POLITECNICO
MILANO 1863

Scuola di Ingegneria Industriale e dell'Informazione
Master of Science in Automation and Control Engineering

Hierarchical Model Predictive Control of series production plants

Supervisor: Prof. Lorenzo Mario FAGIANO
Co-supervisor: Prof. Riccardo SCATTOLINI

M.Sc. Thesis by:
Luca CIAVARELLA Matr. 905952

Academic Year 2019-2020

Abstract

This thesis deals with the control of an automated manufacturing plant, the theory behind the control algorithm is Model Predictive Control.
The plant under study is a manufacturing system with one or more “nodes”; each node is composed of a first machine which carries out a specific process (machine “M” – Manufacturing) followed by a second machine which is responsible for testing the result of the process just performed (machine “T” – Testing).
In the case that a part does not pass the test successfully, it will be moved in a buffer (br) waiting to be reworked by the same machine as before.
The plant includes also a heuristic algorithm that has the task to verify at every instant if machine M can work raw parts or if it must do the setup to rework from br. The decision of the heuristic algorithm is determined by the number of parts contained in buffer br, that has to be always smaller than a predefined value.
The MPC algorithm, implemented in three different approaches, has the task to decide every instant of time if the parts contained in br have to be reworked; in other words, MPC has the possibility to modify, in real time, the limit value of parts in br.
The obtained results show that Model Predictive Control is a good method to control this kind of manufacturing plant, the algorithm designed in this work has the capability to avoid deadlocks and optimize the production time.

Keywords: Model Predictive Control, manufacturing, optimization, rework, simulation, plant.

Sommario

Questa tesi si occupa del controllo di un impianto di produzione automatizzato, la teoria alla base dell'algoritmo di controllo è Model Predictive Control.
L'impianto in analisi è un sistema di produzione costituito da uno o più "nodi"; ogni nodo è composto da una prima macchina che esegue una lavorazione specifica (macchina “M” - Manufacturing) seguita da una seconda macchina che è responsabile del test del risultato del processo appena eseguito (macchina “T” - Test).
Nel caso in cui un pezzo non superi correttamente il test, la parte lavorata difettosa verrà spostata in un buffer (br) in attesa di essere rilavorata dalla macchina precedente.
L'impianto include anche un algoritmo euristico che ha il compito di verificare ogni istante se la macchina M può lavorare pezzi grezzi o se deve eseguire la procedura di setup per rilavorare da br. La decisione dell'algoritmo euristico è determinata dal numero di pezzi contenuti nel buffer br, che deve sempre essere minore di un valore predefinito.
L'algoritmo MPC, implementato in tre diversi approcci, ha il compito di decidere in ogni istante di tempo se i pezzi contenuti in br devono essere rilavorati; in altre parole ha la possibilità di modificare, in real time, il valore limite che comporta il setup.
I risultati ottenuti dimostrano che il Modello Predictive Control è un buon metodo per controllare questo tipo di impianto di produzione, l'algoritmo progettato in questo lavoro ha la capacità di evitare deadlock e ottimizzare i tempi di produzione.

Parole chiave: Model Predictive Control, produzione, ottimizzazione, rilavorazione, simulazione, impianto.

Acknowledgement

I would like to thank all those who have helped me reaching this important goal and all the people who supported and advised me, in particular:

Professor Fagiano, supervisor of this thesis, for the opportunity to delve into this topic and for the help provided.

Professor Scattolini, co-supervisor of this work, for having shared with me his knowledge and his advices that have always been helpful.

My family, because has been able to support and to believe in my abilities during all this journey; I am grateful because they always taught me to do my best.
My grandparents, in particular nonno Luciano who made me discover the beauty of science and nonna Anna that always believes in me.

Giorgia, my great supporter, for having always been close to me and for making me overcome difficult moments by encouraging me to go on.

The big family of Piazza Anita, where each component, has been able to make me grow and accompany me.

Andre, Anna, Claire, Chili, Fede, Jack, Simo, Robi, Sa, Saibe and Tommi because with their happiness and nearness made me live in these years many great and supportive friendships.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Goals

The main purpose of this thesis is to optimize production time in an automated manufacturing system using Model Predictive Control (MPC) technique.

In a global market it's mandatory to maximize productivity and efficiency to maintain competitiveness, it's also essential to pay attention to energy savings both for economic and for environmental reasons that in recent years are gaining particular importance.

For these reasons, in industry, many resources are dedicated to finding management and control solutions going in this direction and Model Predictive Control, among the variety of control techniques, is an advanced strategy that allows to solve optimization problems accounting for scheduling, lot sizing and energy consumption objectives.

The manufacturing system modelled in this thesis is a production plant which presents one or more machines in series executing different working procedures. Each machine is followed by a testing machine that has the task to verify the quality of the job just performed, every pair of manufacturing and testing machines is called "node", the plant can be composed of one or more nodes.

The core of the problem is that every manufacturing machine can choose if is better to work a raw part or rework an old one which didn't pass the test, and is added to a buffer. This

choice, as will be demonstrated later, can produce different effects in terms of production time and consequently energy consumption.

The entire plant is assumed to be controlled by a heuristic algorithm that, in a simple way, decides at each time step the best choice for each machine, as will be demonstrated this algorithm does not provide satisfactory performance especially in stressed conditions.

Figure 1 presents an example of a node: parts flow from left to right and if they do not pass the test (machine T) at the end of the job, they wait to be reworked in buffer *br*.

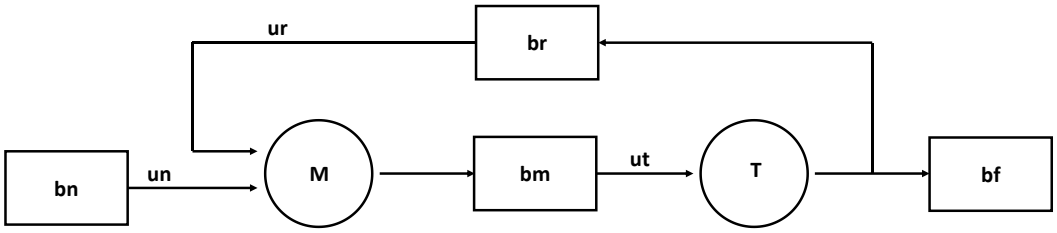

Figure 1: Node example

The role of the heuristic algorithm chosen as "benchmark" is to receive as input the state of the plant and to return as output the best commands for the machines, in particular the algorithm has a value called θ that corresponds to the limit of parts buffer *br* can contain until has to be emptied. In the benchmark solution, the value of θ is fixed. The resulting control system is shown in Figure 2.

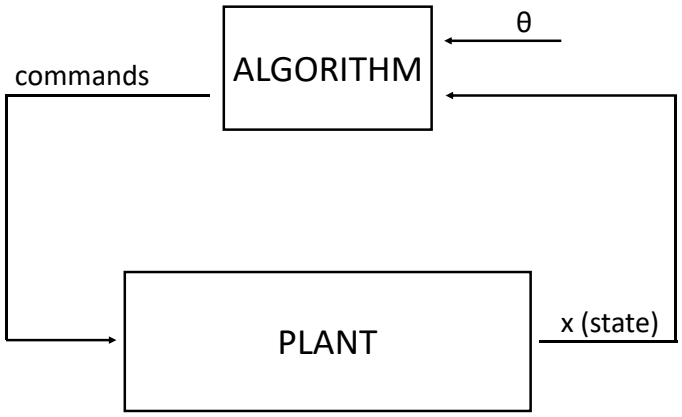

Figure 2: Benchmark system without MPC

The MPC algorithm is added to the system just illustrated without modifying it but taking care of the control at a higher level, so the role of MPC is to find the best θ at any time instant modifying its value in real time and providing the result of the optimizing problem to the benchmark.

Figure 3 shows how MPC algorithm interacts with the whole system. Such a control strategy is hierarchical: the MPC law acts as a supervisor for the implemented heuristic.

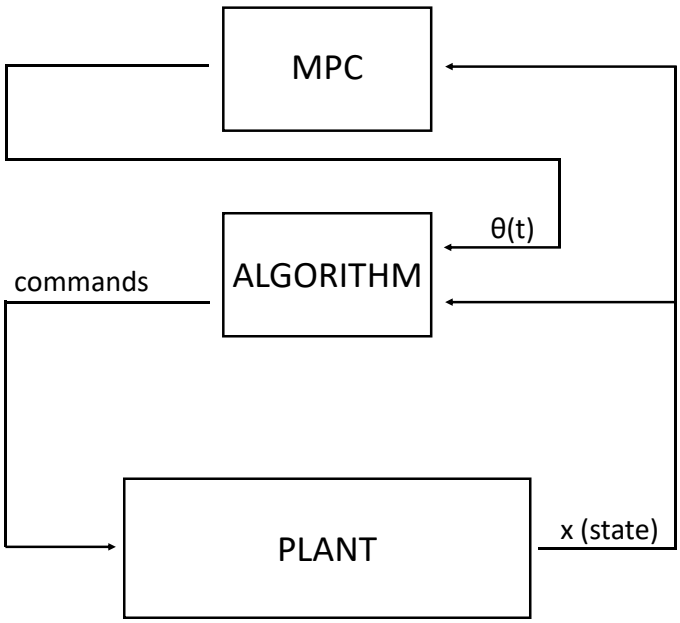

Figure 3: Role of MPC in the system

1.2 Developed MPC Approaches

In this thesis project, MPC algorithm has been implemented following different approaches and different levels of complexity. Three versions of the algorithm have been developed and tested.

A common feature of the MPC algorithms is that they do not return an arbitrary value of θ, but they can modify the actual value by increasing or decreasing it of one unit, otherwise they can obviously maintain it the same as the previous instant.

According to the receding horizon approach, after having evaluated the best sequence of θ along the prediction horizon, the plant will apply only the first value of the sequence, then in the following instant the MPC algorithm will be recalled returning a new value of θ.
Main difference between the three versions is the way how is built the best sequence of θ along the prediction horizon, the three approaches will be quickly presented:

- MPC V1: is the simplest version of MPC, when the algorithm is called it simulates the behaviour of the system generating three sequence of same θ along the prediction horizon (θ_old – 1, θ_old, θ_old + 1). If the plant is composed by more than one node, the algorithm will return the same θ for every node.
  Sequences evaluated every instant: 3.
- MPC_V2: in this case the algorithm evaluates the best combination of different θ along the prediction horizon (PH). As in the previous case, if the plant is composed by more than one node the algorithm will return the same θ for every node.
  Sequences evaluated every instant: 3^{PH}
- MPC_V3: this is the most complex version; it is valid only for plants composed by more than one node. The sequences of θ are generated as in MPC_V2 but, unlike in the previous case, here every node receives its best θ. In other words, every node is optimized on its own.
  Sequences evaluated every instant: #nodes*3^{PH}

1.3 State of the Art

This section introduces at the state of the art of MPC control techniques in manufacturing, main topic of this thesis.
Regarding manufacturing plants different studies and researches have been taken because as anticipated in Section 1.1, it's very important to optimize production to be competitive in global market.
About energy efficiency, MPC gave a great contribution (see [1], [2], [3]) because of its feature to have an optimization function related to the future behaviour of the system.
This function in fact can contain, between the various constraints, also data regarding energy consumption.

Great attention in research, related MPC theory, was dedicated to Automated Storage and Retrieval Systems; the reason is that in a lot of situations this kind of systems are controlled by heuristic algorithms [4].

In this field an important research was published [5]; despite this work is related parallel machines, was important for the present thesis because deals with reworks and scheduling of production.

There are two major categories that characterize the structure of manufacturing plants: with machines in parallel and with machines in series; these two categories determine different problems and different control techniques starting from their need and priorities.

About series of machines an important research has to be mentioned; So and Tang in [6] studied an algorithm that computes, before a processing, the optimal batch dimension in a manufacturing plant with random rework.

The difference in this thesis is that will be presented an online MPC algorithm that every instant can evaluate the best behaviour of the plant.

1.4 Original Contributions of this Thesis

Based on what explained in the previous sections, the main original contributions of this thesis are:

- Mathematical modelling of a customizable automated production plant composed by manufacturing and testing machines.
- Design and development of a heuristic algorithm used to control the plant previously presented.
- Design and development of three different MPC algorithms with the role of real time optimization and deadlock avoidance
- Testing of all the developed algorithms in simulation, and analysis of the obtained results.

The first part of the work was focused on developing a realistic mathematical model for the plant, it’s a discrete-time model completely customizable.

Before starting a simulation it’s necessary to provide information about how to build the system, i.e the number of nodes, size of buffers, times to work, test and setup, time to move one part from the testing machine to buffer of parts to be reworked and so on.

A simulation is based on the number of parts the user wants to work, so when all parts have been worked simulation finishes.

Implementing the heuristic algorithm was the second part of the work of this thesis, this algorithm is basic for the correct operation of the system. Goal of the algorithm is to complete the entire working cycle following instructions it contains in its code, it's not programmed to optimize production but only to bring it at the end.

The last and most important part of the work was to implement the optimization part which is contained in the three MPC algorithms. After identifying the three approaches to build the algorithms, they were implemented in different ways. Common features between the three algorithms is the cost function evaluating the best θ to return, the cost function has the goal to permit the algorithm to find the sequence of θ that along the prediction horizon move forward many parts as possible and performs the least number of setups.

Results will be presented in detail in Chapter 4, in general it's possible to conclude that MPC algorithm is robust against deadlocks and bottlenecks management compared to the heuristic algorithm that needs to receive the right θ at the beginning of a simulation.

MPC algorithms implemented in this work can improve production time by a limited percentage factor.

In conclusion MPC could be a solution to optimize production for a plant like the one in analysis in terms of better and more robust management.

1.5 Thesis Outline

- Chapter 2 deals with the modelling of the plant and development of the heuristic algorithm.
- In Chapter 3 are presented the three MPC algorithms.
- Chapter 4 deals with the conclusions of this thesis and some considerations about future works.

Chapter 2

System Model

2.1 Introduction

The model created, as anticipated in the introductory part, refers to a generic automatic industrial production plant; the main features of the system being analysed are the presence of one or more "nodes" in series.

Each node is composed of a first machine which carries out a specific process (machine “M” – Manufacturing) followed by a second machine which is responsible for testing the result of the process just performed (machine “T” – Testing).

The entire model is discrete-time, variable *k* corresponds to time.

For simplicity in the dissertation and for the significantly different level of complexity, the system is considered first in a single node configuration and secondly in a multi node configuration.

2.2 Single Node Configuration

The single node configuration (Figure 4) consists of two machines (M and T) and four buffers (*bn, bm, br, bf*), containing respectively raw parts, worked parts, fault parts and finished parts. Boolean variables *un, ur* and *ut* indicate activation of the movement of the parts and are enabled by the algorithm that controls the system.

