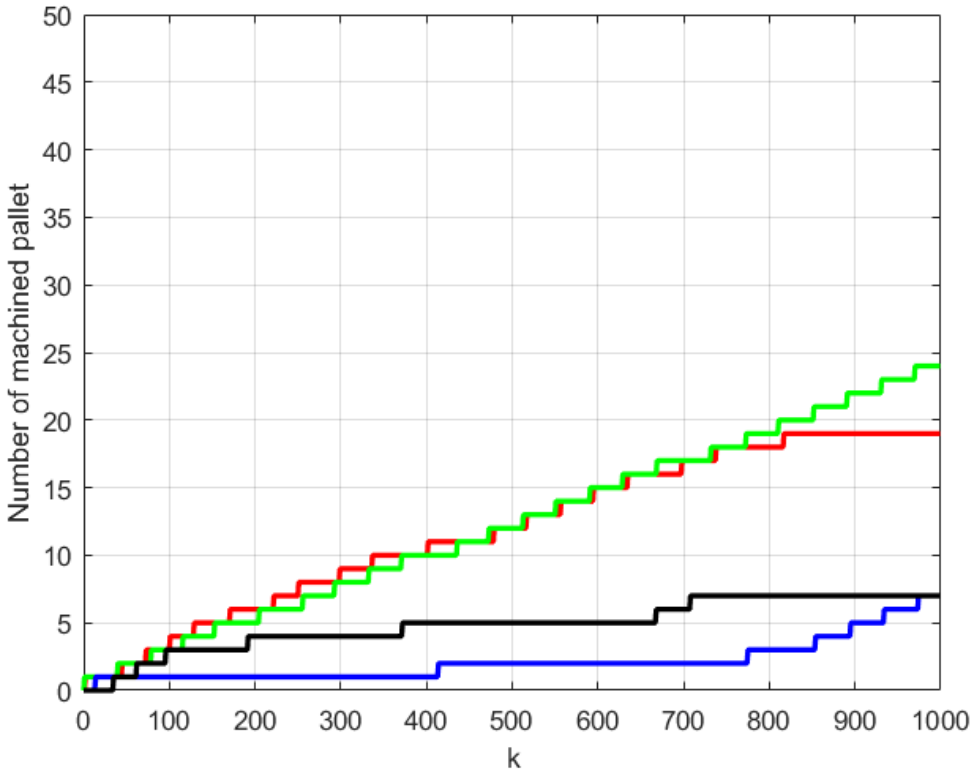

Figure 3.5: Machined pallets for time unit (Control Horizon).

3.4 Attraction Zone

When a pallet come out of the destruction cell it is labelled as empty and its target value is changed from four (destruction cell) to five (no target) since when the pallet exits from the station is empty and no target is assigned to it, waiting for the robot cell to call it back. At the beginning of this Thesis, the MPC model was made in such a way that these empty pallets were not moved by the coordinator, if not in case they were an obstacle for the path of the others. When the robot cell needed a pallet to charge an electronic board on the network, it sent a request to the line supervisor (ISaGRAF environment) that selected one of the empty ones on the transport line and, by changing its target, moved it to this area.
This choice considerably reduces the effectiveness of the system. It was very likely that the pallets remained in an area of the transport line far away from the robot cell until they are called. This obviously introduced a significant drop in performance due to the downtime of that cell waiting for pallet. Indeed, if a pallet starts as soon as possible to approach the load station, the time that passes between the recall of the pallet and its actual arrival is reduced to a minimum.

The fact that a pallet remains stationary if empty is due to the setting of some weight indices. In particular, the responsible are the parameters about the minimum cost (distance from the target) associated to each buffer zone. Let's take as example the parameters defined for the BZ number one of the first module ($\gamma_{1,1}$) in the HYSDEL model:

```
REAL C_bz1_Tp0 = 0; /* Target free */
REAL C_bz1_Tp31 = 1;
REAL C_bz1_Tp32 = 11;
REAL C_bz1_Tp33 = 8;
REAL C_bz1_Tp34 = 11;
```

The first parameter (that is called $q_{\Gamma_i,5}$ in the MLD model) is the one defining the weight associated to a pallet with target one ($\Gamma_i = 5$) in that buffer zone. It is set to zero and then the correspondent variable is no more taken into account in the cost function. The same is for the other BZs. The idea is then to modify this parameter for each node of the network in such a way that the empty pallets are directed in the direction of an area around the robot cell. To do this, it is important to be careful not to modify the behaviour of the system. In fact, the optimizer must always give priority to full pallets moving around the network.
So, since the minimum weight set for the other parameters of the controller is equal to 1 and the still pallets must be moved only in the case of their movements not influence other choices, the parameters for the empty pallet of each BZ can be set equal a number between 0 and 1. In this way, this kind of action has the lowest priority for the controller.

The area selected as the most suited to host the empty pallets, is the one composed by the nodes 28-29-30-31 (see figure 3.6). It has been chosen because it is a redundant path of the transport line and it is very close to the robot cell. The fact that it is redundant is fundamental. In this way, the pallets with higher priority can reach the load station by means of the control action $U_{27,1}$ without wasting time waiting the the empty pallets are moved to allow their passage. So, the node farthest from the robot cell has been

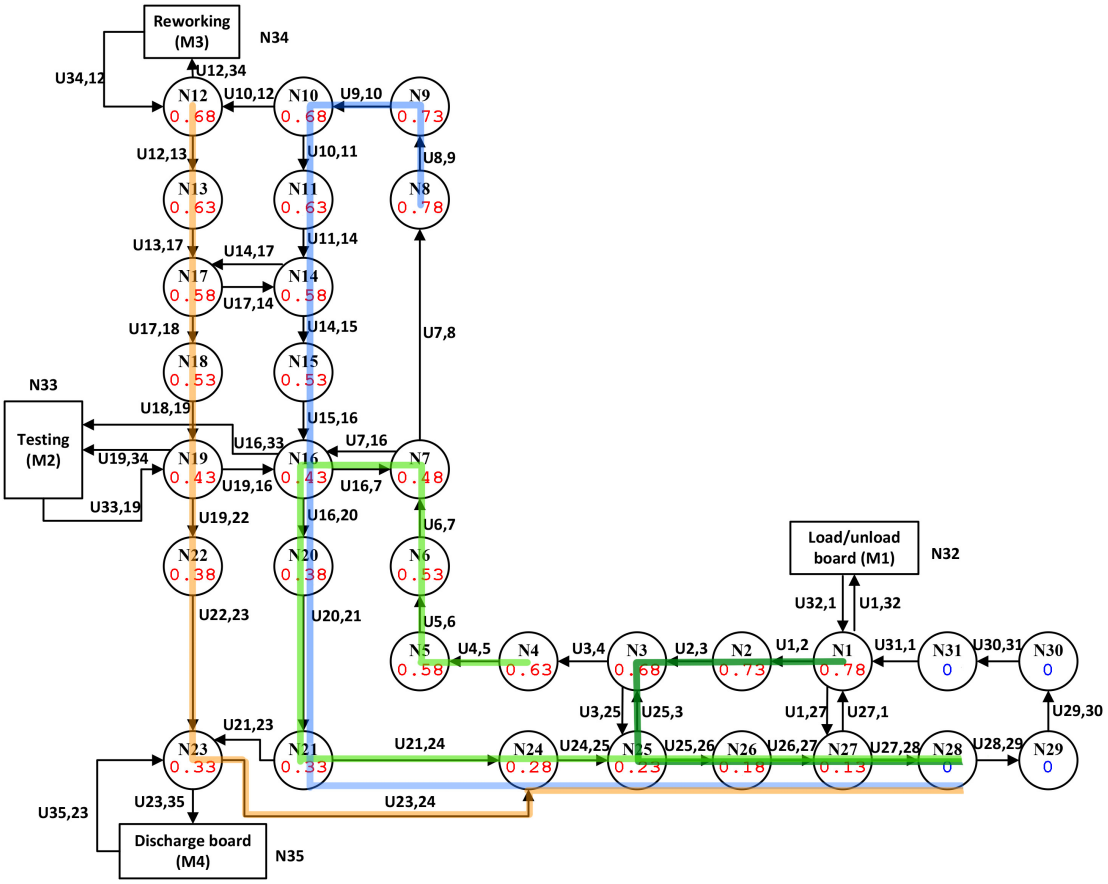

Figure 3.6: Attraction Zone Routing.

individuated. For the farthest one, it is meant the one from which a pallet needs more optimization steps to reach the machine M_1. It has been found in N8 (figure 1.4). The parameter $q_{\Gamma_i,5}$ (representing the weight of the pallets with target 5 in the cost function) associated with N8 is set to 0.78. Then starting from that node, the other nodes of the transport line are weighed in such a way that the farther a node is, the higher the value of the parameter is. At every step the value of the parameter decrease of 0.05, with exception of the area in which the pallet should remain. The parameters of this area are not modified. In figure 3.6, for each node the corresponding value is reported and the possible flows followed by empty pallets depending on the starting node are underlined. The nodes in the area selected define one of the tunnels described previously. The combination of this two techniques has a synergistic effect. The pallets not only are brought closer to the load station but at the same time they are hidden to the optimization process. This area can be also called **accumulation zone**, because it is needed to store the empty pallets, avoiding the controller to waste computational power.

The simulation script developed to understand how actually the improvements to the control system have been valuable can not be used to verify the effectiveness of this change. In fact, in this case, both the tests provide meaningless data. The first test gives us information about the average time of computation, and this feature does not lead to any improvement in terms of computational time. Even the second experiment

is useless principally for one main reason: in the simulation, when a pallet exits from the machine M_4, it receives as target directly the robot cell, while in the real plant they receive first "no target". This is because simulating the behaviour of the load/unload station is very difficult, unless to implement random function to call the pallet every time its work has been finished. However, assigning directly the target of the robot implies that the pallets moves in the direction of the accumulation zone as soon as they are empty. Even if this modifications have been done to implement a cycle in the plant activity, the effect is that the previous experiments would have a sort of attraction zones. So, it is practically impossible to quantify the effect on this changing in the simulation. On the other hand, on the real plant the effect is evident.

3.5 Other improvements

During the Thesis work it was necessary to work on modifications to the plant not strictly related to the improvement of the performance of the control system. For example, a simple command line interface has been created to allow the control developer (or the operator, considering the case of industrial application) to highly customize the control system work session. In this way, in the initialization phase it is possible to make some choices: in particular to decide if to use the MPC control, and then set the *NRH* desired for the session of work, or a simple controller that implements a predefined sequence of operations. Moreover, all the features described in this chapter can be deactivated if needed. This simple modification allows to have a control system highly customizable and then to speed up the test and development phase on the real plant.

3.5.1 Homing Function

At the beginning of this Thesis work, the pallets needed to be placed in certain starting positions determined during the design of the low level control system. The state at the time instant $k = 0$ was set as parameter of the internal model of the line supervisor. For this reason, different versions of the program executive, one for each different initial configuration, were developed. This because the line supervisor of the LLC was not provided of a function to read the sensors of the plant. The idea is then to implement a mechanism to scan the network in the initialization phase to obtain the initial state of the plant without being constrained to put the pallets in predefined position before starting the plant activity. This allows to try different starting configuration without modifying the control software and, in terms of industrial use, it saves considerable time every time the system is started up. Also this modification is oriented to speed up the test phase on the real plant.
The sensors of the plant, as described in the Chapter 1, are handled by different programs with respect to the line supervisor that coordinates the whole control cycle (reading/writing, updating the internal model, activating PLCs). So, It has no direct access to information from the sensors. To solve this problem, bridge variables have been created, one for each sensor of the transport line. They can provide information on the state of the sensor to which they have been associated. Thanks to the internal functions of the system, the PLCs control program constantly updates these variables in such a way that their value is consistent with the sensor status. Obviously, this communication

is unidirectional and therefore the bridge variable cannot be changed by the supervisor. The same procedure was done to allow the supervisor to directly control the actuators. Obviously, in this case, the connection (or binding) was done in the opposite direction. The binding of the actuators is fundamental because some proximity sensors need to be brought into a certain configuration before being able to check the presence of the pallet. It was therefore possible to create an SFC function that, appropriately called in the initialization phase, takes care to collect information on the status of the system and to update the internal line supervisor model in such a way that the position of the pallets passed to the high control when the system is switched on is consistent with reality.
Once the position of the pallets is defined, a target must be associated to each one. The assumption that at the beginning of the operation they are all empty is made and for all the target free ($\Gamma_i = 1$) is defined.
The state of the buffer zones composing the state of the plant in the line supervisor model can be set in the following way:

```
IF ( Sx_P_Ps_LS1 = TRUE ) THEN
BZ1_3_board_failure := 100;
BZ1_3_board_name := 0;
BZ1_3_pallet_route := Route_free;
ENDIF;
```

Where *Sx_P_Ps_LS1* is the variable defined in the line supervisor module to which is associated the value read by the proximity sensor of the module one.
Each BZ is defined with three parameters:

- Board failure. It is an integer variables that keeps track of the outcome of the tests performed by M_2. 100 is the default value and it means *'no failure'*.
- Board name. It an integer that identifies the specific electric board on the transport line. It is assigned automatically when a board is loaded on a pallet. 0 means *'no board loaded on the pallet'*.
- Pallet route. It corresponds to Γ_i for the high-level control. *Route_free* means *'no target to be reached'*.

Note that there are not defined variables comparable to the counter η_i.
A new SFC tree is added to the line supervisor module. It firstly actives all the sensors of the plant and then, for each buffer zone, it configures the initial state of the plant.

So, thanks to the attraction zone just described, when the plant starts its activity all the pallets are defined and they are pushed towards the accumulation zone.

3.5.2 Operator interface for loading/unloading

To allow to the operator of the load/unload station to substitute the robot in case of need, an interface for loading/unloading the pallets from the network has been designed and implemented. For this reason, a workstation made by a console, composed by three buttons and one notification led, and by a tool that helps the operator to open without problems the pallet picked up from the transport line has been implemented. By means of this, it has been then possible to substitute, when it is necessary, the robot in the load/unload task. The panel was created using some modular components, whereas the tool has been designed by the mechanical department of ITIA. The whole station appears as in figure 3.7. Once appropriately wired to the network, the input signal of each of the three buttons has been configured in the ISaGRAF project as well as the LED output signal.

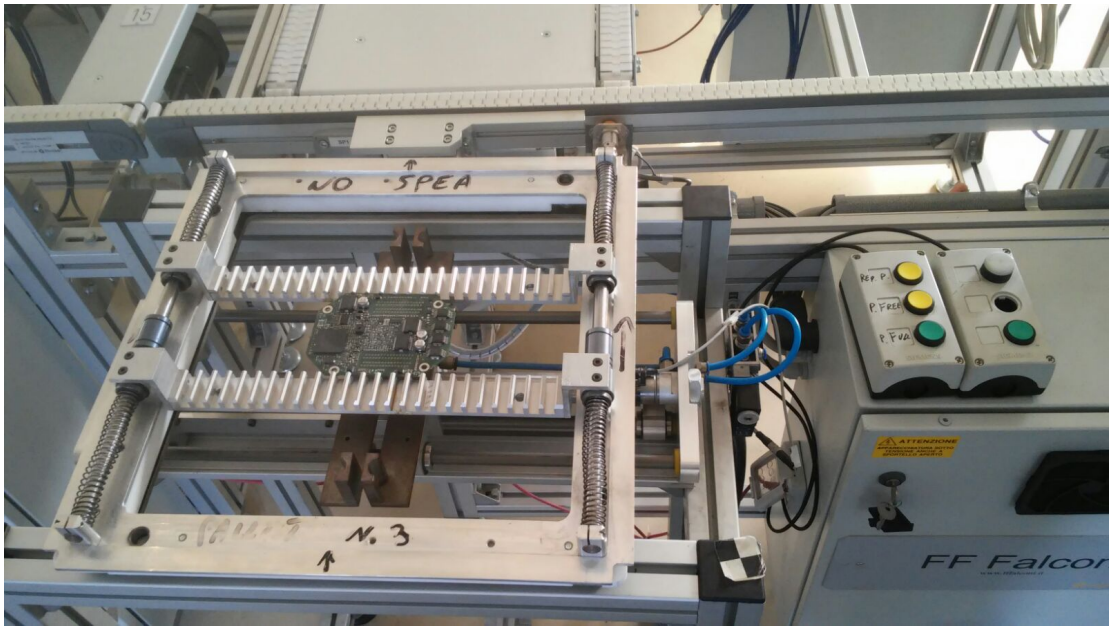

Figure 3.7: Manual Load/Unload Station

For what concerns the logic of the software, the general structure of the project has not been modified, and new SFC trees have been added to the file adhibited to handle the robot operations. This means that for the control logic point of view nothing changes, the automata describing the behaviour of the machines is still valid.
To make free the operator to decide which station use to load/unload the pallets, a new variable (`Real_Robot_using`) has been created. It must be set in the initialization phase.

From the practical point of view, one button is required to the operator to signal to the plant that he is ready to receive a pallet (for the low level control it means that the target of the nearest free pallet is changed from 5 to 1). The corresponding signal in the low level control system is `B_Req_Pallet`. The other buttons and the led are required to handle the load/unload operations when the control actions $U_{1,32}$ (the one used from the controller to communicate to unload the pallet) and $U_{32,1}$ (the one used

from the controller to allow the operator/robot to load the pallet) are activated (see figure 1.6). More in details, the new protocol for this handling the interaction between the operator and the plant is:

Control action $U_{1,32}$ activated.
The corresponding control sequence is activated, the notification led is turned on to signal at the operator that it is possible unload the pallet.
Operator picks up the pallet.
Once the pallet is taken away from the line and then the control sequence finishes, the led is turned off and the state of the "machine operator" passes from 1 to 2.
Operator works.
The plant is stuck, waiting that the end of the load/unload phase.
Operator ends.
If the operator doesn't have board to put on the pallet, he pushes the button corresponding to the LLC signal `B_P_Free` communicating to the plant that an empty pallet is going to be given back. Otherwise, he pushes the button corresponding to the LLC signal `B_P_Full` indicating a full pallet. Independently from the choice, the state of the operator passes from 2 to 3 (and then the controller understands that the control action $U_{32,1}$ could be called). It is important to note that in case of empty pallet the target will be 5, whereas if it is full the target will be 2 (the testing machine).
Control action $U_{32,1}$ activated.
The corresponding sequence is activated, the notification led is turned on to signal at the operator that the plant is ready to receive the pallet.
Pallet is received from the plant.
Once the pallet is detected, the state of the operator passes from 3 to 1, the led is turned off and the sequence finishes.

3.6 Experiment on real plant

To validate the data obtained in the simulation, the MPC controller has been tested on the real plant.
Not being able to replicate previous experiments because of the high number of iterations required (one thousand, for the second one) or for the impossibility to pre-set the target of each pallet at the instant k = 0 (because of the homing function all the targets are placed equal to one in the initialization phase), it was decided to shorten the interval of iterations to 50 and use only three or four pallets.
The validation work was done only on the average computation time, since with such a short horizon it would not make sense to analyse the number of pallets processed.

The starting position for each test was the following:

pallet 1 in N30;
pallet 2 in N29;
pallet 3 in N4;
pallet 4 in N9;

With a specific API of the cplex libraries, the time at each iterations has been stored in a list and at the end of the program they have been saved on a text file. Then, the data about the computational time have been elaborated. They are shown in the table 3.5

Table 3.5: Average time [s] with basic MPC controller (A); Average time with Tunnels (B); Average time with Control Horizon MPC (C)

Num. Pallets	$N_{RH} = 4$	$N_{RH} = 5$	$N_{RH} = 6$
3	0.17	0.29	0.92
4	0.24	0.49	1.18

(A)

Num. Pallets	$N_{RH} = 4$	$N_{RH} = 5$	$N_{RH} = 6$
3	0.09	0.29	0.52
4	0.15	0.43	0.80

(B)

Num. Pallets	$N_{RH} = 4$	$N_{RH} = 5$	$N_{RH} = 6$
3	0.13	0.26	0.76
4	0.22	0.45	0.91

(C)

Comparing the results obtained with those obtained in simulation it is possible to say that the model for the simulations is reliable and the results shown previously have actually been achieved also in reality.

Chapter 4

Fault Tolerant Control

Nowadays, performing well is no longer the only requirement for an industrial plant. Failure robustness has now become essential in industrial applications. In fact, the proliferation of sensors and the increase in the computational capacity of the calculators allows the real-time analysis of the components to identify, and in some cases predict, the faults. Therefore it is essential to develop detection and recovery systems to allow operation even under unexpected conditions. To be precise, a fault is defined as an unpermitted devation of at least one characteristic property or variable of the sytem [12]. Generally speaking, the faults occurring in the industrial world belong to two categories: additive process faults, that are unknown inputs acting on the plant modifying its output (plant leaks, for example), and multiplicative process faults, that are changes of some parameters of the system (the deterioration of the plant equipment, for example) [9]. However, in this thesis, only the ones belonging to one of this subset of the two groups have been taken into account:

- Sensor faults. These are differences between the measured and actual values of individual plant variables.
- Actuator faults. These are discrepancies between the input command of an actuator and the actual input.