At the end of a process, be it in M or in T, the part leaving the machine is automatically moved in the reference buffer, so from M parts will be moved in *bm* while from T, depending from test result, parts will be moved in *br* or in *bf*.

All the movements from a buffer to a machine and vice versa are considered instantaneous except for the movement of a part from T to *br* which instead presents a delay.

The main reasons for inserting this delay in the model are two: the first is that it is quite likely that more time is needed to bring parts back along the chain, the second is that, through this delay, the MPC algorithm will be able to predict with certainty the behaviour of the node after the result (a priori uncertain) of the test.

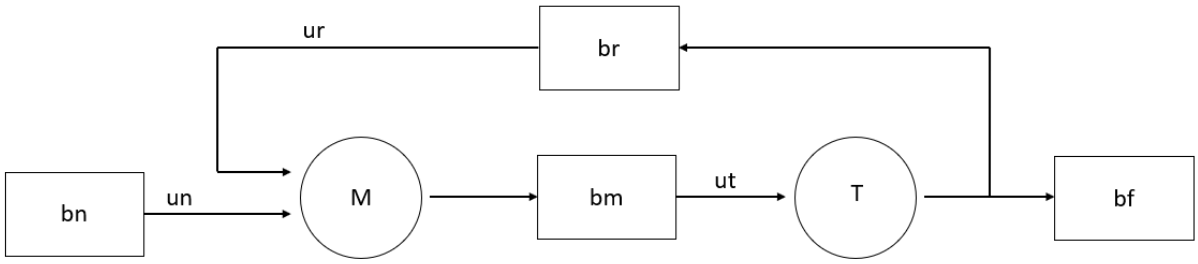

Figure 4: Single node configuration

The buffers *bn* and *bf* which contain the raw and finished parts respectively are considered simply as counters that assume a value equal to the number of parts they contain.

Buffers *bm* and *br* are vectors of predefined size by the operator and in this way oblige the control algorithm to avoid any overloads and bottlenecks, the buffer *bm* knows the origin of the parts it contains.

Every position of the buffer *bm* can assume value 0, 1 or 2 depending on whether it is empty, occupied by a part worked only once or occupied by a reworked part.

In non-critical conditions the *bm* and *br* buffers work according to the "first in first out" (FIFO) logic, this means that the first part that enters the buffer will be the first to exit; as it will be illustrated later, only in some situations the control algorithm has the possibility to violate this rule.

2.2.1 Machine M: Finite State Automaton

Machine M can be represented through a finite state automaton (Figure 5) and considers six different states: L1 (free, position 1), L2 (free, position 2), W1 (raw part processing), W2 (rework of damaged part), S1 (setup 1), S2 (setup 2).

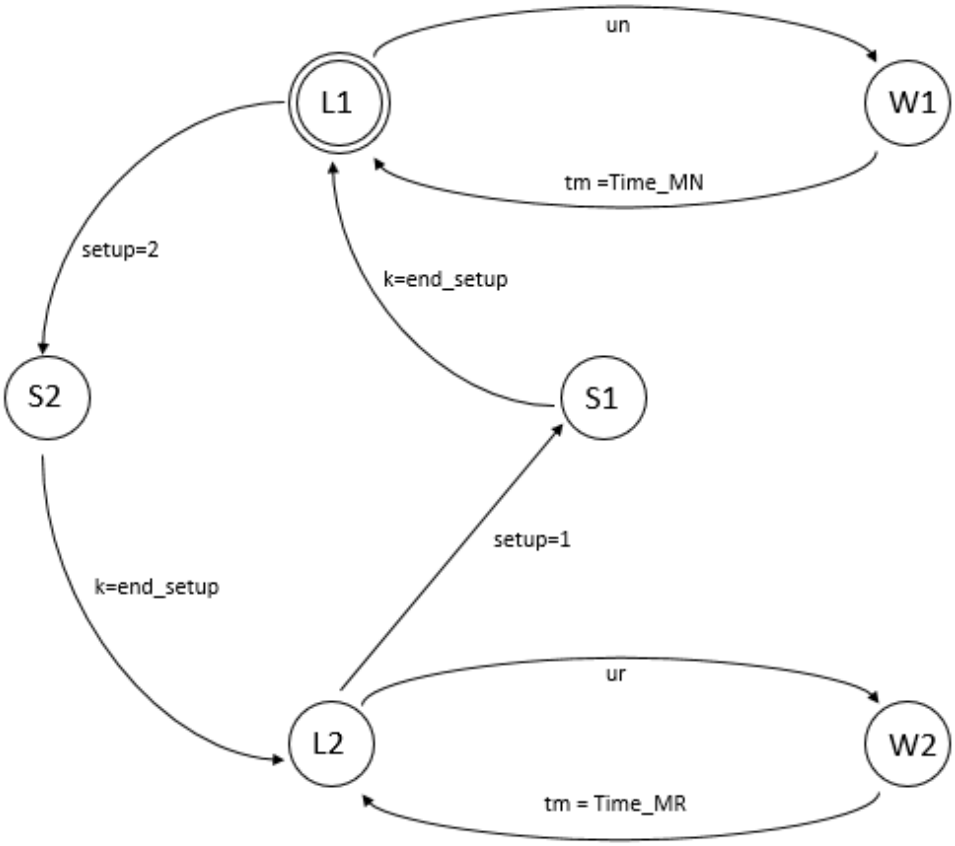

Figure 5: Finite state automaton machine M

From the starting state L1 machine M can enter in state W1 (working raw part) enabling the command *un* that, as written in section 2.2, takes care of activating the movement of one part to the machine. Otherwise starting from state L1 as before, machine M can start the setup procedure to be able, at the end of the process, to work on parts that have already undergone processing and have previously failed the test.

The machining procedures, that take place in states W1 and W2, end when the time necessary for the machining itself has elapsed, note that the model has the possibility of considering different times depending on whether a new part or an already worked part is being processed

(Time_MN and Time_MR). Similarly to the previous situation, from state L2 it is possible to carry out the setup procedure to subsequently start working on raw parts.

The transitions are of two different types: transitions controlled by the algorithm and transitions determined automatically at the end of a process. Transitions *un*, *ur* and *setup* belong to the first category while the other ones (*k = end_setup, tm = Time_MN* and *tm = Time_MR*) belong to the second category and are related to the variable time.

2.2.2 Machine T: Finite State Automaton

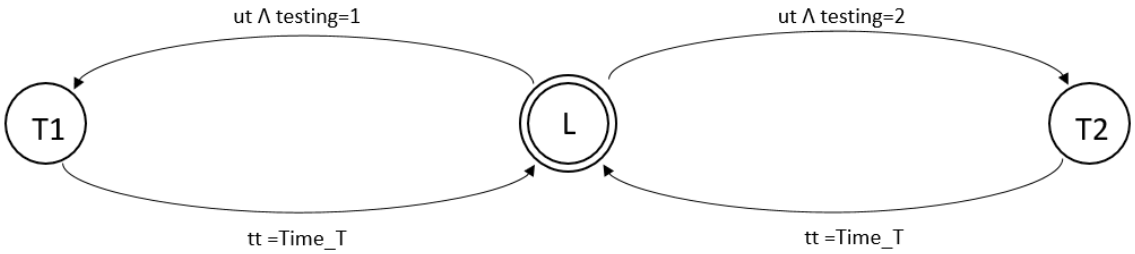

Figure 6: Finite state automaton machine T

Unlike machine M, machine T is simpler (Figure 6); it is initialized to the L (free) state and when the control algorithm verifies that there are conditions for testing a part, it enables the *ut* variable together with the *testing* variable that indicates whether a new or reworked part is being tested.

The information about the type of tested part (new or reworked) is necessary because the hypothesis that has been made is that this condition entails a different probability of success of the test.

More precisely the probability with which a generic part will pass the test can be less than one only the first time a part is worked, while for reworked parts the test is passed with probability one.

Before starting to test a part, the machine checks that there is place in the two buffers where it is possible that the part will have to be placed at the end of the test, therefore both *br* and *bf* must not be full, this in case is tested a new part.

If the machine has to test a reworked faulty part, since it will be certain that it will obtain a positive result in the test, the availability check in the buffers will only take place against *bf*.

2.3 Multi Node Configuration

In the multi node configuration (Figure 7, Figure 8, Figure 9) it was necessary to differentiate the first and the last node from all the others within the chain, the reason is that, as in the single node configuration, the first buffer *bn* and the last *bf* are counters that assume a value equal to the number of parts they contain.

In a similar way to the previous configuration, all the buffers excluding those just mentioned are vectors with a length predefined by the operator.

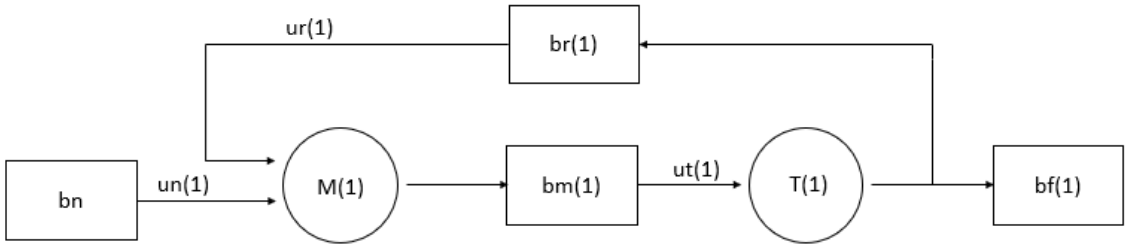

Figure 7: First node

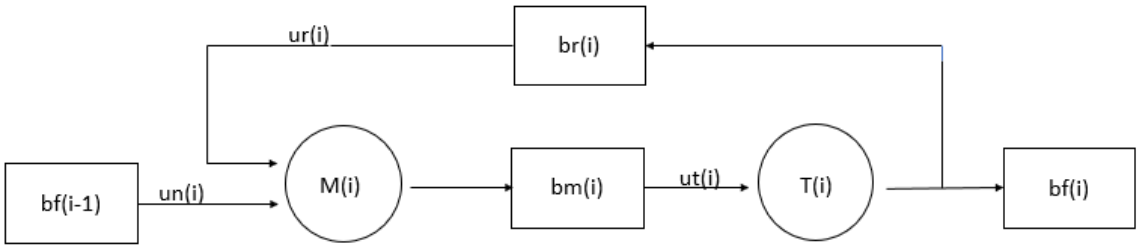

Figure 8: Generic i-th node

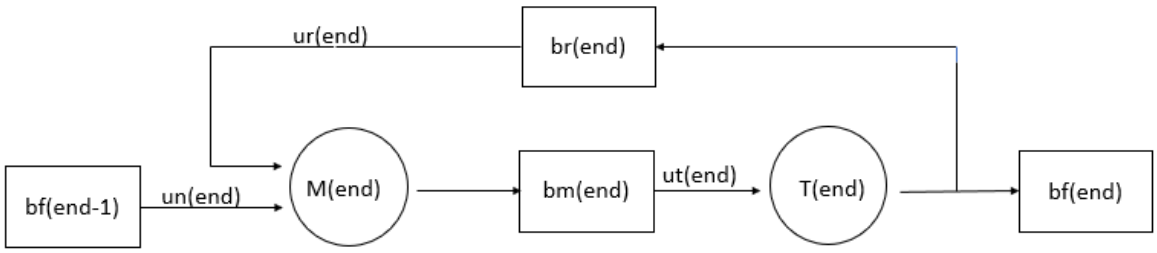

Figure 9: Last node

In this configuration it is useful to remember and better explain a concept anticipated in paragraph 2.2 i.e. the FIFO rule that is used to manage the operation of the buffers and how this rule can be violated in particular cases. The case in which the machine has the possibility to not respect the required logic is shown in the example in Figure 10.

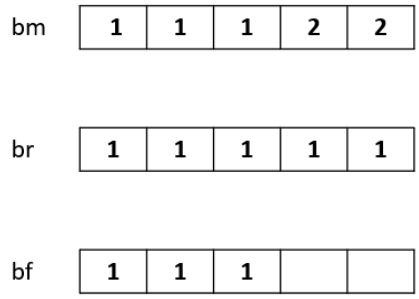

Figure 10: Example of a FIFO rule violation

In this example, where buffers can contain a maximum of five parts, the situation looks like this: the buffer *bm* is full and contains both raw parts worked only one time (value 1) and parts that didn't pass the test and have been reworked (value 2) , the buffer *br* is full while *bf* still has free positions.

According to FIFO logic, machine T in the L state should pick up the part of *bm* which is in first position to test it but is unable to do so because the part it should test would be a part worked only once and being full *br*, as illustrated in paragraph 2.2.2, it's necessary to avoid the risk of having to send a part in an already full buffer, in the case the test will have negative result.

In this situation, since the buffer *bf* is instead not full, the machine T has the possibility to search within *bm* for a reworked part (value 2) and test it immediately, being sure that at the end of the test the part itself will be placed in the buffer *bf*.

There are no other differences to report between this configuration and the previous one, obviously in this case the level of complexity becomes considerably higher because this type of structure is more likely to generate slowdowns and bottlenecks.

2.4 Heuristic Algorithm for Machines Management

The algorithm that is presented in this paragraph is the one that has been chosen as the "benchmark" to evaluate any performance improvement by inserting the MPC algorithm. As anticipated in the introductory chapter, this algorithm behaves quite simply, based mainly on the quantity of parts in the buffer *br* (value θ): if the number of parts in *br* will be greater than θ then the machine will have to rework the failed parts to avoid full filling the buffer itself.

In this benchmark, the value of θ is chosen at the beginning of the simulation and remains constant for the entire duration of the process. This will be different in the MPC approach, where the controller will be able to change θ at each time step by solving a finite horizon optimal control problem (see Chapter 3).

It is possible to set different values of θ between the various nodes.