To guarantee that the process operations satisfy the level of efficiency requested even when a failure occurs, any anomaly needs to be promptly detected and, if possible, the system should be reconfigured in order to remove its effects. These tasks are associated with process monitoring. The goal of this technique is to ensure the success of the planned operations by identifying the anomalies of the behaviour. As a result, downtime is minimized, safety of plant operations is improved and manufacturing costs are reduced.
According to the definitions given by Raich and Cinar [20] the process monitoring can be divided in four steps (their logic sequence is presented in figure 4.1):

- fault detection, that consists in understanding whether a fault has occurred;
- fault identification, that consists in understanding which component is responsible of the malfunctioning;
- fault diagnosis, that consist in understanding the cause of the observed discrepancy in the status;

- process recovery, that consists in removing the effect of the fault. These steps are not always all necessary and some could be delegated to an operator.

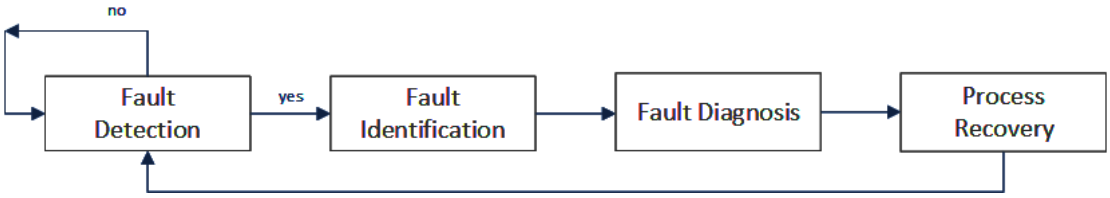

Figure 4.1: Process monitoring scheme.

The idea behind the fault detection is to convert on-line data acquired from the plant into a few meaningful measures, which represent the state of health of the system, and to understand when a fault occurs. The fault detection algorithms can be based on three different approaches:

- Data driven, in which a large scale of data produced by sensors are analysed to identify anomalies;
- Analytical, in which a mathematical model of the system is used to estimate the correct expected state;
- Knowledge based approach, based on a qualitative model of the system.

The data-driven methods are more indicated for large scale system and their efficiency is strictly related to the quality and the number of the sensors and the meaningfulness of the data acquired. For this reason, this kind of approach is not suited for the goal of the thesis.

In the next few paragraphs an analytical and a knowledge based approach are presented. For each one, the pros and cons are underlined regarding their application on the CNR pilot plant. Both techniques have been developed starting from the following assumptions:

- For each control sequence, it is not possible to have simultaneous actuator and sensor faults.
- It is supposed that there is not error in the model parameters.
- It is assumed that when the system is switched on all its components are in perfect conditions.
- For each control sequence, only one fault can occur.

4.1 Fault Detection: Analytical Approach

The analytical approach applies to relatively small (few inputs, outputs and states) systems where satisfactory models and enough sensor are available. It is based on the concept of analytical redundancy. In contrast to physical redundancy, when measurements from different sensors used to measure the same quantity are compared to each other, in this case, sensor measurements are compared to computed values of the respective variable. The resulting differences, called residuals, are representative of the presence of faults in the system. In other words, they are the results of consistency checks between the plant observations and a mathematical model (as summarized in figure 4.2). It is then possible to say that, generally, an approach of this type arrive at a diagnostic decision based on the residuals [19].

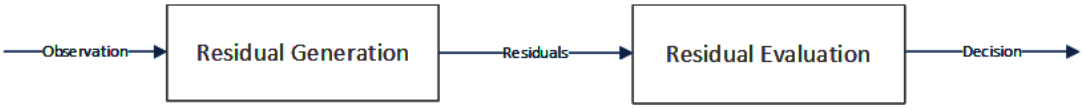

Figure 4.2: Fault detection composition.

There are different ways to generate the residual, the one designed for our plant is the observer-based method, in which the output of the system is derived from the measurements with the aid of observers. The vector of residuals is defined as the difference between the measured and the estimated output.
The difficulties of this type of approach are mainly two: the ability to construct a reliable observer of the system and to set a threshold of the error used to judge whether the residual is zero or non zero. However, both problems for our system seem to be easy to solve.
To guarantee the correct operation of the system, it is important that at each step the pallets are actually in the buffer zones corresponding to non-zero elements in the status vector returned by lines supervisor to the DCPIP. Concerning the first issue, by analysing the structure of the hierarchical control system developed during the thesis, it is possible to conclude that the line supervisor of the low level control already acts the role of observer of the system status and then is not necessary to implement a new one. Concerning the threshold, since all the elements of interest are boolean variable associated to the presence of a pallet in a certain buffer zone, the problem is trivial. Let's define $x_i^m(k)$ the vector of the state given by the observer, $x_i^s(k)$ the one given by the sensors of the plant and N_{BZ} as the total number of buffer zones:

$$x_i^m(k) = \begin{cases} 0 \quad if\ the\ i-th\ BZ\ is\ not\ occupied \\ 1 \leq x \leq 5 \quad otherwise,\ \ x \in N \end{cases} \qquad i = 1,..,N_{BZ}$$

$$x_i^s(k) = \begin{cases} 0 \quad if\ the\ i-th\ BZ\ is\ not\ occupied \\ 1 \quad otherwise, \end{cases} \qquad i = 1,..,N_{BZ}$$

Since at this stage there is not interest in the target of each pallet and the objective is to

have two measures that give the vector of residual by means of a simple subtraction, the elements of x_{ij}^m are normalized with respect of their target t_i and a new vector is defined:

$$\tilde{x}_i^m(k) = \frac{x_i^m(k)}{t_i} \quad , \quad \tilde{x}_i^m(k) = \begin{cases} 0 & if\ the\ i-th\ BZ\ is\ not\ occupied \\ 1 & otherwise, \end{cases} \qquad i = 1,..,N_{BZ}$$

So, it easy define the vector of residual $r_i(k)$ as:

$$r_i(k) = |\tilde{x}_i^m(k) - x_i^s(k)| = \begin{cases} 1 & if\ there\ is\ a\ discrepance\ between\ model\ and\ reality \\ 0 & otherwise, \end{cases}$$

In this way we have obtained N_{BZ} residuals, each one corresponding to a buffer zones, with which is possible to build the matrix of signatures. It is a matrix that has on the rows the residual and on the columns the faults that could occur. If a fault k impacts on a residuals p, the correspondent element e_{pj} is equal to 1. With f_i it is indicated the fault occurs to the set of operation concerning the movement of the pallet into the buffer zone i.

Table 4.1: Residual matrix analytical method

	f_1	f_2	f_3	f_4	..	f_n
r_1	1	0	0	0	..	0
r_2	0	1	0	0	..	0
r_3	0	0	1	0	..	0
r_4	0	0	0	1	..	0
..	..	..	..	..	..	..
$r_{N_{BZ}}$	0	0	0	0	..	1

Note that the matrix is in canonical form, so the identification and the isolation of the fault is possible. A possible scheme of control for this kind of approach is presented in figure 4.3, where $U(k)$ is the vector of control actions computed by the MPC, $x\_p(k+1)$ is the predicted state useful to calculate the control actions at $k+1$, s_j is the set of control sequences needed to implement the control actions, $x(k+1)$ is the real state of the transport line at instant $k+1$ and $R(K)$ is the matrix of residual necessary to the controller to understand if there has been any faults on the system.

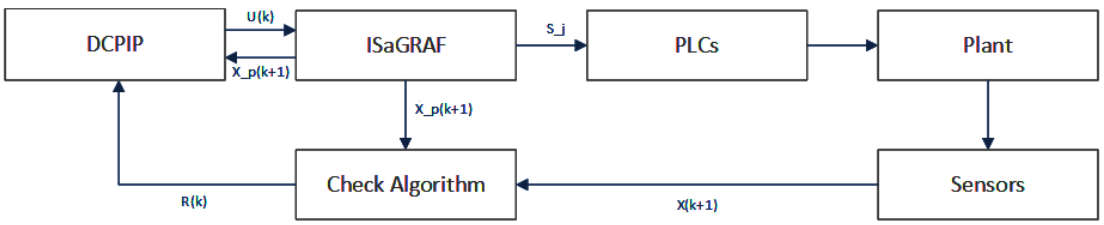

Figure 4.3: Residual-approach operations sequence.

With this method it is therefore possible to understand in a simple way which sequence of control actions has not been terminated due to a fault. What is not possible, however, is to understand directly which element associated with that sequence is broken.

Although apparently this is a particularly suitable approach to the fault detection of our system thanks to the fact of not having to design an observer from scratch and the simple management of residual, it was not possible to implement it. ***Reading the status of the transport line at each step would slow down the execution of the control sequences too much and therefore cause a drastic reduction of performance***. For this reason, it was necessary to find a detection system that did not require the reading of the various sensors.

4.2 Fault Detection: Knowledge-Based Approach

An alternative approach for process monitoring is to use knowledge-based method such as causal analysis, based on casual modelling of fault-symptom relationships, or expert systems, used to mimic the reasoning of human experts. [14].
The main idea behind this approach is to build a qualitative model of the process exploiting the knowledge of the system and then use it to derive the matrix of the signatures. Therefore the only difference with respect to analytical redundancy methods is due to the qualitative analysis used in the definition of the residuals. However, this way of reasoning could introduce some qualitative ambiguities in the model building and then a lack of resolution.
Taking advantage of the modular characteristics of the transport line, it is possible to take into account a sample of three consecutive buffer zones for defining some qualitative rules and then extend the reasoning to the whole plant. In this way, it is possible to define rules for the detection and identification of faults by developing effect cause relationships exploiting the knowledge about the operations necessary to move a pallet.

Let us consider a module with three buffer zones called A, B and C, two further boolean variables $w_{A,B}$ and $w_{B,C}$ are associated to the control actions that move the pallet from A to B and from B to C, respectively. They are set to one in case of some problems occur during the control sequence associated to these control actions. The idea is therefore to use these variables to construct the matrix of the residuals of the subsystem and to extend it to the whole transport line. At the beginning, a pallet is in A. For simplicity, it can be assumed, without loss of generality, that each control sequence involves an actuator (a motor) that takes care of the displacement and a proximity sensor positioned at the end of the pallet path that indicates to the PLC to switch off the actuator. The situation is sketched in figure 4.4, where in red is indicated the position of the pallet for the line supervisor (and then for the high level control) and in green its actual position.

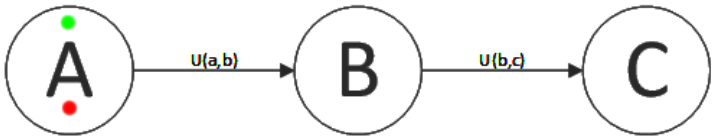

Figure 4.4: BZs analyzed for qualitative rules.

When the coordinator activates the action $U_{a,b}$ and a fault occurs, there are two possible

evolutions of the system:

- The fault affects the motor, and so the pallet does not arrive in B.
- The fault affects the sensor, and so the pallet arrives in B but it is not detected.

At the beginning of this Thesis, the low level waited for the SFC sequences implemented in the PLCs to end. In the event of a fault, therefore, the system was blocked. The idea is then to implement a timer for the PLC within which the operations must be completed, otherwise the PLC is forced to finish and a boolean variable ($w_{A,B}$ and $w_{B,C}$, for the case taken into account) is set to true to keep track that something has gone wrong. However, the communication between the line supervisor and PLCs is not changed. This means that even if the control sequence is terminated due to the timer, the PLC signals that it has finished the operations to the supervisor without the latter being informed of the incident. In fact, the low control does not analyse the warning variables but, on the contrary, limits itself to writing on a text file because the high control could read them.
This design choice has been made to allow the development of an algorithm capable of analysing the situation a posteriori and to prevent the block system every time a fault occurs.

Keeping this change in mind, imagine the evolution in the system in the two cases presented above.

Actuator fault

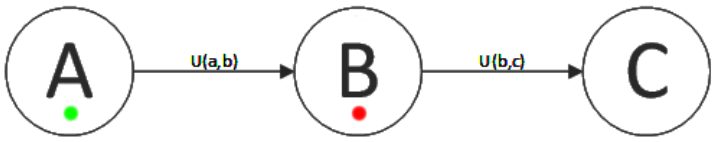

Figure 4.5: BZs analyzed after actuator fault.

In case of failure of the actuator, and therefore of the motor that moves the conveyor belt on which the pallet is placed, the pallet remains in the buffer zone of departure (A). The sequence associated with $U_{a,b}$ ends due to the associated timer because the proximity sensor is not activated by the arrival of the pallet. As a result, the PLC sets $w_{A,B}$ and signals to the supervisor that the control action has been terminated. The supervisor updates his model of the system as if the operation were terminated correctly. Then, it passes the vector of the state to MPC algorithm. The evolution of the system is presented in figure 4.5.
When the high-level control activates the control action $U_{b,c}$, the motor related of the path from B to C is turned on. Since there are no pallets in B, the sequence must end due to the timer. For this reason, even the variable $w_{B,C}$ is set to 1. Reasoning in this way, it is possible to build the residual matrix using the warning variables as residuals:

Table 4.2: Residual matrix for actuator faults

	f_a
r_1 ($w_{A,B}$)	1
r_2 ($w_{B,C}$)	1

Sensor fault

Contrary to what has been said for the case of the actuator, if a sensor breaks, there is no discrepancy between reality and the model, as shown in figure 4.6. In fact, the conveyor system has worked perfectly and the pallet has been brought from A to B. The problem is that its arrival is not detected and the sequence must finished due to the timer. As a result, the variable $w_{A,B}$ is set to 1.
When the high-level control activates the control action $U_{b,c}$, the motor related of the path from B to C is turned on. Since it has been assumed to consider the single failure case, the pallet arrives in C without problem. This means that the correspondent sequence finishes correctly and the variable $w_{B,C}$ is kept to 0. As before, it is possible to build the residual matrix using the warning variables as residuals:

Table 4.3: Residual matrix for sensor faults

	f_s
r_1 ($w_{A,B}$)	1
r_2 ($w_{B,C}$)	0

With this type or reasoning, the control system become totally robust to the sensor faults from the moment that they are simply ignored and by using the prediction gives by the model/observer supplies to the error feedback taken from the plant. So, it is able to compensate for the wrong information received from the system through the use of data generated by the mathematical model. In a sense this is the idea behind the concept of the virtual sensor application [7].
The matrix of residual, considering both the faults, is:

Table 4.4: Residual matrix

	f_s	f_a
r_1 ($w_{A,B}$)	1	1
r_2 ($w_{B,C}$)	0	1

Note that the case in which the engine remains always active is not interesting for the analysis of this Thesis because the system is equipped with mechanical stops at the end of each pallet path and therefore the entire transport system would work even without the application of the fault detection algorithm. More interesting is the case in which a sensor constantly detects a pallet in a buffer zone even if it is not present in reality. In this case, since the control sequence that moves a pallet from A to B would terminate instantaneously seeing the pallet already in B, this fault is treated as a failure of the actuator (it entails the non-arrival of the pallet in the planned BZ).

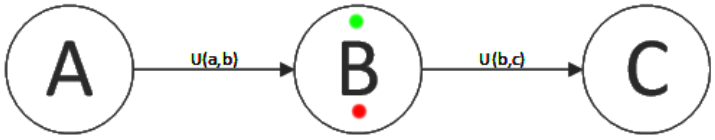

Figure 4.6: BZs analyzed after sensor fault.

4.2.1 Signed Directed Graph

In order to build the matrix of the signatures associated with this method, it is possible to formalize the previous considerations by means of the Signed Directed Graph (SDG). It is a graph showing the way in which the process variables are related each other and describes the behaviour of the system after some events. The SDG developed for the subsystem of the three buffer zones considered in figures 4.4, 4.5 and 4.6 is reported in figure 4.7. Nodes represent physical variables, while arcs the mutual influence. More specifically, the nodes of this graph represent the variable associated with the presence of a pallet in the buffers zones.

Generally speaking, a node can assume three values: + when its measure is larger than in the normal conditions, - whet its value is smaller and 0 when its measure can be considered equal to the one in normal conditions. Each arch represents the correlation between the nodes, called the cause node and the effect one, and it has a sign that can take a value of + or - depending on the type of relationship between two nodes. If they change in a similar way there is a +, otherwise -.

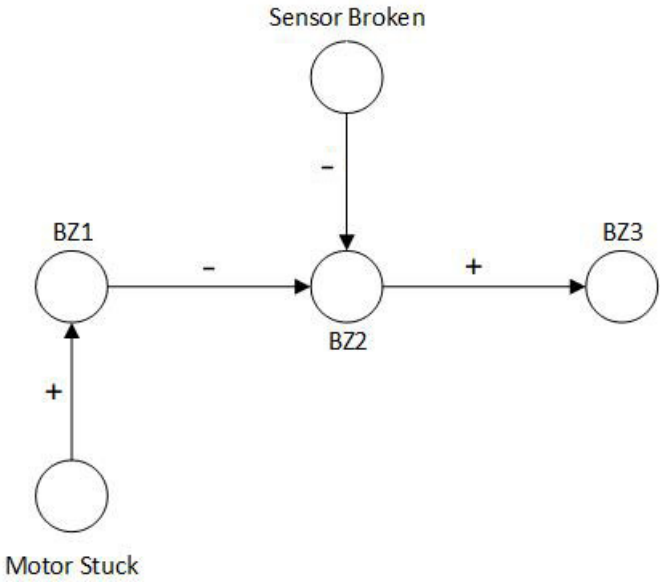

Figure 4.7: SDG for the subset on the transport line.

The goal of SDG is to locate the possible root nodes representing the system faults based on the observed symptom. To achieve this, the measured node deviation are propagated from the effect nodes to the cause nodes via consistent arcs until the root

nodes are identified. An arc is consistent if the sign of the cause node times the sign of the arc times the sign of the effect node is positive.
In figure 4.8 it is reported the case where the symptom is the lack of the pallet in the buffer zones 2 and 3 after the execution of the control action and its presence in the first one. Based on a consistent path check the fault is determined uniquely as "Motor Stuck" from the moment that a + in the other unmarked nodes results in an inconsistent arc. The same reasoning can be done for figure 4.9, where the symptom is the missed detection of the pallet in BZ2 after the control action that moves the pallet from BZ1 to BZ2. The identified fault is "Fault broken".
Once the SDG has been obtained and the fault-symptom pair has been determined, it is possible to define the signature matrix. The nodes become our residuals and the corresponding element in the matrix of residuals is set equal to 1 when the variation of that variable is caused by the fault characterizing the column. In order to reduce the dimension of the matrix, only two residual are taken into account: the first residual corresponds to BZ1 while the second corresponds to BZ2. As easily seen from the final matrix (table 4.6), two residuals are enough to detect and isolate the two faults.
The matrices of the signatures related to SDG in figures 4.8 and 4.9, are the following:

Table 4.5: (A) Residual matrix for actuator faults; (B) Residual matrix for sensor faults

	f_a
r_1	1
r_2	1

(A)

	f_s
r_1	0
r_2	1

(B)

For the two faults, the overall the matrix of residual is:

Table 4.6: Residual matrix for sub-set of three BZs

	f_a	f_s
r_1	1	0
r_2	1	1

The same type of reasoning can be extended to transport sequences involving more than one sensor and one actuator. In this case the components are divided into two groups: the group used to bring the pallet from one node to another and the group used to monitor the buffer zones to detect the arrival of the pallet. In this case, the fault detection algorithm will no longer locate the failed component, but the group of components in which to look for the fault. By combining the various matrices associated with the groups of three buffer zones, it is possible to obtain a single matrix that allows us to identify a fault considering the whole system and at the same time to decide if it is related to a sensor problem or if it concerns the actuators. This is fundamental because, as already underlined, the control system developed in this Thesis reacts differently to

the two types of faults: as explained in the following, it compensates through the prediction of the state to the fault on the sensors, while it is reconfigured when an actuator fault occurs so as to exclude from the control actions the failed ones.

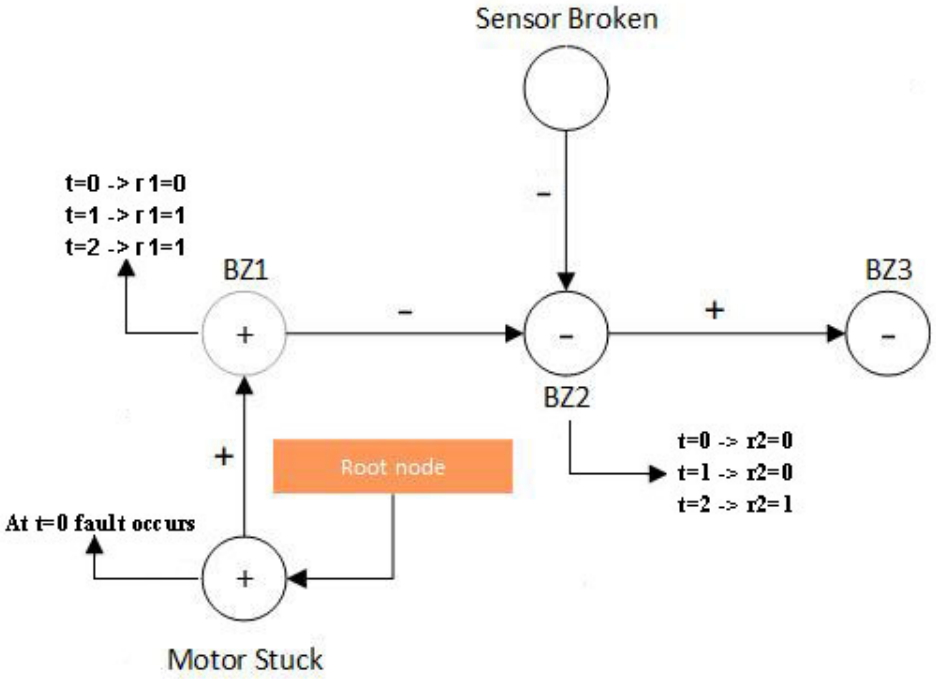

Figure 4.8: SDG actuator fault.