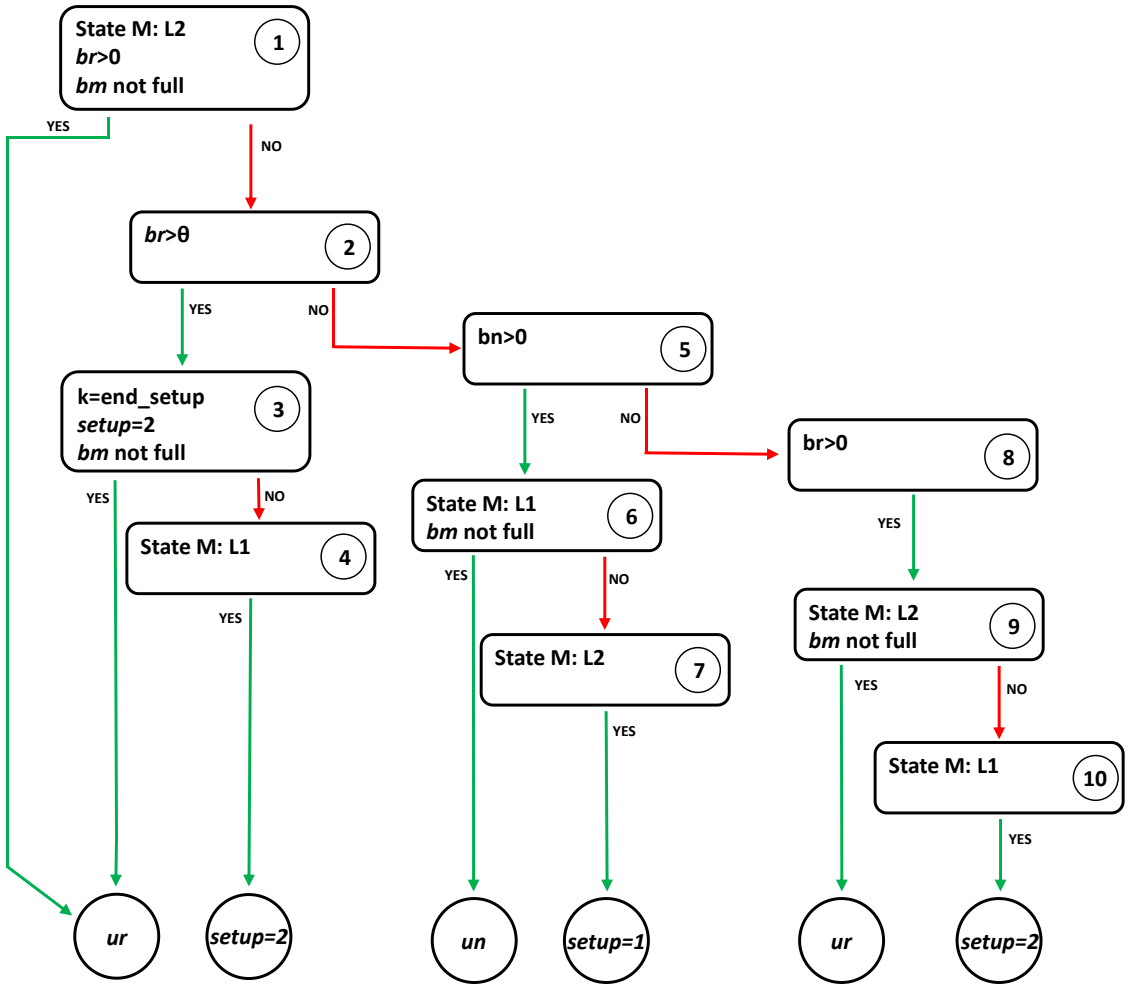

Figure 11: Heuristic algorithm, machine M

Figure 11 presents the decision tree of the algorithm: it is composed of ten conditions and depending on whether they are respected or not different paths are taken, the conditions represented in the rectangles with rounded corners can be true or false.

As illustrated in paragraph 2.2.1, the possible commands that the algorithm can deliver to machine M are three: *ur*, *un* and *setup*.

By observing the tree it is possible to identify three branches that develop vertically, starting from the left the first branch that is visited by the algorithm (conditions: 1,2,3,4) is used to consider the processing of faulty parts, it is important that the first branch is dedicated to the buffer *br* because it risks, if completely filled, to block the functioning of the whole system. The second branch (conditions: 5,6,7) evaluates the processing of raw parts, therefore if the parts in *br* are less than θ, while the third branch (conditions: 8,9,10) deals with considering the situation in which there are no parts in buffer *bn* and therefore only the parts left in *br* remain to be processed, even if they are less than θ.

The first condition of the scheme has the purpose of allowing the machine M to work all the parts that are present in the buffer *br*, thus completely emptying it before eventually commuting back to work on raw parts. Therefore, when *br* contains more elements than the parameter θ, the machine will perform the setup to rework parts, and once the setup is finished it will empty the entire buffer before returning to work new parts.

It should also be noted that all the conditions preceding the enabling of the *ur* or *un* variables, which therefore lead to the actual start of a process (conditions: 1,3,6,9), check that the *bm* buffer is not full.

The algorithm in "simple" conditions, i.e. when has to work a small batch or when times are distributed so as not to create bottlenecks, achieves very satisfactory performance in terms of the total production time of a batch, however problems arise when the processing times are very heterogeneous and therefore tend to generate bottlenecks that the algorithm cannot handle.

Part of the algorithm is also the one that deals with managing the operation of the T machine. This part of the code is much simpler, the decision tree is shown in Figure 12.

Obviously also the tree is less complex than the one that manages the machine M; the final command is in fact always the same the difference is that with ut(1) we mean the operation of testing the first part available in *bm*, while with ut(x), as can be seen from the way in

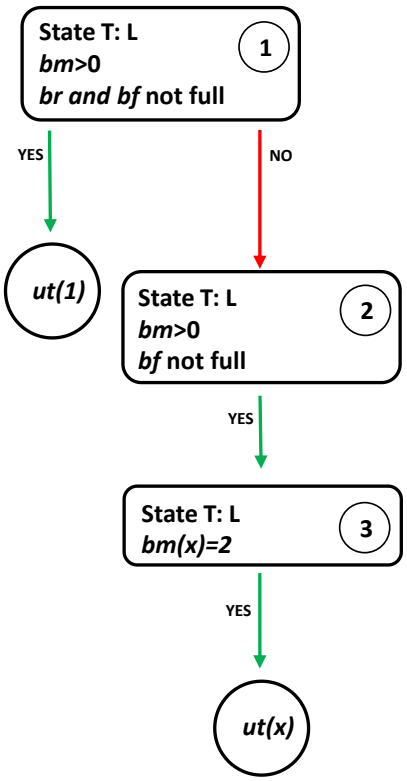

Figure 12: Heuristic algorithm, machine T

which the right branch develops, means the operation of testing the first part already reworked which is in *bm*, if present obviously (violation of the FIFO rule).

2.5 Heuristic Algorithm: Simulation Results

This section describes the results of numerical simulations executed to test the benchmark algorithm. Tests have been executed both on single node and multi node configuration, each test shows the strengths and weaknesses of the algorithm.

2.5.1 Test 1: Single Node

- Number of parts: 100
- Buffer dimensions: 5 parts
- $\theta = 4$
- Time machining raw parts: 1
- Time machining fault parts: 1
- Time testing: 1
- Probability test passed: 80%
- Moving time from T to *br*: 10
- Setup time: 40

In the following test (Figure 13) is possible to see how the algorithm can work in "normal" conditions, at the beginning it starts to work raw parts and when the number of parts in buffer *br* reaches value θ, machine M starts to do the setup.
Setups are visible in the graph because in those cases both buffer *bn* and *br* remain constant or increase, obviously during setup the number of parts in those buffers can't decrease.
In this test θ was chosen equal to the maximum value it can take so buffer dimension minus one, this value permits to have maximum performance from the algorithm because minimizes the total number of setups during the cycle and setups are the most important waste of time.
In this conditions buffer *bm* can always work off the parts without accumulating them.

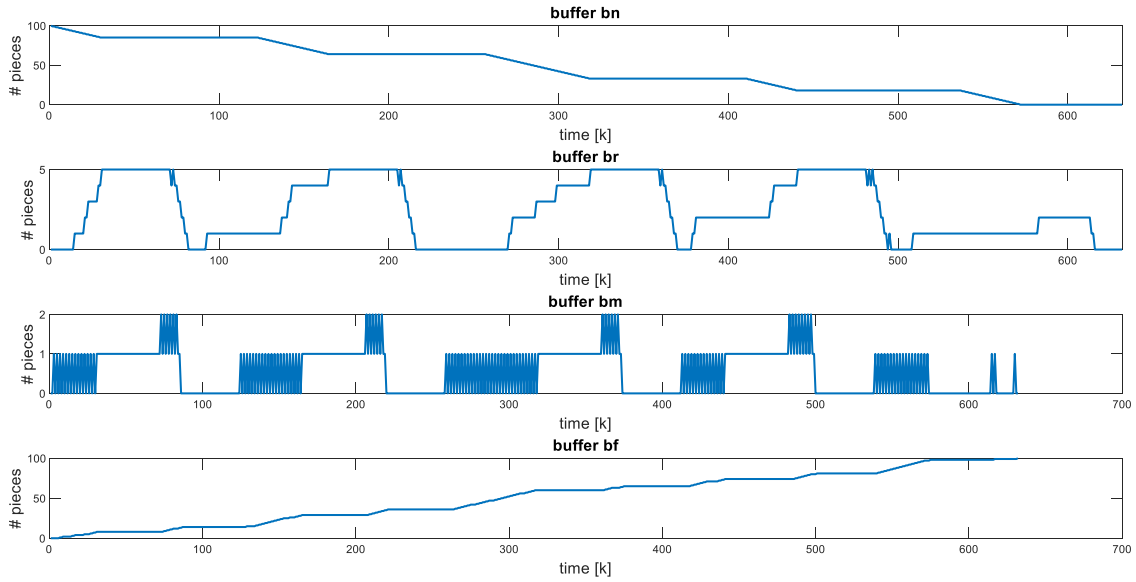

Figure 13: Test 1 – buffers behaviour

2.5.2 Test 2a: Single Node, deadlock

- Number of parts: 100
- Buffer dimensions: 5 parts
- $\theta = 4$
- Time machining raw parts: 1
- Time machining fault parts: 1
- Time testing: 5
- Probability test passed: 80%
- Moving time from T to *br*: 10
- Setup time: 50

Test 2a (Figure 14) have been executed to show the first limit of the algorithm, the difference from Test 1 is that in this case the time for testing a part is increased to accumulate parts in buffer *bm.* At the beginning all works perfectly, the algorithm waits the growth of the number of parts in *br*; the problem is that when *br* reaches value θ and M starts setup it's too late because *bm* is full.

During setup, machine T can't test because it contains parts worked only once and it is programmed to avoid the risk to send parts in the already full buffer *br* (in case of negative test result).

When setup finishes both *bm* and *br* are full so that nothing can be done to continue the working process.

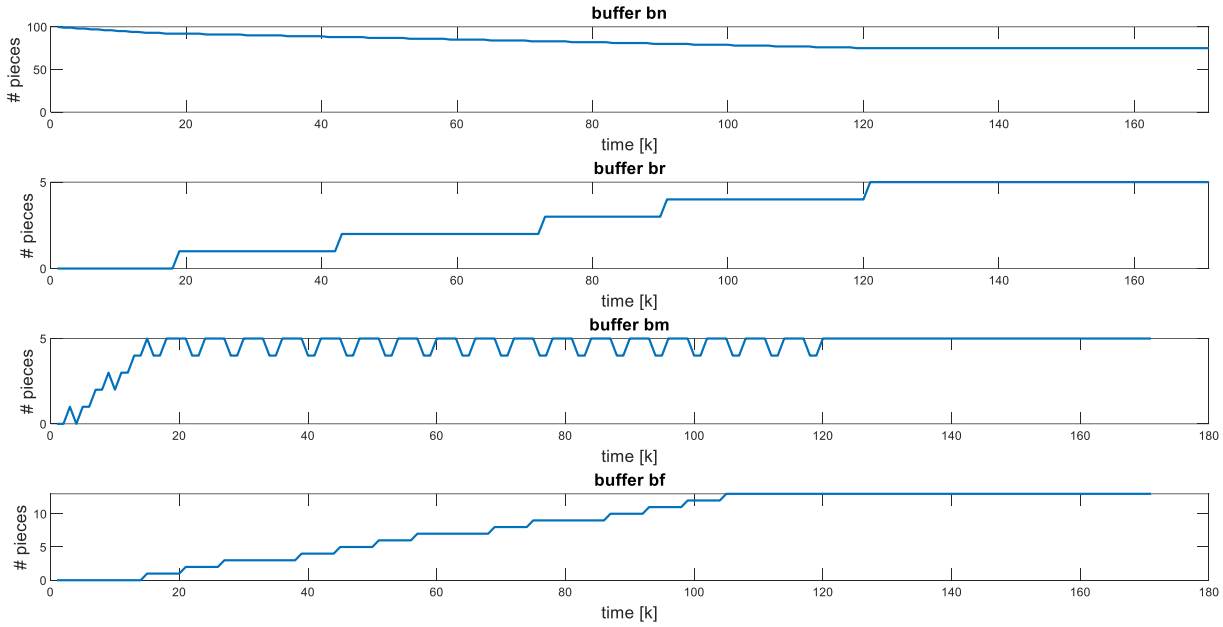

Figure 14: Test 2a – buffers behaviour

2.5.3 Test 2b: Single Node, deadlock avoided

- Number of parts: 100
- Buffer dimensions: 5 parts
- $\theta = 3$
- Time machining raw parts: 1
- Time machining fault parts: 1
- Time testing: 5
- Probability test passed: 80%
- Moving time from T to *br*: 10
- Setup time: 50

Test 2b (Figure 15) shows the solution for deadlock problem presented in Test 2a, the difference from the previous test is that value θ is decreased of one unit.

With this modification the system can finish its working cycle and the general behaviour of the machines is quite good.

It's important to know that decreasing of one unit value θ was a good solution for this specific test, in different situations with different timings it might be necessary to decrease the value, from the maximum possible, of more than one unit. On the other hand if θ takes the minimum value, i.e. θ=1, many setups occur leading to performance degradation.