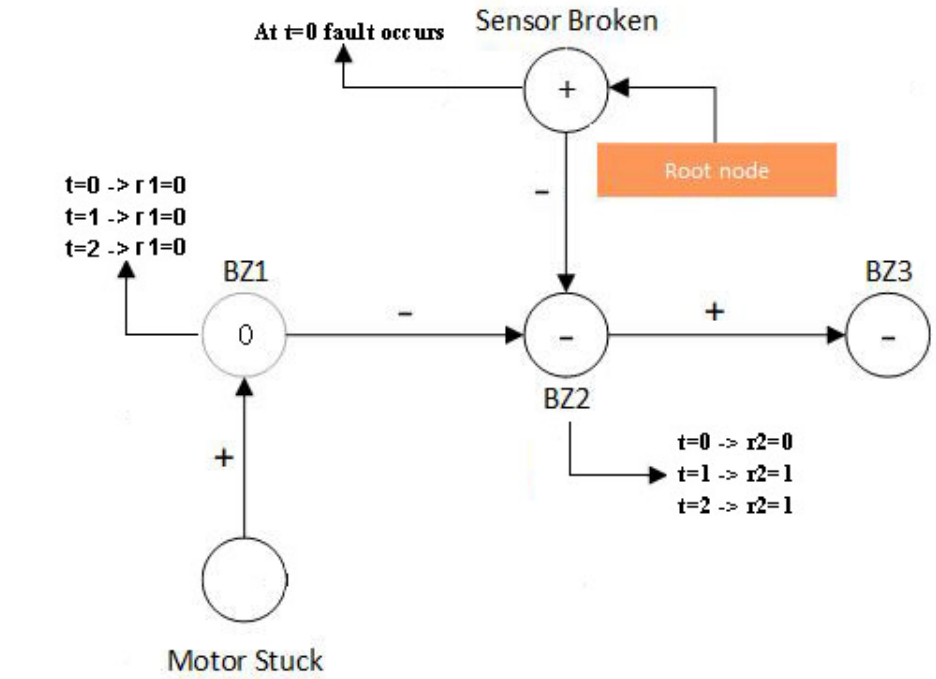

Figure 4.9: SDG sensor fault.

4.3 Fault Detection: Implementation

The implementation of this detection mechanism concerns both levels of the control system. As for the low level, it has been necessary to modify all the control sequences by adding the mechanism to exit from the cycle in case of fault. For the high level, instead, it has been necessary to think to a way to translate the received data of the PLCs into a data structure similar to the residual matrix.

Changes to LLC

As for the low level of control, all its sequences have been modified adding a timer to make the action of the PLCs terminate in the event of fault. Moreover, if this action ends up due to the timer, a new variable reports to the high control level that problems have occurred to the corresponding control sequence. Let's take as example the sequence number one, reported in the figure 4.10.

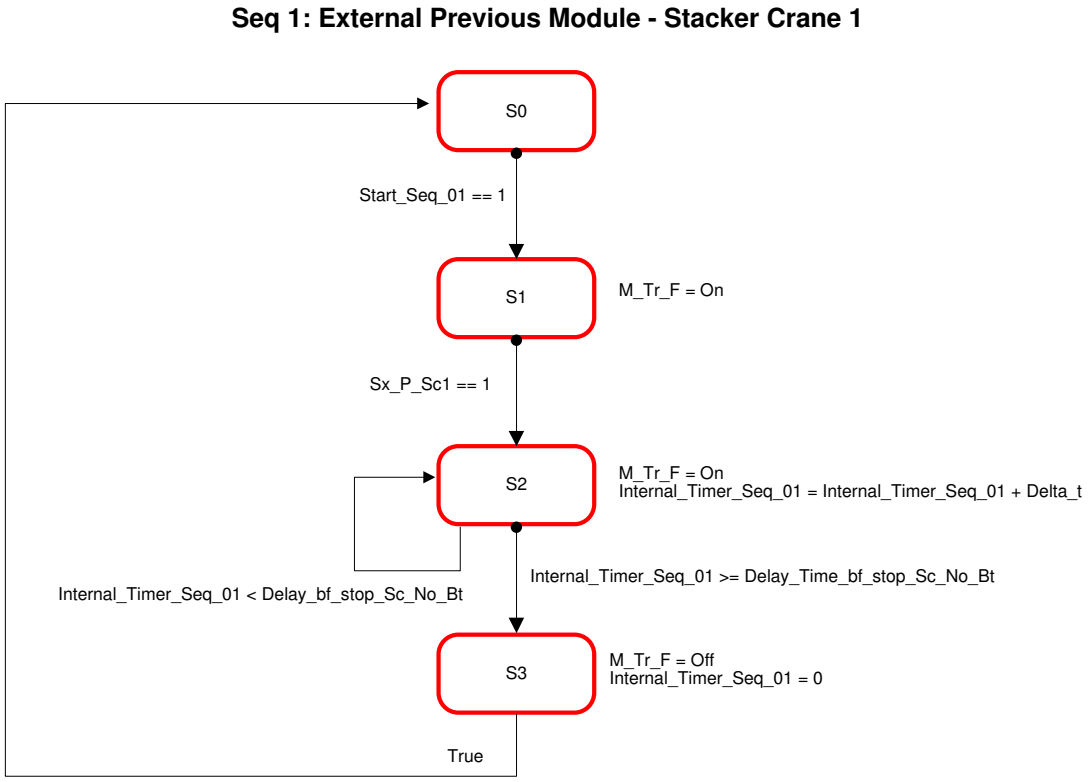

Figure 4.10: Example of SFC sequence before FD implementation.

When the signal of starting the sequence arrives, the PLC turns the motor on so that the conveyor belt is activated and the pallet begins to flow in the direction of the arrival zone buffer. The motor is kept on until the proximity sensor detects the arrival of the pallet, after which the motor is switched off and the PLC signals to the line supervisor that the control sequence has been terminated. It means that in the event of a motor or sensor failure, as there are no mechanisms for which the PLC can detect it, the control sequence does not end and the whole system is blocked. The sequence has therefore

been modified by adding a branch to the SFC tree as shown in figure 4.11.

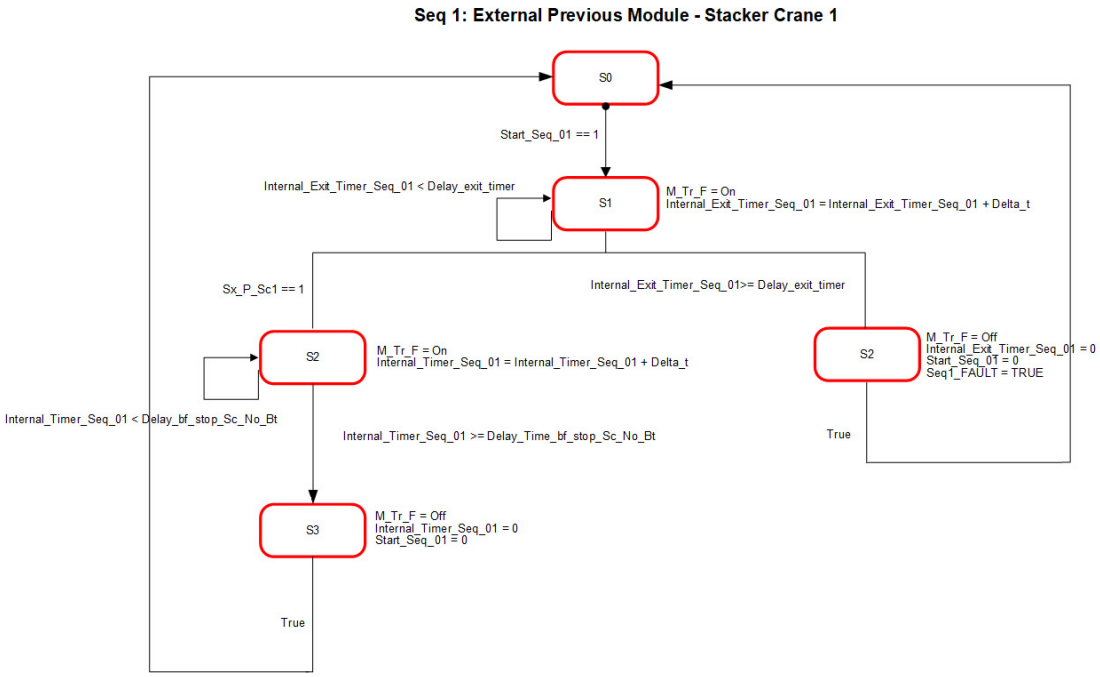

Figure 4.11: Example of SFC sequence after FD implementation.

New boolean variables are then created as warning variables. Each variable is called *Seqns_FAULT*, where *ns* indicates the number of the control sequence to which it belongs. At any moment k, it is necessary to run a function to acquire the values of these variables and communicate them to the high level. This communication is carried out, as in the case of transmission of the status, by writing the data just acquired as a vector, to a specially created text file. To achieve this task, a SFC sub-program, called by the line supervisor, is created. Its behaviour can be schematized as in figure 4.12.

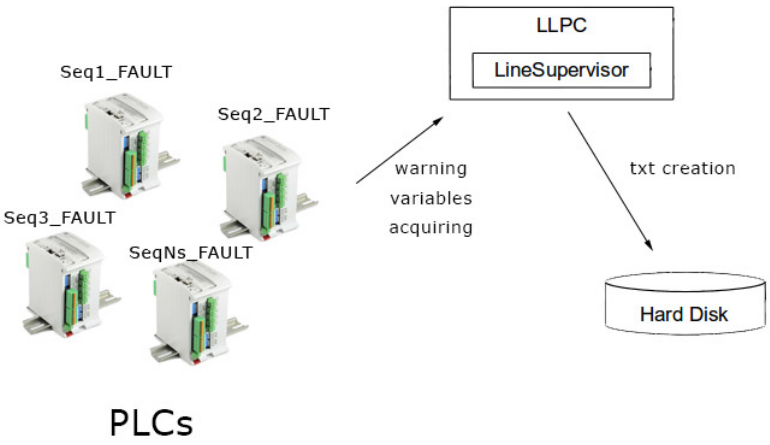

Figure 4.12: Warning variables gathering.

Changes to HLC

In the high level control, a new function *faultsHandler* has been added to the *Deman Control* class. It has the task of reading the vector of the alarm variables and interpreting them in order to let the controller understand whether a fault occurs. The operations necessary to interpret the information read from txt file are mainly two. First of all it is necessary to find the control actions corresponding to the sequences where problems were found. After that, it is necessary to find an algorithm that allows to understand if it is a fault of the sensor or of the actuator interpreting the matrix of the residual.
For what concerns the control actions determination, a control action can be composed by one or more control sequence and it must be marked as problematic if at least one of them is finished for effect of the exit timer (Table 4.8 lists the control actions supported by the sequences that compose them).
For this reason, at every cycle the program builds the vector ca_fault of fifty-one elements, in which each element is associated with a control action. The construction of this vector is based on the data read from the text file containing the values of the warning variables, since each of them corresponds to a specific control sequence. So, by analysing these variables it is possible to understand which sequence is in fault, and then which control action. If a warning variable is set to one, the corresponding element of ca_fault is increased by 1 to indicate that a problem has been detected for that certain control action. So,

$$ca\_fault[i] = \begin{cases} 0 \quad (if\ the\ i-th\ control\ action\ is\ not\ problematic) \\ bigger\ or\ equal\ to\ 1 \quad (otherwise) \end{cases}$$

Once the translation from control sequence to control action has been done, each element of the new vector is analysed and the following rule is applied to determine if a fault affects a sensor or an actuator (the vector *previous_ca* is a copy of the control action applied to the system at the precedent cycle):

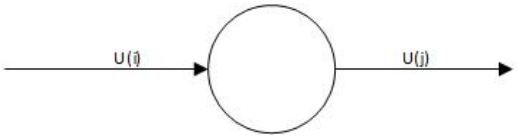

Figure 4.13: DCPIP class structure.

```
if ((ca_fault[i] > 0) AND (previous_ca[j] == 1))
if (ca_fault[j] > 0) {
cout << ACTUATOR FAULT << endl;
} else {
cout << SENSOR FAULT << endl;
}
```

To better explain this rule, let us consider a situation in which a node has one control input action and one control output action, as in figure 4.13. It is worth noting that, although the structure of the rule is the same for all the elements analysed, a thorough analysis of the system is fundamental to understand what control actions are to be considered or not in this analysis (from a practical point of view this means that choice

of indices i, j is not trivial and should be weighted).
For each element of ca_fault different from zero, *previous_ca* is checked in order to understand if at the previous step the control action corresponding to its output has been called. In this case, even the the correspondent element of *previous_ca* is checked and there are two cases:

- It is bigger than zero, so that the control action is in fault and there is a problem on the actuator
- It is zero, so that the control action is not in fault. It means that there is a problem on the sensor and the system can ignore it

The check on the control action is essential because, with respect to the situation presented above, it is not known when the controller call U_j and the algorithm is made in such a way to ignore the fault until it has identified its type.

With this method, there are two relevant advantages: it is possible to detect and isolate a fault without introducing delays due to the operations necessary to read the sensors and the control system become totally robust to the sensor fault. On the other hand, the failures are detected with a cycle of delay and this could cause some problems. For example, if there is a pallet too close to the ones stuck due to an actuator fault, and the control system detect lately this fault, the pallets could crash and an electronic board that could have taken another path is lost.

4.4 Fault recovery

Fault recovery is a critical part of process monitoring required to obtain a controller capable to operate even in the event of a fault in one of its components. In this phase, the control system is reconfigured, when possible, to eliminate the effects of the detected failures.

Since all the faults that can happen to the system are potentially critical so such as to block the plant activity, there is the need to implement an active fault tolerant control mechanism like the one shown in figure 4.14. Model Predictive Control is particularly suitable for this type of solution because it allows quickly to add new constraints or modify the objective function at each step, thus including the conditions imposed by the breaking of a component in the control algorithm.
The fault detection system previously presented provides the high-level controller with the information necessary to reconfigure the MPC algorithm or, in the event of an unrecoverable fault, to stop the transport line and report to the operator where to take action to restore normal activity. The task of the recovery function is then to understand when and how the controller has to be reconfigured. In the following, the rules set to handle the actuators and the sensors faults are presented.

It is important to underline that the strategy adopted in this case is not able to guarantee a full fault tolerant control, i.e. the ability to operate without significant degradation of performance in the event of any failure, solely due to the physical configuration of the

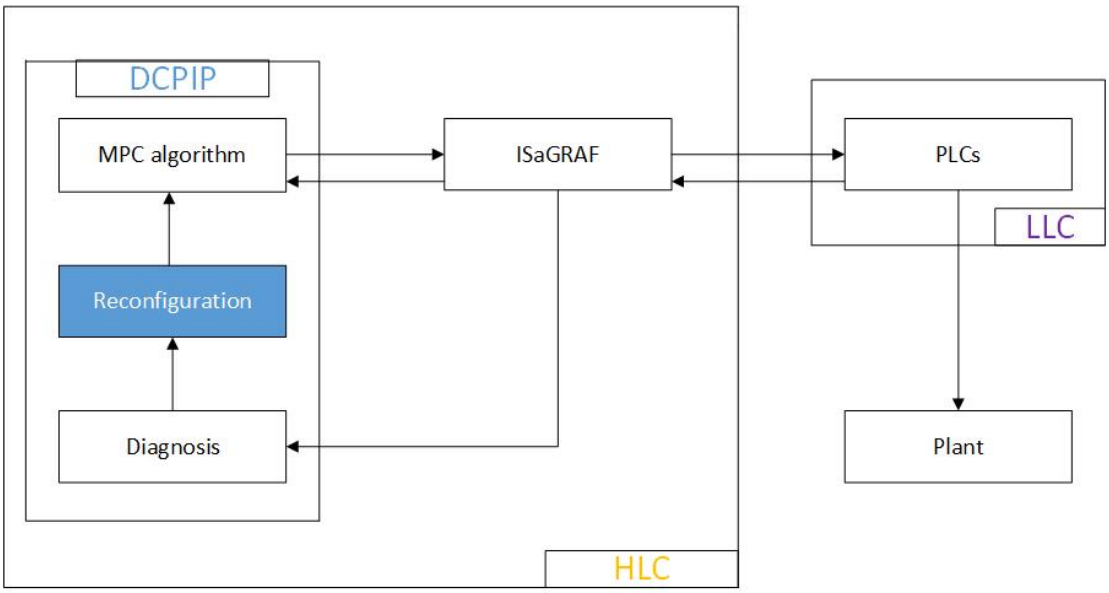

Figure 4.14: Active fault tolerant control scheme.

system. The plant is considered to work properly when it is able to guarantee that the work cycle is performed, so when it is certain that a pallet loaded in the load/unload station is taken to the testing machine and, depending on the outcome of the test, to the other two work cells. The particular configuration chosen for the arrangement of the modules and the impossibility of using the motors to make the pallets turn back after a fault greatly reduces the chances of finding alternative ways to bring the pallets from one machine to another.
With some changes to the layout of the modules or to the structure of the modules, it could be created a much more connected graph on which a fault recovery strategy would be more effective.

Fault Recovery: Sensor Faults

As far as the faults on the sensors are concerned, the control system is totally robust as long as there is only one sensor broken in the sequences involved in the movement of the pallets. This limitation is derived from the fault detection algorithm used. In fact, in the case of several sensors broken on two consecutive control actions, it would not be possible to distinguish between faults of the actuators and of the sensors. In this case, the residual matrix would have two identical columns, making it possible to perform only the failure detection and not the identification.
Thus, the high-level has been designed to use as state its prediction obtained from the line supervisor, that is used as observer to replace the real sensors in fault with a variable obtained from the mathamatical model. This principle is the based of the virtual sensors technique [7].

4.5 Fault Recovery: Actuator Faults

The fault management on the actuators is much more complicated than the one of the sensors. The recovery process can be divided in three subtasks:

- Understanding if the fault is a critical one, so that the system should be stopped.
- Adding the new constraints to the MPC optimization problem to avoid the controller to invoke the control action associated to the failure.
- Changing the ISaGRAF model to obtain a correct prediction of the state.

Fault Entity Evaluation

Determining the severity of an actuator rupture in the case of the pilot plant means understanding whether the system is able to continue the de-manufacturing activity after a reconfiguration of the controller. To do this, it is possible to use the advanced mathematical techniques defined for the oriented graph analysis [1]. There are several algorithms to determine if two nodes are connected or not. In particular, what interests for the analysis carried out in the Thesis work is that some nodes, those corresponding to the machines, are connected to each other by at least one path. In particular, to guarantee operation, the following routes must always be guaranteed (the nomenclature of figure 1.4) are used):

Table 4.7: List of fundamental paths of the plant

S	E	Path description
$N32$	$N33$	The path that connects the robot cell with the testing machine
$N33$	$N34$	The path that connects the testing machine with the reworking one
$N33$	$N35$	The path that connects the testing machine with the discharge board
$N33$	$N32$	The path that connects the testing machine with the robot cell
$N35$	$N32$	The path that connects the discharge board with the robot cell
$N34$	$N33$	The path that connects the reworking station with the testing machine

Note that the path from $N32$ to $N33$ is not equal to the one that connects $N33$ to $N32$. The same holds for all the nodes.

It is therefore essential to implement an algorithm to understand if these paths are feasible every time a fault occurs and so to understand if a fault is critical or not. This can be done in two ways: an off-line study of the graph is carried out and a lookup table is created in which an outcome is associated to each possible sub-graph or on-line checks are carried out dynamically.

Off-Line analysis of the effects of an actuator fault on the transport line, which regarding the graph corresponds to the elimination of an arc, is very complicated. This is because a considerable number of configurations should be analysed since our analysis has to take into account even the possibility that more than one fault on different control

actions occurs. There is the possibility, for example, to have a problem with the generic control action $U_{a,b}$ and after with $U_{b,c}$. As a consequence, two different configurations must be studied: the graph without the arc correspondent to $U_{a,b}$ and the one without the arcs corresponding to $U_{a,b}$ and $U_{b,c}$. Note that if the failures were in the reverse order, a different configuration should be analysed. From a mathematical point of view, without using heuristic rules that could significantly reduce the number of sub-graphs (for sub-graphs it is meant one in which the node are the same, but the arcs are different) to investigate, the possible configurations obtainable are $2^{n_{ac}}$ and so, in this case, are 2^{51}.
Even if for a large number of cases the solution is obviously trivial, off-line studies of this type are very time consuming and the construction of a C++ (lookup) table in which the sequence of faults are related to their entity is very complicated. Moreover, an analysis of this type would not lend itself to larger plants as the number of graphs to be investigated grows exponentially as the number of edges in the graph increases.

An on-line analysis in which the graph is implemented as a proper data structure in the C++ platform and the arcs are eliminated dynamically has been taken into account. In fact, it is possible to create a new data structure *node* on which the main algorithms of the graph theory are easily applicable. On the web there are a lot of open source libraries [6, 13] to be downloaded for describing and handling graphs for every programming language. With the help of these libraries, it is possible to store the model representation of the transport line as a private member of the *Deman Control* class. In this way, this data structure is stored in memory for all the work sessions of the plant, as the modifications that have been done on it. Consequently, when there is a failure of an actuator, the controller could dynamically delete the corresponding arc and then apply an operational research algorithm, such as Dijkstra, to understand if the paths that connect the machines still exist.
At the beginning of the work, this approach has been taken into account, but an evident delay was introduced since the algorithms are quite time-consuming (especially considering that every step lasts about five seconds). Moreover, the structure associated with the graph increases in dimension as the dimension of the plant increases. So as for the time requested to perform the algorithm chosen at every step.
Therefore, the solution described can so applied in cases where the computing power is not a problem and is not suitable for large transport lines.