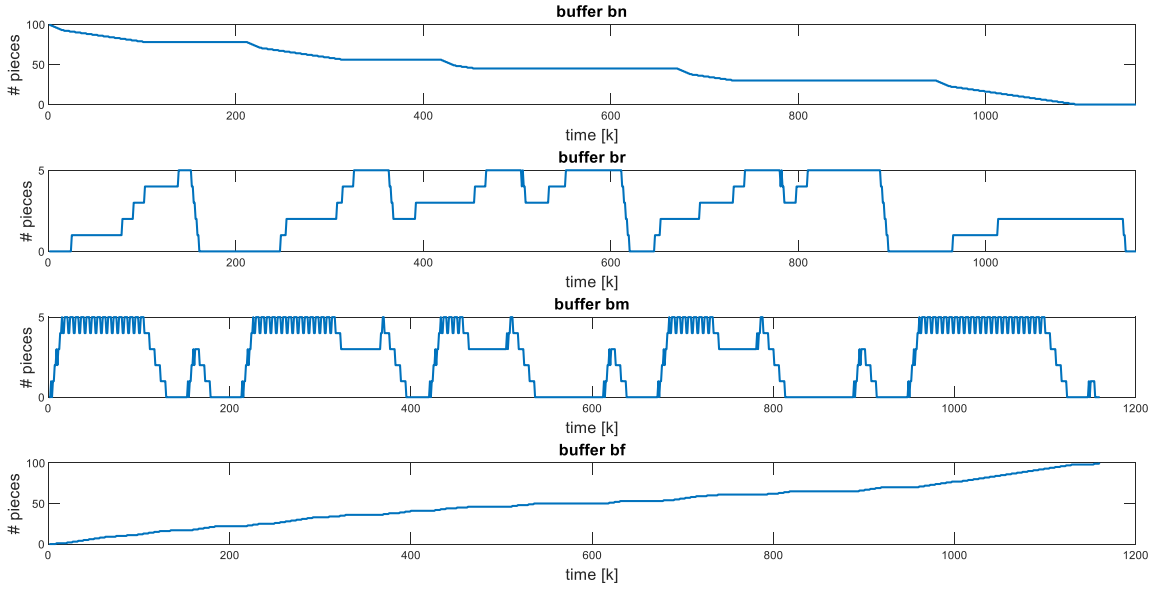

Figure 15: Test 2b – buffers behaviour

2.5.4 Test 3: Multi Node

- Number of nodes: 3
- Number of parts: 50
- Buffer dimensions: 5 parts
- θ = 3
- Time machining raw parts: [1, 1, 3]
- Time machining fault parts: [1, 1, 3]
- Time testing: [1, 5, 30]
- Probability test passed: [70%, 80%, 75%]
- Moving time from T to *br*: [10, 10, 10]
- Setup time: [50, 50, 50]

Test 3 is important to understand how the algorithm works in the multi node configuration, Figure 16 and Figure 17 report respectively the behaviour of nodes 2 and 3.

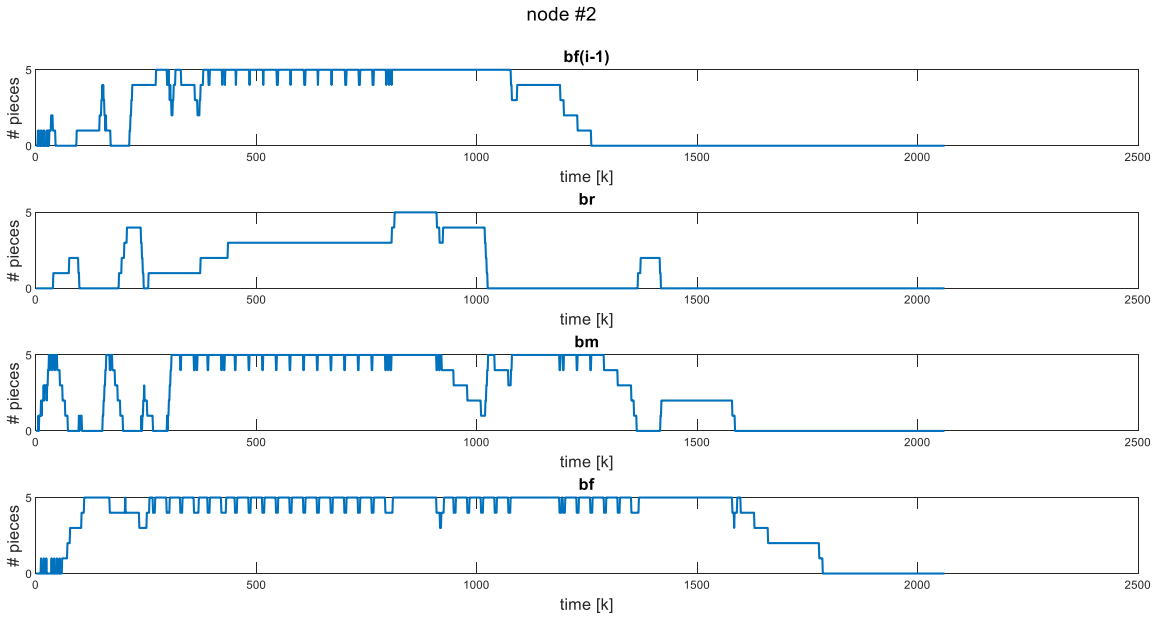

Figure 16: Test 3, Node #2

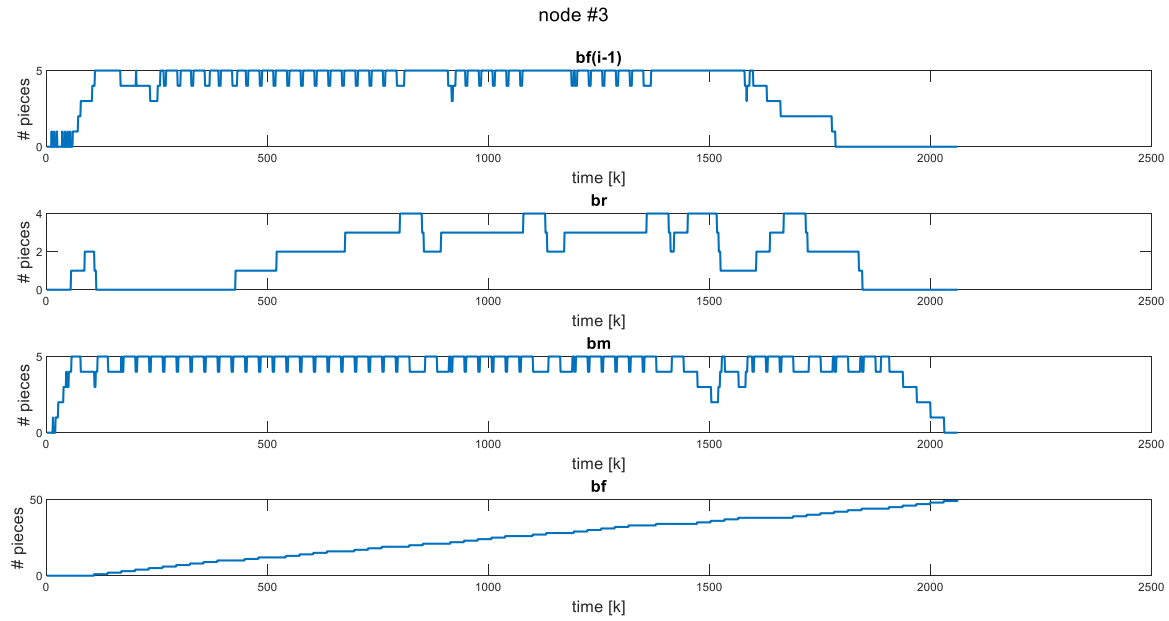

Figure 17: Test 3, Node #3

This test is important because highlights another problem of the heuristic algorithm, problem that could be solved by contribution of MPC.

Recalling the decision tree of the algorithm presented in section 2.4, an important feature of it is that is possible for the algorithm to execute setup also if *bm* has no place for new parts. In the following test is possible to see that both in node #2 and #3 buffers *br* often are not completely empty, this behaviour is motivated by the concept just mentioned.

Every situation in which the algorithm decides to stop emptying *br* is because *bm* is full, so following the decision tree the only thing that it can do is to setup to work raw parts. We will show that the MPC algorithm will be able to solve this problem by decreasing in real time the value of θ, thus forcing machine M to wait to continue emptying buffer *br*.

Chapter 3

MPC Algorithm

This Chapter describes the design and development of the MPC algorithm proposed in this thesis, including the plant model that the algorithm employs, the way it generates a sequence of optimal inputs, and what is its relationship with the heuristic approach.
The last part is concerned with the differences among the three MPC algorithms that have been implemented in this work, presented both through code explanations and test results.

3.1 Model Predictive Control

Model Predictive Control (MPC) is a family of advanced control algorithms that uses an explicit process model to predict the future response of a plant. The algorithm, at each time step, computes the best sequence of the manipulated input variables to optimize the future output [7]. Only the first input of the optimal sequence is applied to the system and a new sequence is computed at the next time step, this principle is called Receding Horizon.
This type of algorithm has had a huge impact in the industrial field since its development at the beginning of the 80s [8]. Because of the high computational cost that distinguishes this technique, MPC was originally developed for chemical plants, characterized by slow dynamics. Nowadays, thanks to the improvement of the numerical solution methodologies and, above all, thanks to the increase of the hardware power, it is widely used in all the process industry and some applications can be found in other engineering fields [9].

To understand the reasons behind its popularity it is important to list the main characteristics of the MPC algorithms. The main advantage of Model Predictive Control algorithms is the possibility to formulate the control problem as an optimization one, with the additional possibility to define constraints on state and input variables. Unlike other optimized controls, another advantage of MPC is that hardly a feasible solution for the optimizing problem, leads to an unfeasible solution in the future instants. On the other hand, this situation can occur using other optimized control because, differently from MPC, because they often do not consider the evolution of the model and constraints imposed in future instants.
These characteristics make MPC algorithms very flexible and suitable for many different applications.

3.2 General structure

When the function is called, it receives as input two variables:

1. *x*: is the state of the system and includes all the information related the actual plant situation. The array *x* is composed as follow:

$$x = [k\ \theta\ bn\ M\ tm\ w\ s\ lw\ es\ bm\_MPC\ T\ tt\ t\ br\_MPC\ bf\_MPC\ mp]$$

Where:

- *k:* is the time instant in which the algorithm has been called
- θ: is the last value saved by the plant
- *Bn:* is the number of parts in the buffer
- *Bm_MPC, br_MPC, bf_MPC:* are vectors containing in series the arrays related every buffer of every node.
- *M, tm, w (working), lw (last_work):* indicate respectively if M is processing a part, the counter if M is machining, the type of part in process (raw or rework) and the type of the last part processed (raw or rework).
- *T, tt, t:* indicate respectively if T is testing a part, the counter if T is testing and the last one indicates what type of part is testing machine T.
- *S (setup), es (end_setup):* indicate respectively if a machine is executing the setup and when will finish.

- *mp (moving_parts)*: indicates the number of parts that are moving from T to *br* and when they will arrive in the buffer.

M, tm, w, lw, T, tt, t, S and *es* are arrays with dimension corresponding to the number of nodes.

2. *sim_data:* this variable describes the features of the simulation in progress and has the following structure:
   Most important information included in this variable is the prediction horizon. Other data acquired by the algorithm through this variable are the timings to work and to test a part, time to setup a machine, number of nodes, dimension of the buffers, probabilities to pass a test.

After having acquired these two long arrays the algorithm extracts every value and rebuilds the entire plant in its workspace being ready to start the prediction phase.
The system rebuilt inside of the algorithm is quite different from the real one. Differences can be found from the modelling point of view because the time for testing is considered unitary; this modification helps the MPC algorithm to predict with more advance parts moving to buffer *br*.
On the other hand, a big difference between the real plant and the MPC's one is the decision tree of the heuristic algorithm, obviously included in the fictitious plant. In section 2.5.4, testing the heuristic algorithm, was highlighted a problem that was, in some situations, the inability, for the plant without MPC, to completely empty buffer *br*.
MPC algorithm, for is nature, should be capable to avoid this problem but, due to unitary testing times, inserted for the reasons mentioned above, it was necessary to modify the decision tree inside of the MPC.
To understand better the solution applied, the decision tree of the heuristic algorithm that is contained in MPC is reported in Figure 18.

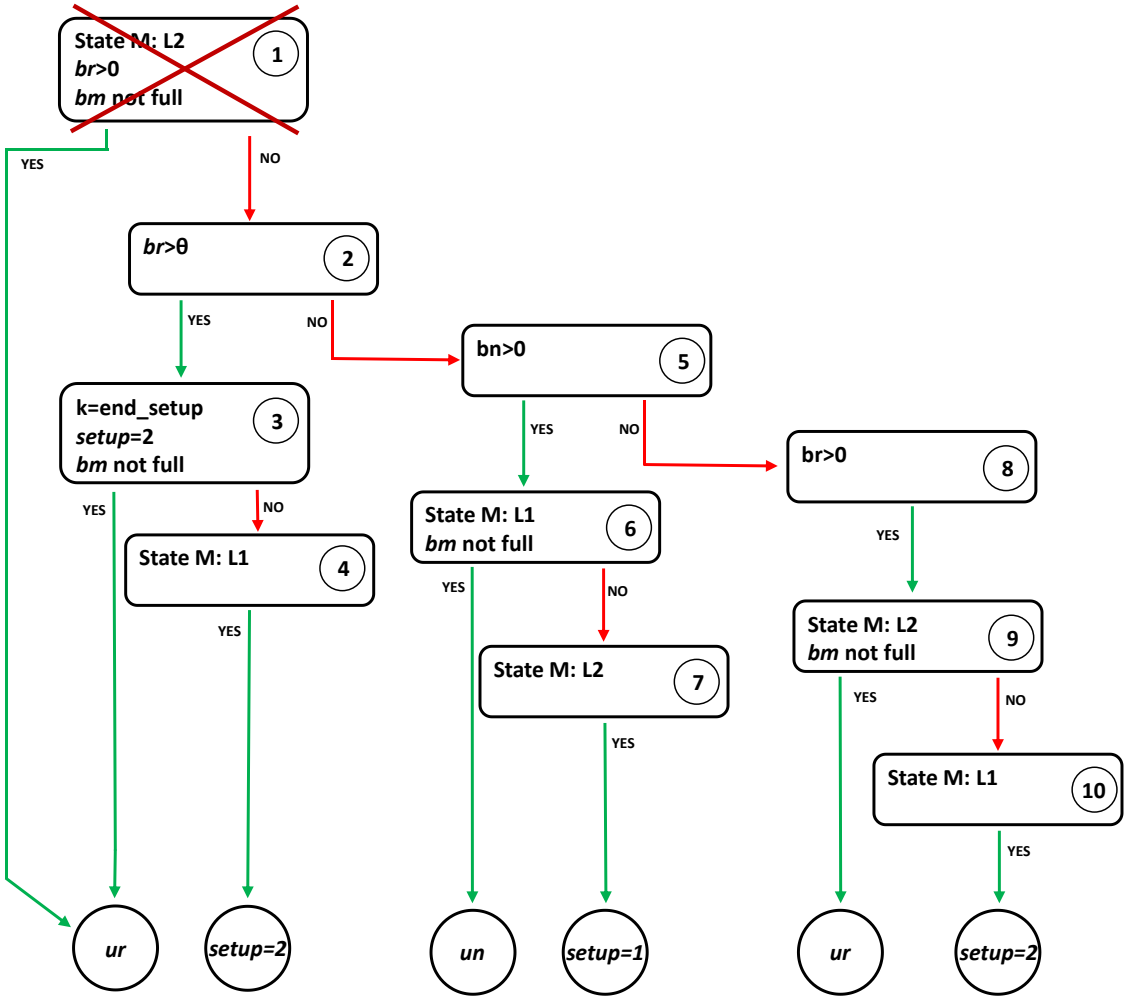

Figure 18: Decision Tree Heuristic Algorithm, contained in MPC

The solution was to remove, from the decision tree contained in MPC, the first condition, that is the one permitting to completely empty buffer *br*.