None of the two previous options is therefore satisfactory for the needs of the pilot plant. For this reason, a new solution was sought. The algorithm is represented by a hybrid method in which, through the off-line preliminary study of the graph, the complexity of the problem to be solved on-line is reduced.
The analysis is moved from the directed graph derived from the MLD model to a very simplified one in which only four nodes, the machines, and six arcs are present. It is shown in figuree 4.15. The off-line investigation has a primary role in the definition of the arcs. They do not more represent a control action to move a pallet between two buffer zones, but they are a boolean expression derived from this analysis. If the expression equivalent to an arc is equal to true, it means the graph is no more connected and the system must be stopped. On-line control is greatly simplified as it is reduced to a simple calculation of a boolean expression.

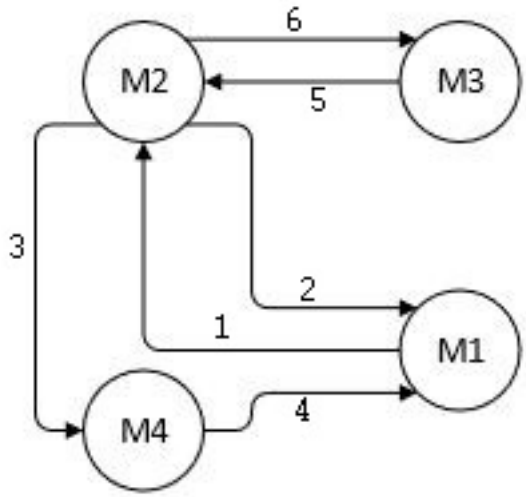

Figure 4.15: Simplified plant scheme.

The preliminary study was aimed at determining these expressions. For each couple {S,E} of Table 4.7, the operations to be carried out are as follows:

1. Following a Best-First Search approach, a data structure (called in the following Reachable Tree) similar to the tree one, in which are represented all the possible paths from one node to another, has been created (particular attention must be paid to the cyclical routes). It is a graph in which there is a root node, the starting one, corresponding to the starting machine and the leafs are all representing the arrival machine. The arcs between the nodes are the control action required to move the pallets as in the complete graph.

2. The control actions are divided into two sets: those that determine the blockage of the plant, called critical, CA_c and the ones for whom an acceptable level of productivity is maintained CA_n. At the beginning, they are empty.

3. Then, the control actions are examined and classified in one of the two groups based on the following rule: if after removing an arc the root node is not linked to at least one leaf the corresponding control action, U_i is a critical one. $CA_c = \{ U_i \}$. Otherwise, $CA_n = \{ U_i \}$

4. At the end of the investigation, the elements (and the their starting nodes) belonging to CA_c are removed from the Reachable Tree (RT).

5. Starting from the simplified reachable trees, the boolean expression to define an arc of the simplified graph are derived. A boolean element is associated to each control action, than using a bottom-up approach, the RT is examined. Two consecutive actions are related by the operator or ($\vee$), while two actions starting from the same node by the operator and ($\wedge$).

6. The expression just found is linked with the boolean associated to the elements of CA_c by the or ($\vee$) relationship.

To better explain how these rules apply, first take the RT corresponding to the number 5 arc of the simplified graph as an example. It is represented in figure 4.16 (A), for simplicity it is presented a version already partially leafed. Eliminating individually the control actions U_{21} and U_{22}, the tree is no more connected, whereas doing the same with the others at least one path from M_3 to M_2 is still present.

$$CA_c = \{ U_{21} , U_{22} \} \quad , \quad CA_n = \{ U_{33} , U_{32} , U_{29} , U_{25} , U_{26} \}$$

Then (B) is obtained by removing the arcs belonging to CA_c.
After that by inspecting (C), the following expression is derived:

$$(B_{U26} \vee B_{U25} \vee B_{U29}) \wedge (B_{U32} \vee B_{U33})$$

Where B_{Ui} is the boolean associated to the i control action.
Finally, defining F_5 connect with the arc number five, it is obtained:

$$F_5 = B_{U21} \vee B_{U22} \vee (B_{U26} \vee B_{U25} \vee B_{U29}) \wedge (B_{U32} \vee B_{U33})$$
$$F_5 = TRUE \rightarrow \textit{arc 5 to be eliminated} \rightarrow \textit{system must be stopped}$$

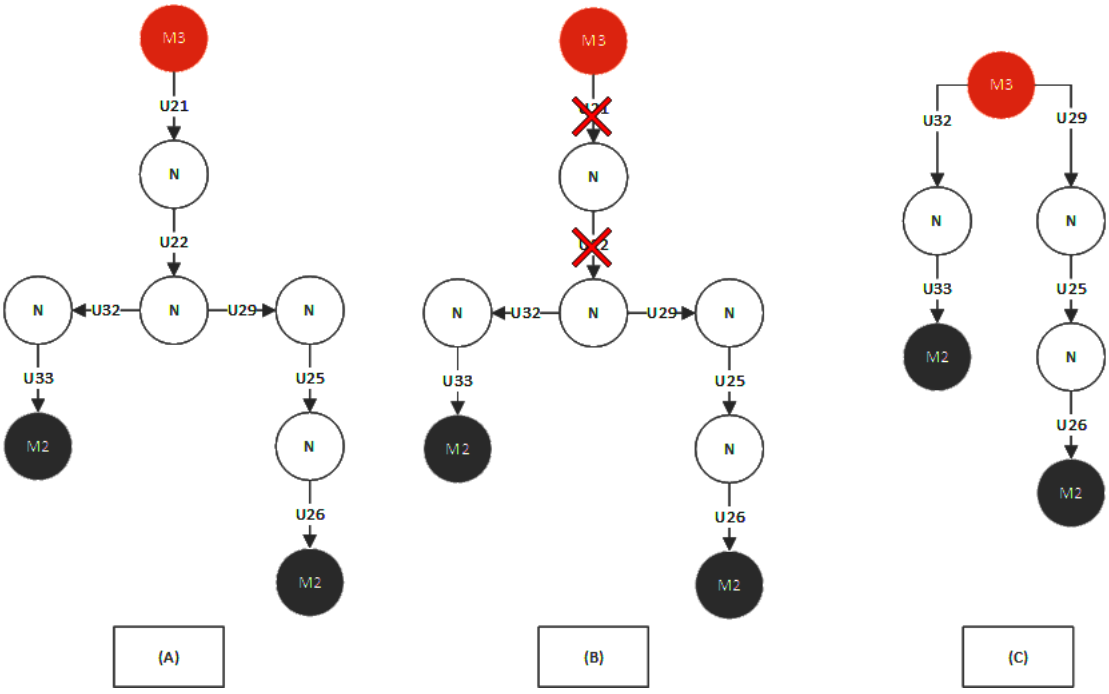

Figure 4.16: RT corresponding to path 5.

To provide another example, an RT with a slightly more complex structure is presented in figure 4.17. It represent the arc number two, and so the path from the testing machine to the load/unload station. Even in this case, the tree has been partially leafed to obtain a clear drawing. This example explains perfectly how even more complicated structures can be expressed in a simple way. Noting that the final part of each branch is identical, we can define the sub-tree p1 as:

$$p1 = B_{U4} \wedge (B_{U46} \vee B_{47} \vee B_{48} \vee B_{49} \vee B_{50})$$

Then, it is substituted into the RT to simplify its structure and to compact the expression of F_2. Repeating the same procedure of before, now three control actions have been individuated as critical from the moment that they are present in all the three branches of the tree and their elimination brings to a not connected reachable tree. Note that, even the events in $p1$ are repeated in all the possible evolution of the system but the control actions belonging to it do not break the RT if individually in fault. On the other hand, it could be possible to add to CA_c the boolean associated to $p1$ but the physical meaning of the critical control action set could be lost.
F_5 is then:

$$F_5 = p1 \vee ((B_{U31} \vee B_{U27} \vee B_{U37} \vee (B_{U38} \wedge (B_{U40} \vee B_{U51}) \wedge (B_{U34} \vee B_{U39} \vee B_{U40})$$

Thus, with the same procedure is possible to define all the F_i, $i = 1,..,6$, and calling is it possible to formalize the final check done by the high-level control system in order to understand if the system must be stopped:

$$(\ F_1 \ \vee \ F_2 \ \vee \ F_3 \ \vee \ F_4; \vee F_5 \ \vee \ F_6\ ) \ = \ TRUE \ \rightarrow \ \textit{system must be stopped}$$

So, basically, with a little effort in performing the off-line analysis, easily executable even for larger graphs, logical expressions, to be checked on-line with practically null computing cost, are derived.

Controller reconfiguration

MPC is very suited to implements fault recovery. In fact, by solving at any time instant a new optimization problem, allows one to include in the problem constraints which represent the occurrence of actuator faults.
When an actuator is in fault, the corresponding control variable is set to zero into the model. So when a fault has been detected following the action U_i, the high-level controller takes three countermeasures:

- The boolean variable associated to the faulted control action is setted to true, to implement the logic described above.
- A new set of inequalities are added to the model dynamically. New rows are added to A_{ineq} and b_{ineq}, computed in the controller initialization phase and stored in the memory, in such a way to include the constraints:

$$U_i(k) \leq 0$$

$$U_i(k+1) \leq 0$$

$$..$$

$$U_i(k+N) \leq 0$$

  Where N is the prediction horizon chosen.
  The goal is to exclude the problematic control action from the ones at the availability of the MPC algorithm.