The reason behind this modification is explained by the example presented in Table 1 that shows the situation at a general k-time instant.

Table 1: Plant situation at instant k

Data	Value
Buffers capacity	5
Time for machining	1
Time for testing	10
State M	L2
State T	T1, time steps to finish 8
Parts in *br*	4
Parts in *bm*	5
Actual θ	4
Time to setup	100

As explained in section 2.5.4 in this kind of situation, being *bm* full, the heuristic algorithm would go through conditions 1,2,3,5,7 of the decision tree and would begin the setup to work raw parts. The MPC algorithm should force the plant to wait until the testing phase is finished, because in that moment machine T will test another part and will be freed a place in *bm* permitting machine M to process one part from *br*.
Table 2 and Table 3 represent respectively the real plant situation and the prediction executed by the MPC algorithm after two k-steps.

Table 2: Plant situation at instant k+2

Data	Value
Buffers capacity	5
Time for machining	1
Time for testing	10
State M	L2
State T	T1, time steps to finish 6
Parts in *br*	4
Parts in *bm*	5
Actual θ	3
Time to setup	100

Table 3: Plant prediction at instant k+2

Data	Value
Buffers capacity	5
Time for machining	1
Time for testing	1
State M	W2
State T	T1, time steps to finish 1
Parts in *br*	3
Parts in *bm*	4
Actual θ	3
Time to setup	100

From Table 3 is possible to understand the reason why MPC algorithm return θ = 3 as best solution. Instead, being the real situation represented by Table 2, thanks to these

modifications, the heuristic algorithm will go through conditions 1-2-3-4 and, because condition 4 will be false, the plant will wait without doing nothing as desired.

3.3 Deadlock Avoidance

One of the reasons why MPC algorithm is necessary is for its ability to work at the maximum production speed avoiding deadlocks. In section 2.5.2, testing the heuristic algorithm, was highlighted the problem of choosing, at the beginning of a simulation, the right θ that could avoid deadlocks.

In the plant in analysis, deadlock is defined as the situation in which both *br* and *bm* are full, and *bm* contains parts worked only once. In this situation both machines M and T can't work because after them they find full buffers, for this reason, from this point, all the process is blocked.

When MPC algorithm is called, it executes the prediction along the horizon and if it reaches a deadlock situation it begins the deadlock avoidance procedure.

This procedure cancels temporarily, so until the problem is solved, the optimization problem returning directly the smallest value of θ; in this way machine M will stop immediately production and will execute the setup as soon as possible. Obviously the prediction horizon should be long enough to have the time to effectively avoid the deadlock.

3.4 MPC_V1

The first version of MPC algorithm is the one that has the smaller computational cost and, in the theory, the minor predictive capacity; will be demonstrated later that the second version returns worst results.

When MPC_V1 is called, after having rebuilt the entire plant, executes three different simulations with the length of the prediction horizon. The difference between the three simulation is that the first one consider as θ the value received from the plant minus one, the second one uses the value of θ same as the one received from the plant and the third simulation increments the value of "old" θ by one unit.

The minimum value for θ is 0 while the maximum value is the dimension of the buffers minus one, it would not make sense return a value of θ equal to the dimension of the buffers because it would be impossible to find in *br* more than θ parts.

It's important to underline that during the simulation θ is maintained constant but, thanks to the receding horizon approach, the value that the algorithm will consider as the best will be applied on the plant only for one time instant, the next time the algorithm will be recalled and will return a new value for θ.
At the end of every simulation the MPC algorithm evaluates function J that permits to choose the best θ between the three possible, the structure of the function is the following:

$$J = bn + \sum_{i=1}^{n\_nodes} (Q_1 * M_i + Q_2 * bm_i + Q_3 * br_i + Q_4 * T_i + Q_5 * bf_i) * i - Q_6 * setup_i$$

With

$$Q_6 > Q_5 > Q_4 \geq Q_1 > Q_3 > Q_2$$

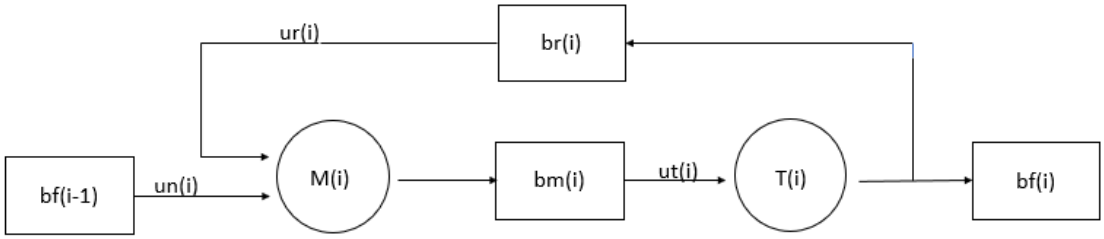

Figure 19: Structure of a Generic i-th Node

Remembering how is built a general node reported in Figure 19, is possible to understand the reason of the structure of function J. Thanks to the inequality between the Q coefficients, the more a part proceeds along the node the more it makes the function acquire value.
The subtraction at the end of the equation it's necessary because, after some tests, setups have been individuated as the most important waste of time during production.
In the next subsection will be presented a test that permits to verify the correct operation of the MPC_V1 algorithm.

3.4.1 Test: MPC_V1

- Prediction horizon: 4
- Number of nodes: 5
- Number of parts: 50
- Buffer dimensions: 5 parts
- Time machining raw parts: [1, 1, 5, 1, 10]
- Time machining fault parts: [1, 1, 5, 1, 10]
- Time testing: [1, 1, 5, 1, 30]
- Probability test passed: [70%, 80%, 75%, 70%, 75%]
- Moving time from T to *br*: [3, 3, 3, 3, 3]
- Setup time: [100, 100, 100, 100, 100]

This test has the goal to verify the correct operation of MPC_V1 algorithm.

The prediction horizon has been chosen equal to four because this value was the right balance between useful prediction and sustainable computing time.

For this test was built a plant with five nodes, furthermore the timings for machining and testing can cause bottlenecks.

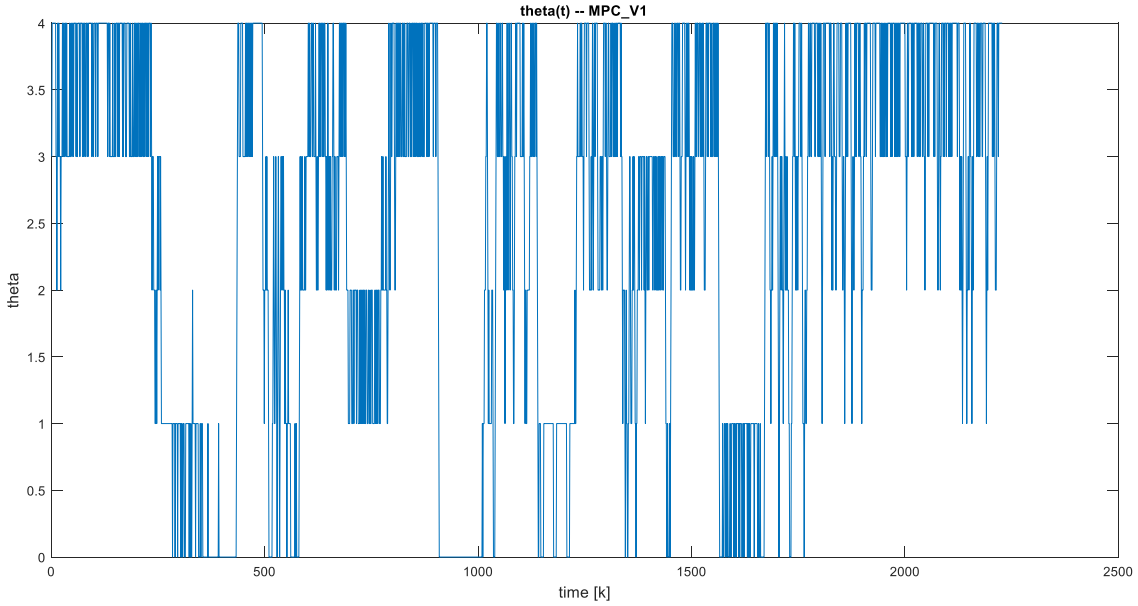

Figure 20: Test MPC_V1, Trend of θ

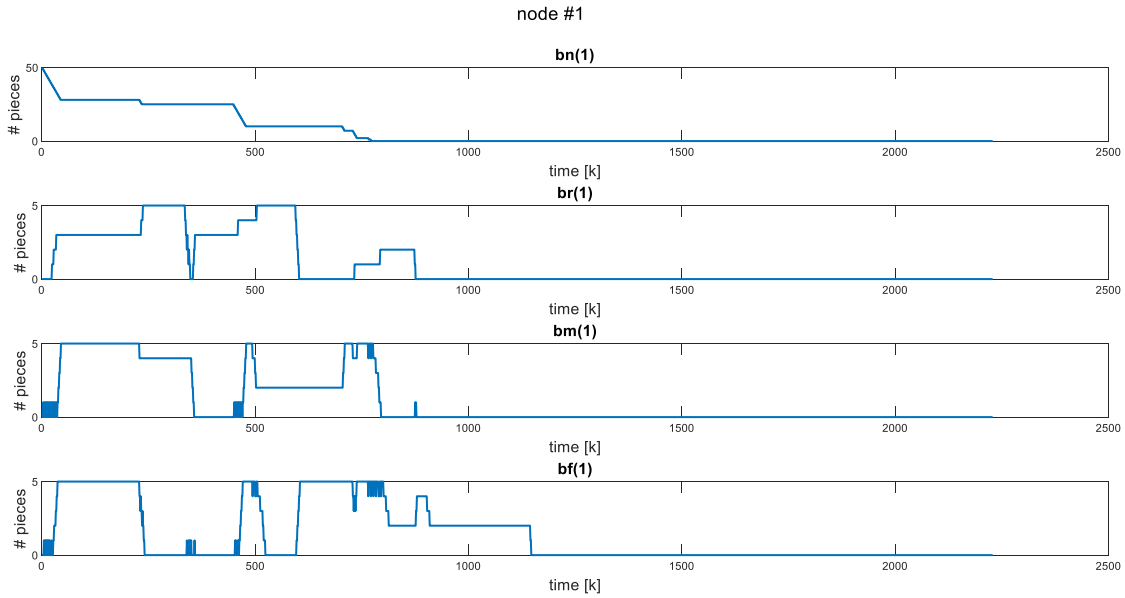

Figure 21: Test MPC_V1, Node #1

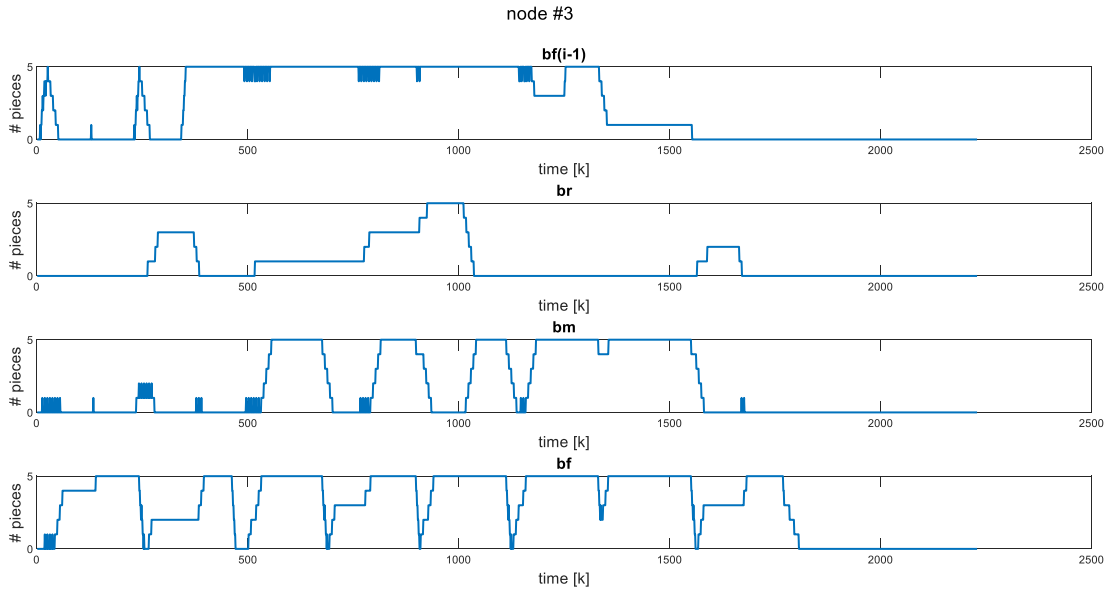

Figure 22: Test MPC_V1, Node #3

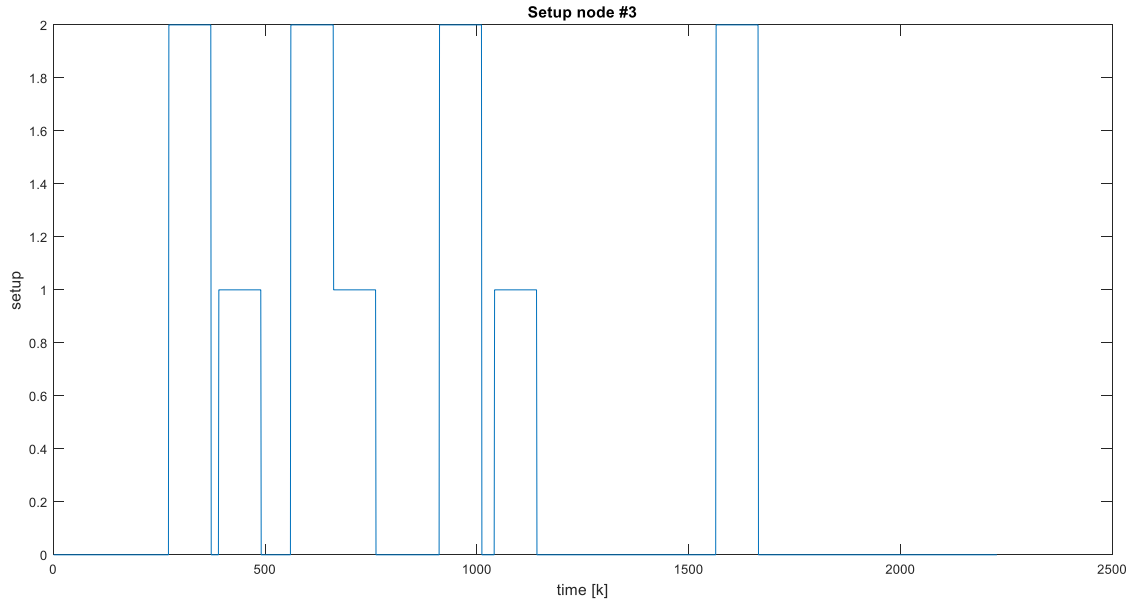

Figure 23: Test MPC_V1, Setup Node #3

Figure 20 represents the trend of θ controlled by MPC_V1, the possible range of values, as written in the description of the algorithm is [0; 4].

From this test is important to highlight the capacity of the algorithm to carry out a process like the one in analysis but is also possible to find some defects, that arise from the specific features of MPC_V1.

For example, Node #1 is responsible of the decrease of θ soon after time 500 because, as shows Figure 21, buffer *br* is full; this decrease, since θ is equal for every node, leads to the setup procedure of Node #3 reported in Figure 23.

Since the setup for Node #3 is not necessary (*br* contains only one part, Figure 22), when the procedure finishes, machine M_3 repeats the setup (setup=1) to return to work raw parts.

Executing setups at the same time between various nodes is highlighted in this test as a defect, in other situations, as will be shown in the conclusive chapter, this procedure could be useful. In some conditions, interrupting production to execute setup in all the nodes at the same time could be a solution to save time.

3.5 MPC_V2

The second version of the MPC algorithm also considers the system in its entirety, it will return for this reason a unique value θ, same for all the nodes.

The difference with MPC_V1 is that in this case the sequence of θ along the horizon is variable; as before, the receding horizon approach permits to apply only the first value of θ from the best sequence returned by the algorithm.

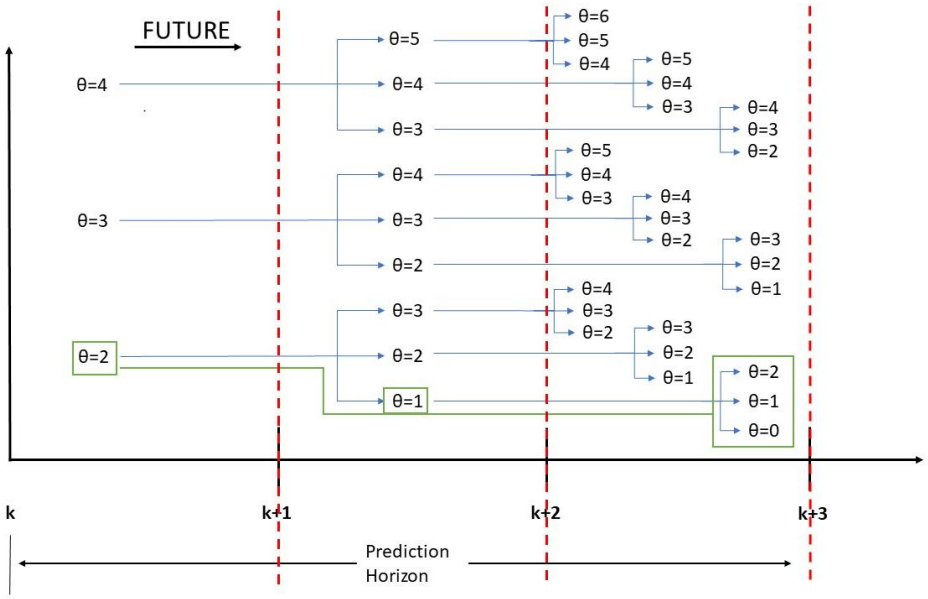

Figure 24: Searching best θ, MPC_V2

Figure 24 presents an example of how MPC_V2 generates possible sequences of θ, specifically here the prediction horizon (PH) is equal to 3, from the technical point of view the search procedure of optimal θ works as follows:

1. Simulation of the plant of one time instant with θ=2.
2. Being the prediction horizon higher than one (PH=3), recall MPC_V2 with the new state generated from simulation and prediction horizon equal to two (subtracted one from the previous value).
3. Simulation of the plant of one time instant with θ=1.
4. Being the prediction horizon higher than one (PH=2), recall MPC_V2 with the new state generated from simulation and prediction horizon equal to one (subtracted one from the previous value).
5. Simulation of the plant of one time instant with θ=0, θ=1 and θ=2.

6. Being PH=1, calculate three J functions (one for every θ simulated) and return the higher value to MPC_V2 at point 4.

This list represents the path highlighted in green in Figure 24, in this example will be evaluated 27 difference sequences and MPC_V2 will return the value of θ that permits to maximize J after three k-steps.
Function J is structured equal to the one used in MPC_V1, the equation is reported for completeness:

$$J = bn + \sum_{i=1}^{n\_nodes} (Q_1 * M_i + Q_2 * bm_i + Q_3 * br_i + Q_4 * T_i + Q_5 * bf_i) * i - Q_6 * setup_i$$

With

$$Q_6 > Q_5 > Q_4 \geq Q_1 > Q_3 > Q_2$$

3.5.1 Test: MPC_V2

- Prediction horizon: 4
- Number of nodes: 5
- Number of parts: 50
- Buffer dimensions: 5 parts
- Time machining raw parts: [1, 1, 5, 1, 10]
- Time machining fault parts: [1, 1, 5, 1, 10]
- Time testing: [1, 1, 15, 1, 30]
- Probability test passed: [70%, 80%, 75%, 70%, 75%]
- Moving time from T to *br*: [3, 3, 3, 3, 3]
- Setup time: [100, 100, 100, 100, 100]

As we will see more specifically in the conclusive chapter, unless this approach should be more accurate than MPC_V1, MPC_V2 returns worst results in terms of production time.
In the results from the test executed here, there are several situations in which it is evident that is not efficient control the whole plant with the same θ.

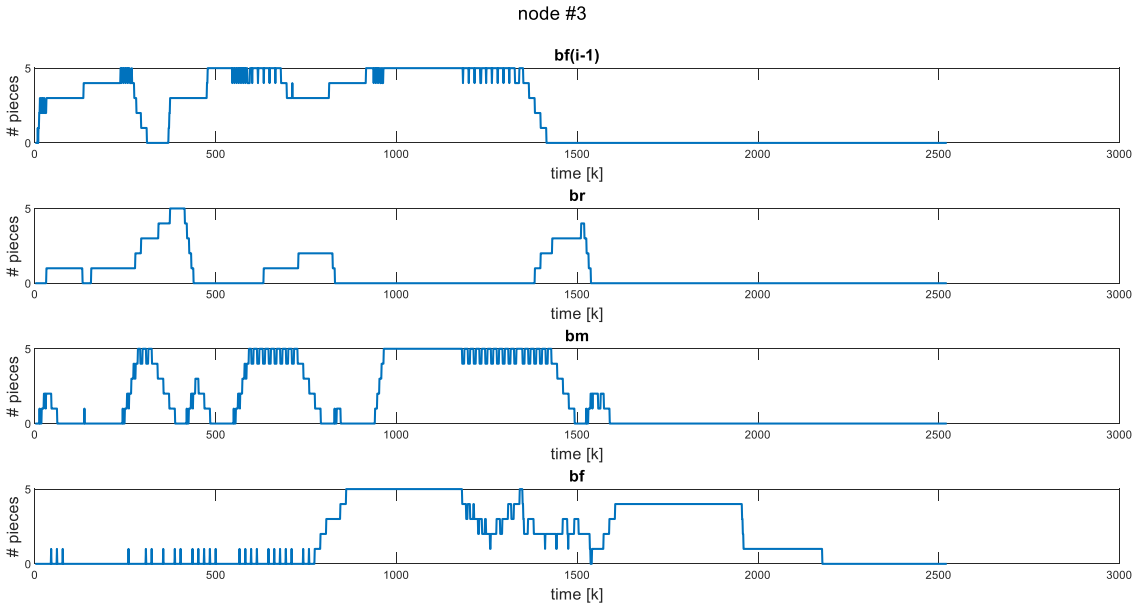

Figure 25: Test MPC_V2, Node #3

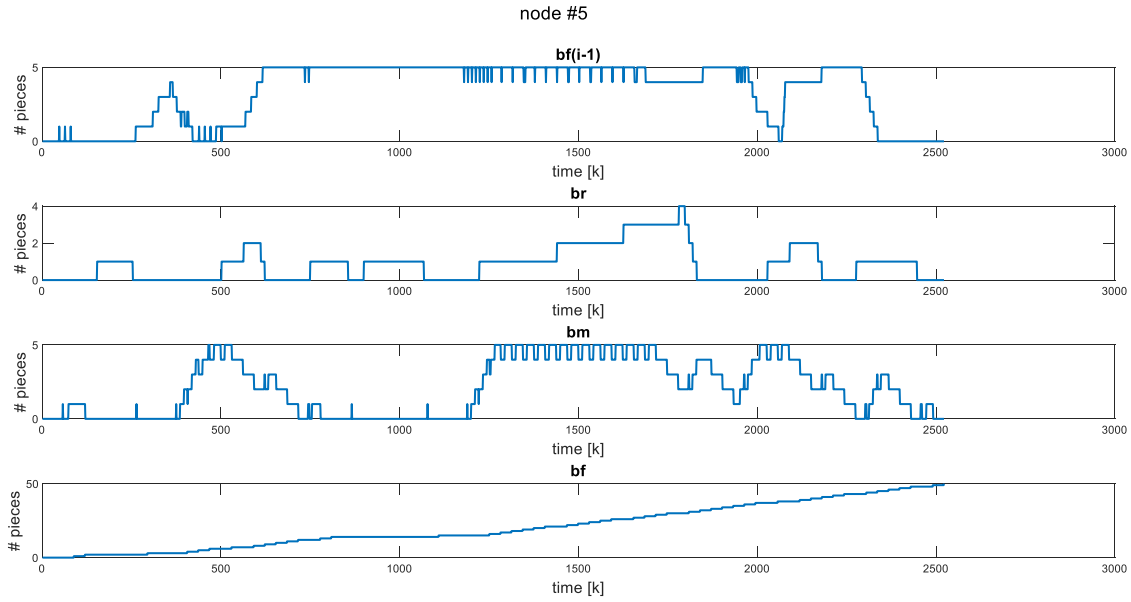

Figure 26: Test MPC_V, Node #5

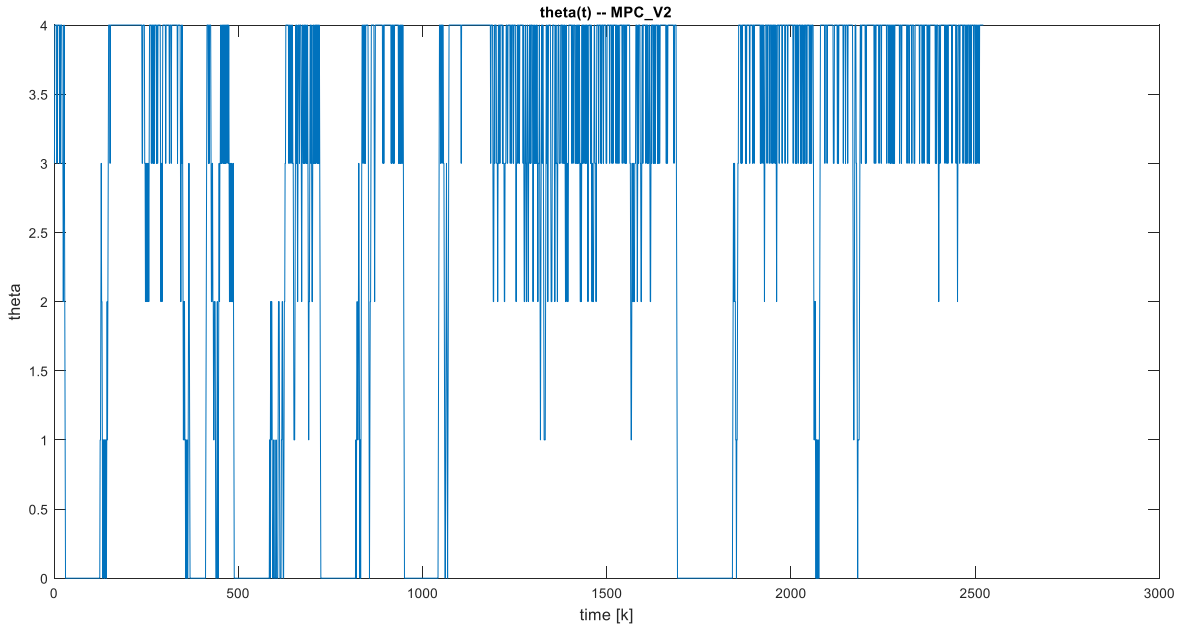

Figure 27: Test MPC_V2, Trend of θ

This test shows that the algorithm works but is evident that use the same θ for all the nodes it's not the right solution. Figure 25 for example shows the behaviour of node #3, at k equal to about 800, buffer *br*, containing only two parts, is emptied because at that time, θ reaches value 0, as shown in Figure 27.

Same situation happens in node #5 (Figure 26) at k equal to about 650, 800 and 1050 when buffer *br* contained two parts the first time and one part the other two times. As in the previous case, in this moments θ is equal to 0.

3.6 MPC_V3

The third version of the algorithm is the more complex and complete, the most important difference between this one and the previous one is that here every node is optimized on its own.

With this version of the algorithm every node calls its own MPC_V3 that has the goal to return the best sequence of θ, obviously the sequence is long as the prediction horizon.

When the algorithm is called for node *i*, it receives as input the following variables:

- *x*: contains the actual state of node *i*.
- *sim_data*: contains all data simulation for node *i*.

- *x_prev*: contains the actual state of node *i-1*.
- *sim_data_prev*: contains all data simulation for node *i-1*.
- *prediction*: is the best sequence of θ for node *i-1* that the algorithm had chosen shortly before.

Is important that the MPC algorithm called for node *i* can simulate also node *i-1* because has to know the behaviour, along the prediction horizon, of *bf(i-1)*.

When the algorithm is called before testing the three possible θ it will simulate the behaviour of the previous node, then the optimizing procedure is the same as MPC_V2, the scheme that explains the generation of possible sequences of θ is reported in Figure 28.