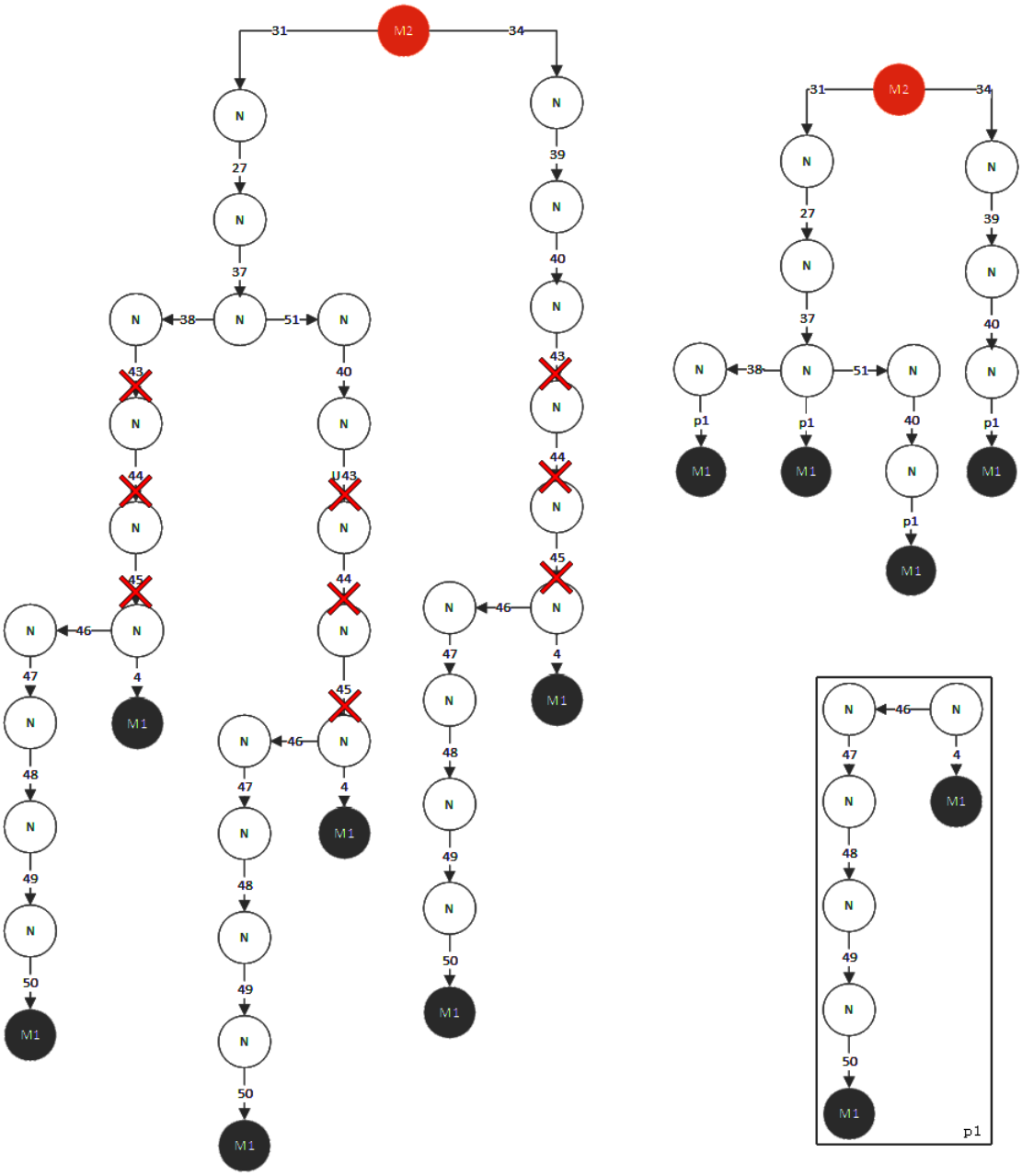

Figure 4.17: RT corresponding to path 2.

- A flag is raised to signal to ISaGRAF that a fault that has to be handled has been detected.

To do that, the function *faultsHandler* implemented for the fault detection has been integrated. Thus, instead of simply reporting to the operator that a fault just occurred, for each element of ca_fault it implements these three points as follows:

```
if ((ca_fault[i] > 0) AND (previous_ca[j] == 1))
if (ca_fault[j] > 0) {
cout << ACTUATOR FAULT << endl;
AineqFault= newMatrix;
AineqFault= constraintDefinition;
Aineq = concatenateMatrices(Aineq, AineqFault);
updateModel();
raiseFlagIsagraf[i] = 1;
} else {
cout << SENSOR FAULT << endl;
}
```

These three actions are enough to reconfigure controller.
It is important to underline that the reconfiguration of the model of the HLC, adding the constraints, alone is not sufficient to make the recovery mechanism effective. It is also necessary to correct the state of the system memorized by the high-level control so that the control actions are consistent with the faulted system. As previously described, before passing the control actions to LLC, the controller checks the status on which they have been calculated and the status passed to it by the line supervisor, if they are different the actions are dropped. Therefore even the model of the line supervisor must be changed. For this reason, a new text file, in which the informations about which control action has just faulted, has been created. The line supervisor reads it from this file and changes the state of the transport line that has in memory to make it consistent with the real situation.
In this way, the fault has been totally recovered.

Note that for implementing the tolerant control algorithms, two different file text have been created: one written by the line supervisor which contains a sector of boolean indicating what sequences have presented problems, while the other written by the DCPIP which contains an indication about what actions have been disabled due to an actuator fault.

Table 4.8: List of control actions and control sequences that composing them

U1	S2M1
U2	S3M1 S1M2
U3	S26M1 S5M13
U4	S27M1 S7M13
U5	S28M1
U6	S36M1
U7	S2M2
U8	S3M2 S4M3
U9	S7M2 S5M12
U10	S5M2 S7M12
U11	S2M3
U12	S3M3 S1M4
U13	S24M4 S12M5
U14	S7M4 S27M8
U15	S26M8 S5M4
U16	S13M5
U17	S7M5 S25M6
U18	S6M6 S5M7
U19	S2M6
U20	S3M6 S9M8
U21	S2M7
U22	S3M7 S9M9
U23	S18M7
U24	S1M7
U25	S10M8
U26	S21M8
U27	S20M8 S12M10
U28	S16M8 S14M9
U29	S17M9 S15M8
U30	S6M8 S35M9
U31	S31M9 S28M8
U32	S10M9
U33	S22M9
U34	S23M9 S12M11
U35	S30M9
U36	S29M9
U37	S13M10
U38	S7M10 S33M3 S12M12
U39	S13M11
U40	S7M11 S34M10 S33M3 S12M12
U41	S8M11
U42	S32M11
U43	S13M12
U44	S11M12 S12M13
U45	S13M13
U46	S11M13 S12M14
U47	S13M14
U48	S7M14 S5M15
U49	S2M15
U50	S3M15 S19M1
U51	S37M10 S5M11

Chapter 5

Conclusions

The Thesis has described the improvement of an MPC controller for the transport line for de-manufacturing systems. In particular, two themes have been addressed: the reduction of the times needed to perform a control action and the implementation of a system of fault detection and recovery.
Through the implementation of advanced control techniques, such as MPC with control horizon, combined with the application of heuristic rules developed thanks to an in depth study of the structure of the line, clear improvements have been obtained from the point of view of lowering the computing power required. This resulted in a significant reduction in the average time needed to calculate the solution, which in some cases reached more than 96% compared to the basic version of the controller from which the work started. Moreover, thanks to the parallelism introduced between the problem-solving activity and movements realization, even the total production time has been sensibly reduced and for the most of the transition the optimization problem has been completely hidden. It means that the first topic of this Thesis has been totally solved and the bottleneck of the production time is now due to the needed to implement the movements of the pallets, which requires a fixed time.
Concerning the detection and the handling of the faults occurring on the plant, the case of single failure has been studied. A knowledge-based model for the definition of the residual matrix building and an opportune method to active fault recovery have been implemented with very good results. The system is now able to react to failures with only two steps of delay, while satisfying the constraint of not wasting time in reading the state of the transport line by the sensors.

All the improvements achieved have been tested on the pilot plant of the CNR-ITIA, with very good results.

A possible continuation of this Thesis could therefore be the attempt to reduce the steps necessary for the controller to detect a fault. Furthermore, fault recovery is quite limited by the configuration chosen for the transport modules. Studying another plant configuration that keeps the distance between the machines unchanged but adds connections to the graph could prove to be fundamental.
An important limitation to the control system is given by the solver. In fact a more effective configuration for the Cplex environment must be implemented to solve the bug of saturation of the workstation in which the main process of the solver runs.

Bibliography

[1] J. Bang-Jense and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag, 2007.

[2] A. Cataldo. *Model Predictive Control in Manufacturing Plants*. PhD thesis, Politecnico di Milano, 2016.

[3] A. Cataldo and R. Scattolini. Logic Control Design and Discrete Event Simulation Model Implementation for a DeManufacturing Plant. *[online] Available: http://automazione-plus.it*, 2014.

[4] A. Cataldo and R. Scattolini. Dynamic Pallet Routing in a Manufacturing Transport Line With Model Predictive Control. *IEEE Transactions on Control Systems Technology*, 24:1812–1819, 2016.

[5] E.F. Chamacho and C. Bordons. *Model Predictive Control*. Springer, 2007.

[6] G. Csardi and T. Nepusz. *Igraph, Refence manual*.

[7] S.M. de Oca and V. Puig. Fault-tolerant control design using a virtual sensor for LPV systems. In *Proc. Conference on Control and Fault-Tolerant Systems (SysTol)*, pages 88–93, Nice, France, 2010.

[8] M. Colledani G. Copani, A. Brusaferri and Others. Integrated De Manufacturing systems as new approach to End of Life management of mechatronic devices. In *Proc. of the 10th Global Conference on Sustainable Manufacturing*, volume N, Istanbul, Turkey, 2012.

[9] J. Gertler. *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker, Inc., 2 edition, 2007.

[10] IBM ILOG. *CPLEX Optimization Studio CPLEX Users Manual*.

[11] ICS Triplex ISaGRAF Inc. *ISaGRAF, Getting Started*. 2009.

[12] R. Isermann and P. Ball. Trends in the application of model based fault detection and diagnosis of technical processes. In *Proc. of the 13th IFAC World Congress*, volume N, pages 1–12, Piscataway, New Jersey, USA, 1996.

[13] L.Q. Lee J. Siek and A. Lumsdaine. *The Boost Graph Library, User Guide and Reference Manual*.

[14] E.L. Russell L.H. Chiang and R.D. Braatz. *Fault Detection and Diagnosis in Industrial Systems*. Springer, 2 edition, 2007.

[15] R.M. Lima and I.E. Grossmann. Computational advances in solving Mixed Integer Linear Programming problems. *AIDAC*, pages 151–160, 2011.

[16] G. Copani M. Colledanim and T. Tolio. De manufacturing Systems. *Procedia CIRP*, 17, 2014.

[17] J. Maciejowski. *Predictive Control with Constraints*. Pearson Prentice Hall, 2002.

[18] D. Mayne and J.B. Rawlings. *Model Predictive Control: Theory and Design*. Nob Hill, 2009.

[19] R.K. Mehra and J. Peschon. An innovation approach to fault detection and diagnosis in dynamic system. *Automatica*, 7:637–640, 1971.

[20] A. C. Raich and A. Cinar. Statistical process monitoring and disturbance diagnosis in multivariable continuos processes. *AIChE J.*, 42:995–1009, 1996.

[21] C. Sanderson and R. Curtin. *API Documentation for Armadillo*.

[22] R. Scattolini. Architectures for distributed and hierarchical model predictive control, a review. *Journal of Process Control*, 19:723–731, 2009.

[23] R. Scattolini. *Advanced and Multivariable Control*. Pitagora Editrice Bologna, 2014.

[24] J.P. Vielma. Mixed Integer Linear Programming Formulation Techniques. *SIAM REV*, 2015.