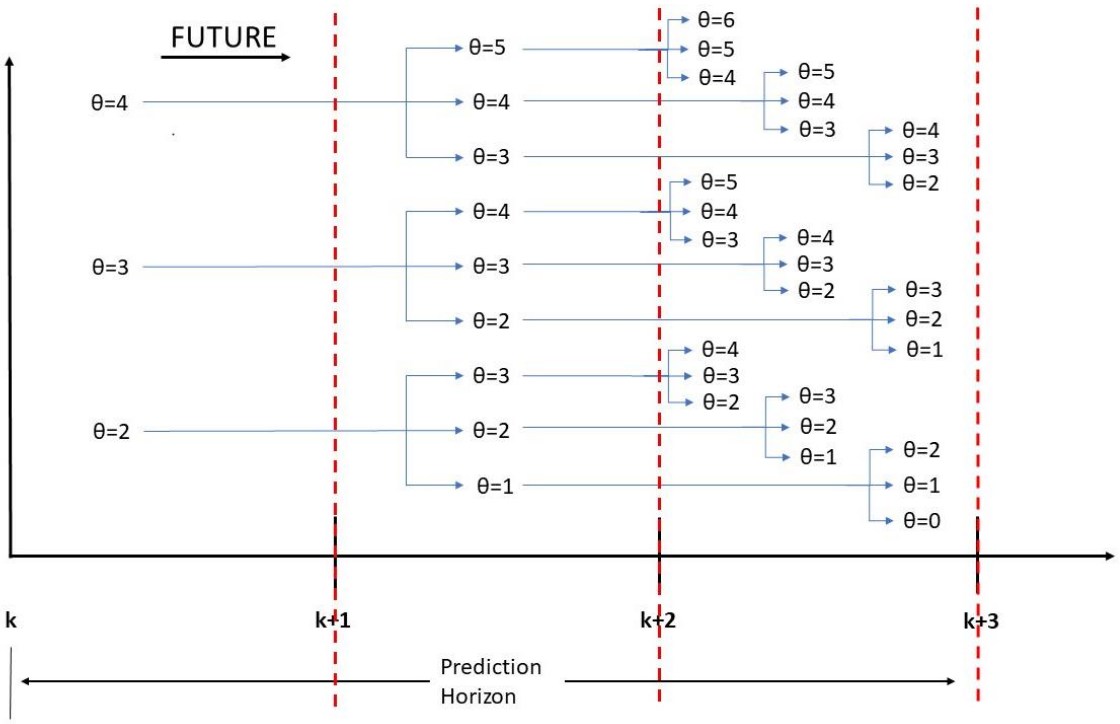

Figure 28: Searching best θ, MPC_V3

Another peculiarity of MPC_V3 is that has the possibility to return θ = -1, this negative value should permit to machine M, if *br* is empty, to wait an incoming part in state L2 without working or doing setup.

This feature is responsible for the structure of J:

$$J = bn + Q_1 * M + Q_2 * bm + Q_3 * br + Q_4 * T + Q_5 * bf) * i - Q_6 * setup * worked + 50 * worked$$

With

$$Q_6 > Q_5 > Q_4 \geq Q_1 > Q_3 > Q_2$$

Observing the form of function J is possible to find two important differences from the previous version: the absence of the summation and the presence of variable *worked*.
The summation is not present because, as written before, the algorithm is optimizing only one node.
Boolean variable *worked* assumes value 1 if, during the prediction, machine M has worked one or more parts, otherwise if machine M didn't work anything *worked* will be equal to 0.
In other words, the setup procedure will penalize the function only if is possible working parts during the prediction phase.
Variable *worked* is fundamental for the proper operation of MPC_V3 because, without its presence if function J, the algorithm should maintain θ = -1 and never return to work raw parts. Thanks to variable *worked*, if the algorithm does not predict incoming parts in *br*, setup will not penalize J.

3.6.1 Test: MPC_V3

- Prediction horizon: 4
- Number of nodes: 5
- Number of parts: 50
- Buffer dimensions: 5 parts
- Time machining raw parts: [1, 1, 5, 1, 10]
- Time machining fault parts: [1, 1, 5, 1, 10]
- Time testing: [1, 1, 15, 1, 30]
- Probability test passed: [70%, 80%, 75%, 70%, 75%]
- Moving time from T to *br*: [3, 3, 3, 3, 3]
- Setup time: [100, 100, 100, 100, 100]

This test highlights the capability of MPC_V3 to manage variable θ, different for each node. Figure 29 for example, representing behaviour of node #1, shows that when buffer br reaches the maximum value, at that time MPC algorithm decreases θ (Figure 30) and brings its value equal to -1.

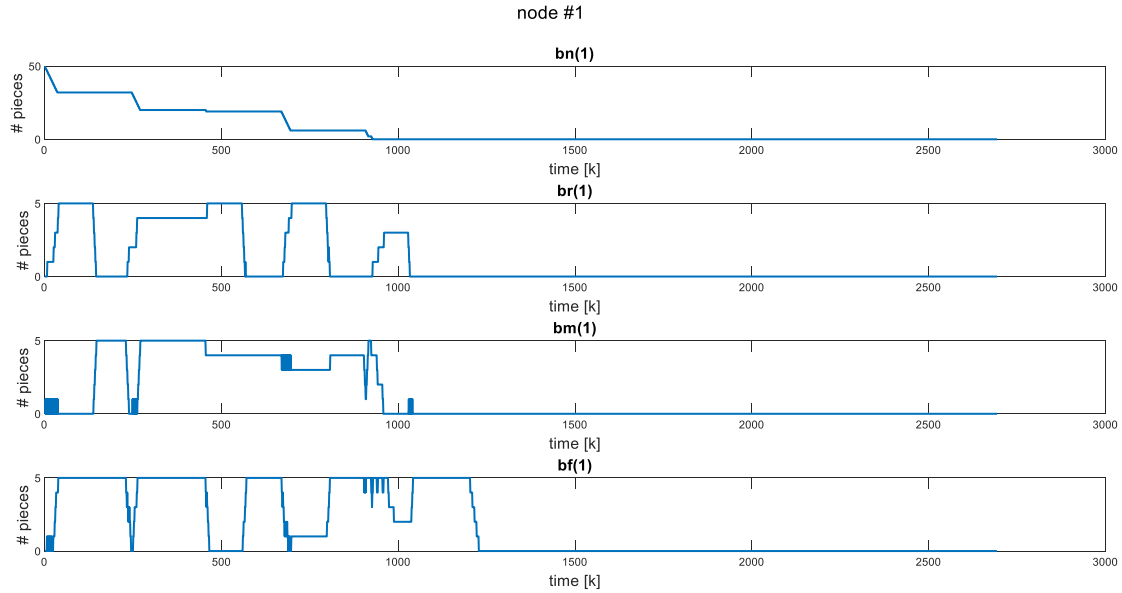

Figure 29: Test MPC_V3, Node 1

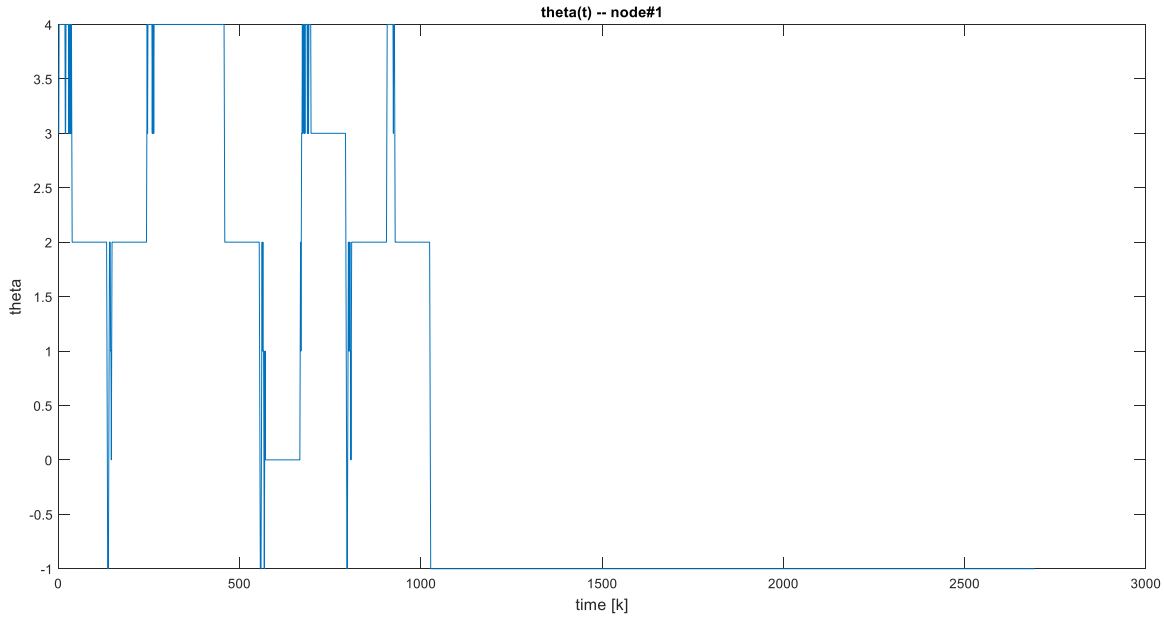

Figure 30: Test MPC_V3, Trend of θ, Node #1

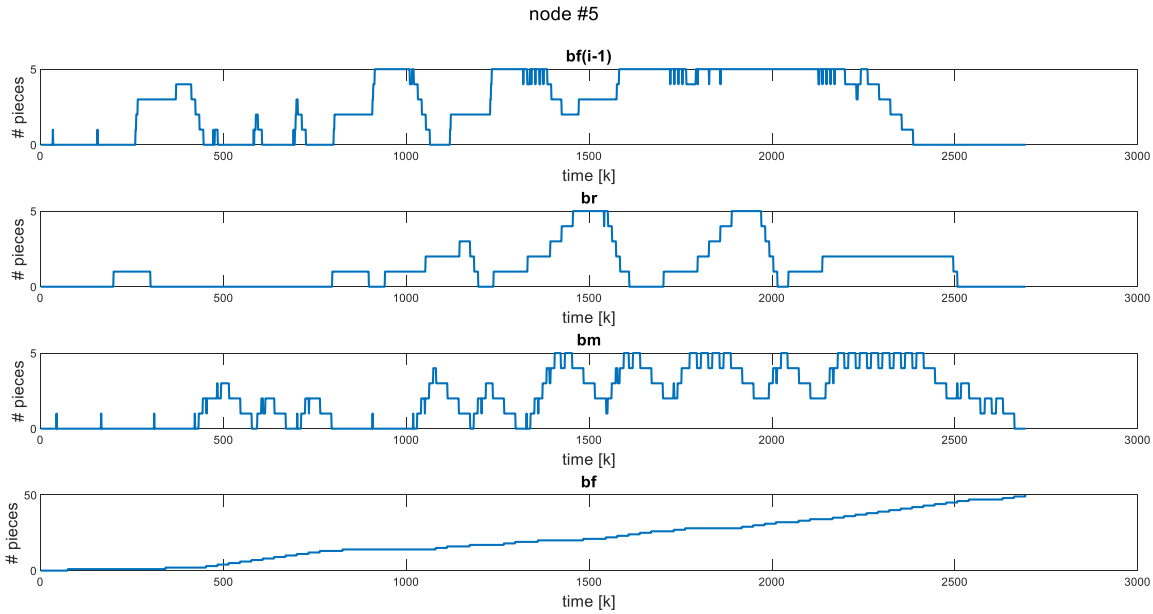

Figure 31: Test MPC_V3, Node #5

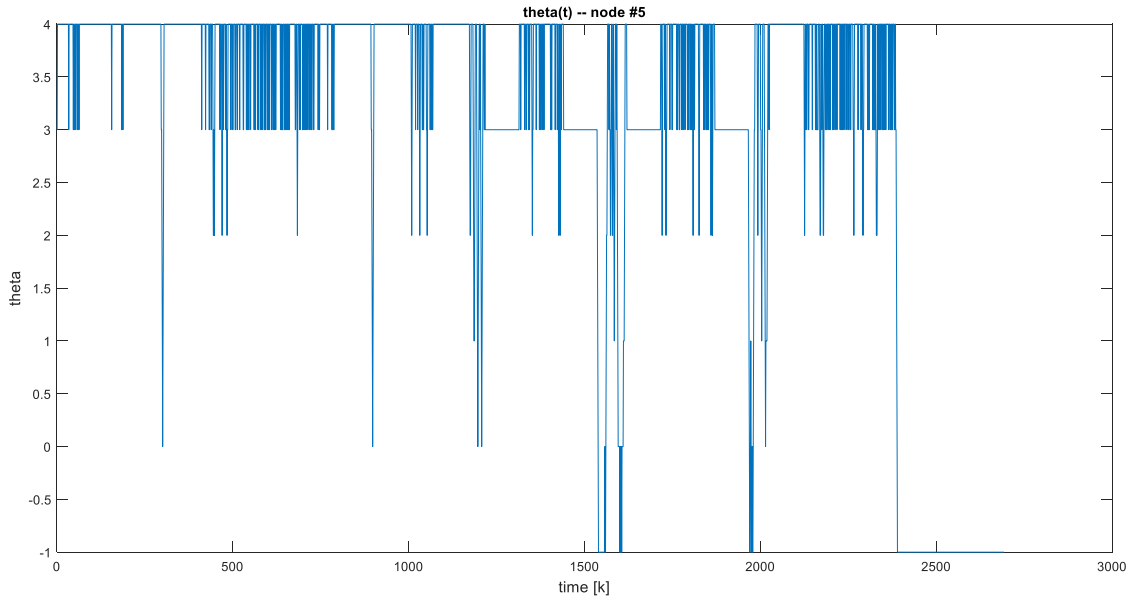

Figure 32: Test MPC_V3, Trend of θ, Node #5

Node #5 (Figure 31) it's interesting because, there, the algorithm seems not working correctly; specially in the first part of the process, for three times, buffer *br* is emptied before reaching its maximum value.

Actually MPC is not responsible of those reworking processes but is the heuristic algorithm that, being in that moments buffer *bf(4)* empty, executes the setup to rework parts from *br*.

Chapter 4

Results and conclusion

This chapter presents the results of the tests carried out to verify the contribution of MPC algorithms making a comparison with the heuristic algorithm alone.

In the first part every MPC approach will be compared with benchmark showing the capability of MPC to save time and to provide same or better performances with smaller buffers which therefore leads to economic savings.

During the evaluation of performance between the MPC approaches was individuated the problem of the influence of testing results, to make a corrects comparison was important to have the same conditions.

For this reason, the solution found was to execute different simulations using for every simulation one sequence of testing results, from machines T, and with that sequence compare the performances of the various approaches.

In the last part of this chapter will be presented some possible future works to improve the MPC algorithm and allow it to deliver even better results.

4.1 Optimization of heuristic algorithm

The first part of the evaluation of the MPC algorithms started from benchmark, was fundamental to evaluate the maximum performance that the heuristic algorithm could provide to the plant.

To this aim was considered the following simulation:

- θ = **various**
- Number of nodes: 3
- Number of parts: 50
- Buffer dimensions: **various**
- Time machining raw parts: [1, 1, 3]
- Time machining fault parts: [1, 1, 3]
- Time testing: [1, 3, 30]
- Probability test passed: [70%, 80%, 70%]
- Moving time from T to *br*: [10, 10, 10, 10]
- Setup time: [100, 100, 100]

The goal was to find the best combination between dimension of the buffers and θ that permits to provide the minimum production time.

Obviously, it was taken for granted that, starting from a buffer dimension, the maximum θ that permits to end the simulation correctly without deadlock is the one that provides better production time.

The combination of buffer dimension and θ was chosen to guarantee a sequence of 50 tests without deadlocks, for this reason some values are not considered, the production time reported in Table 4 corresponds to average times on the 50 tests executed.

The result is that the best combination is buffer dimension equal to 10 and θ equal to 9.

Table 4: Benchmark optimization

Buffers dimension	θ	Production time
5	2	2139
6	4	2120
10	9	2041
20	19	2047

4.2 MPC VS Benchmark

To compare the MPC algorithms with benchmark, the procedure has been the following:

1. Simulate the plant controlled by the heuristic algorithm in its best conditions (buffer dimensions equal to 10 and θ equal to 9).
2. Extract from the previous simulation the sequence of the testing results.
3. Simulate the plant controlled by MPC algorithm using the testing results previously mentioned.

4.2.1 MPC_V1 VS Benchmark

The three figures reported (Figure 33, Figure 34, Figure 35) show the behaviour of every node in the two different simulations.

It's interesting to see that buffer *br(1)* in MPC reworks one part more than in benchmark, but, despite this fact, Node #1 in both the approaches end to process parts at the same time.

About node #2 is important to highlight that, during the second emptying phase of *br*, that happens in both the approaches because *bf(1)* is empty, MPC has the possibility to rework one part less than heuristic because MPC is in advance against benchmark; this fact entails, for MPC, to waste more time later and finish processing in node #2 a little bit time after benchmark.

In Node #3 MPC works faster than the benchmark and can finish the whole process with an advance of about 70 time steps.

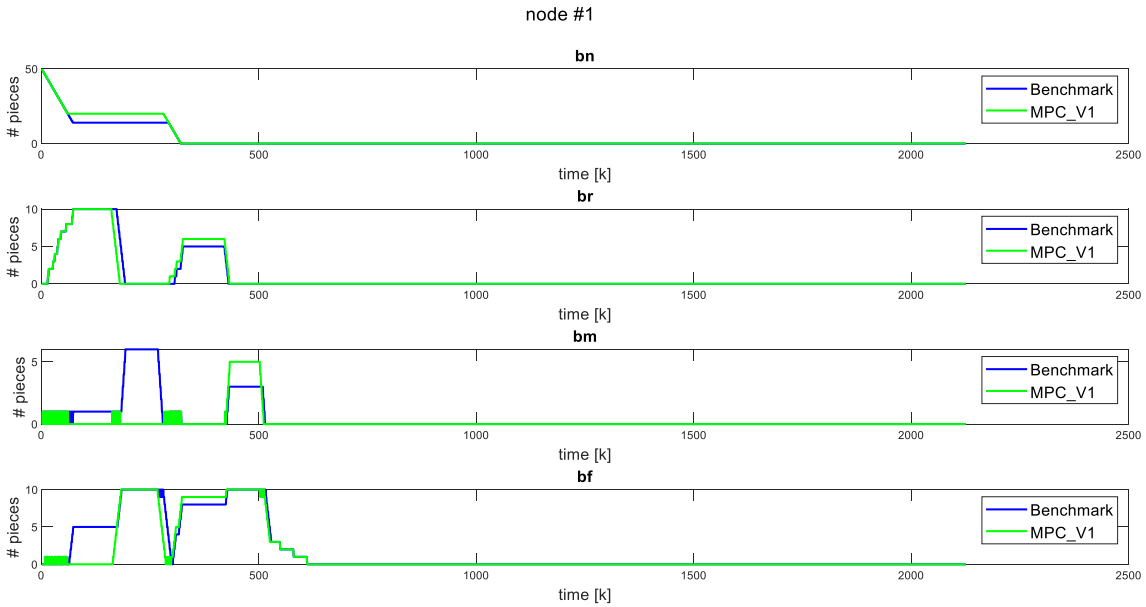

Figure 33: MPC_V1 VS Benchmark, Node #1

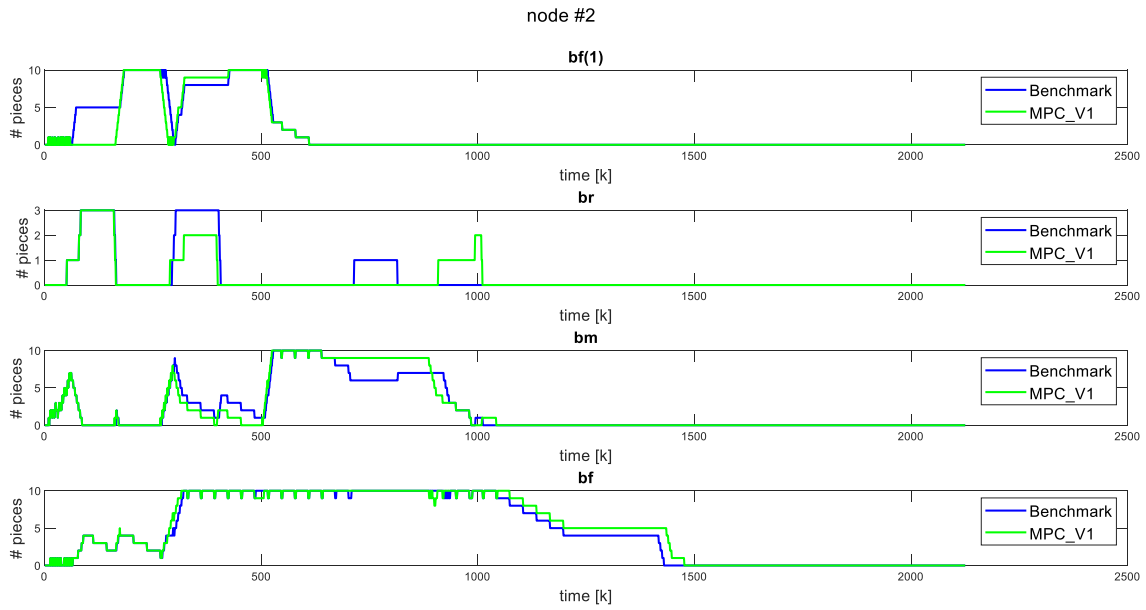

Figure 34: MPC_V1 VS Benchmark, Node #2

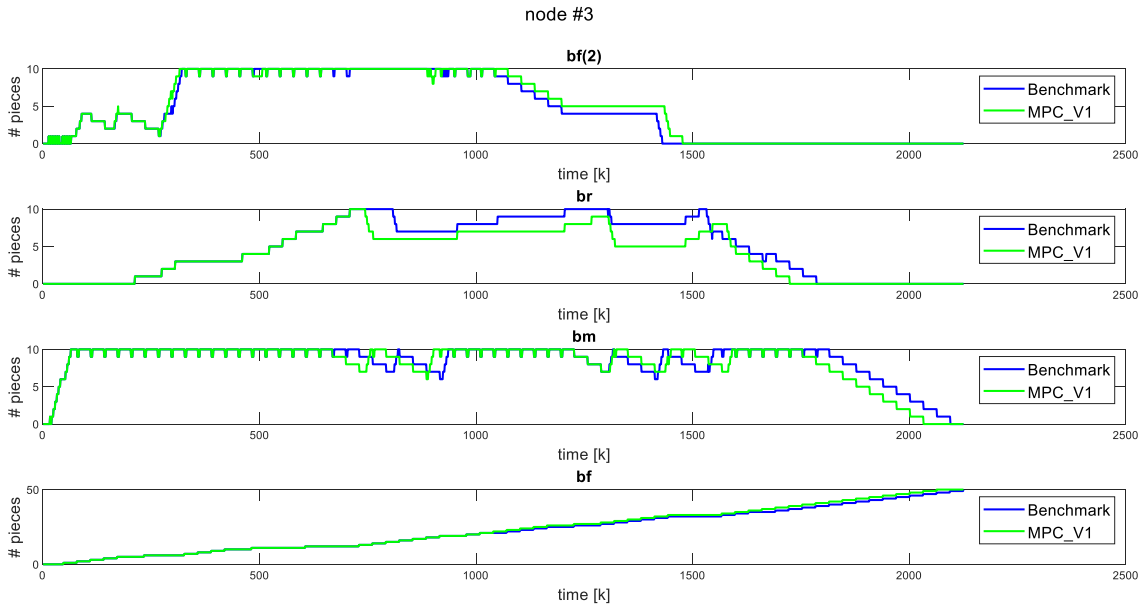

Figure 35: MPC_V1 VS Benchmark, Node #3

4.2.2 MPC_V2 VS Benchmark

The result from the comparison between MPC_V2 and benchmark is not so satisfactory and in the last part of the chapter will confirmed that MPC_V2 is not the best approach.

In Node #1 (Figure 36) the behaviour between the two different algorithms is quite the same and the node in analysis finish its process at the same time.

In the processing on Node #2 (Figure 37) MPC gains some delay against benchmark mostly because *bf (2)* is emptied slower and so also *bm*, this behaviour that seems to be a slowdown by MPC will provoke instead a gain of time in Node #3 (Figure 38).

Buffer *br(3)* is controlled by MPC_V2 in a good way such that can save one setup against benchmark; the simulation ends with an advance of MPC_V2 of about 70 *k* (as MPC_V1).

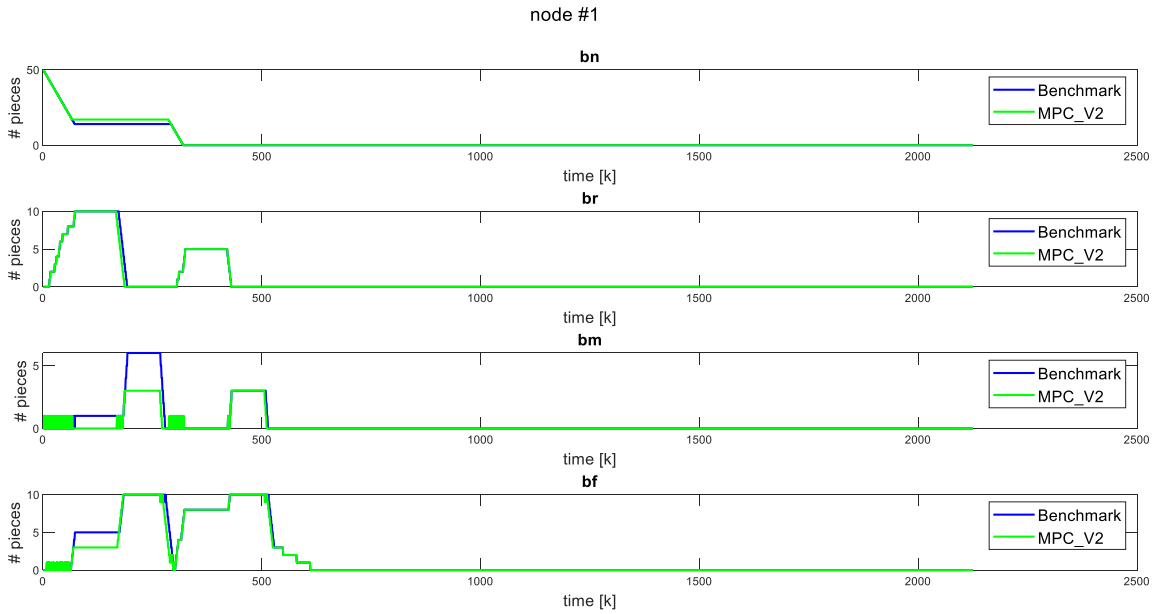

Figure 36: MPC_V2 VS Benchmark, Node#1

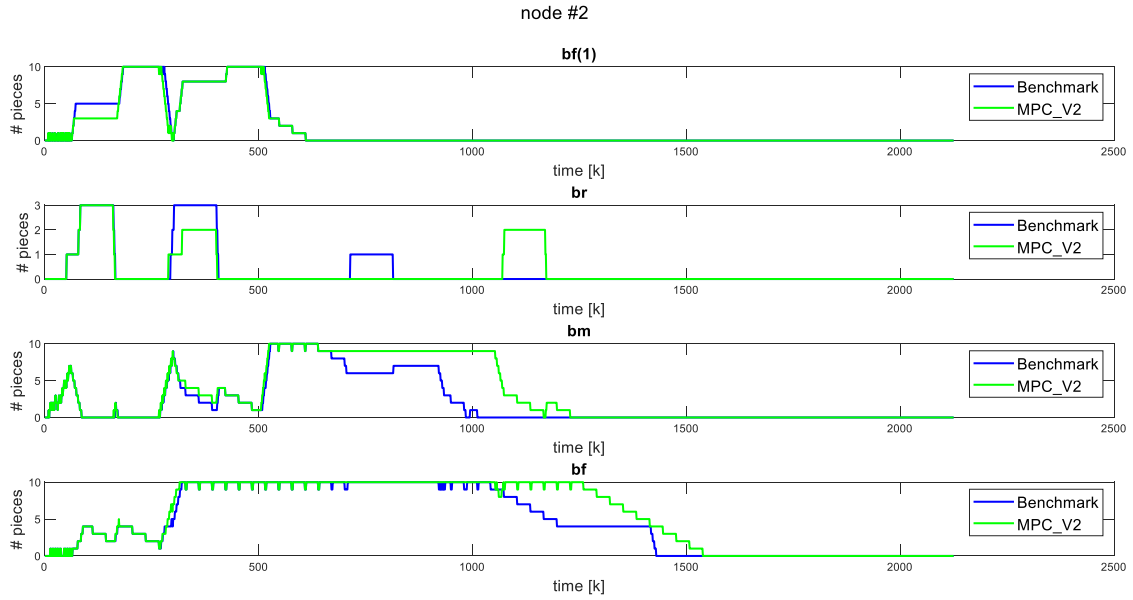

Figure 37: MPC_V2 VS Benchmark, Node#2

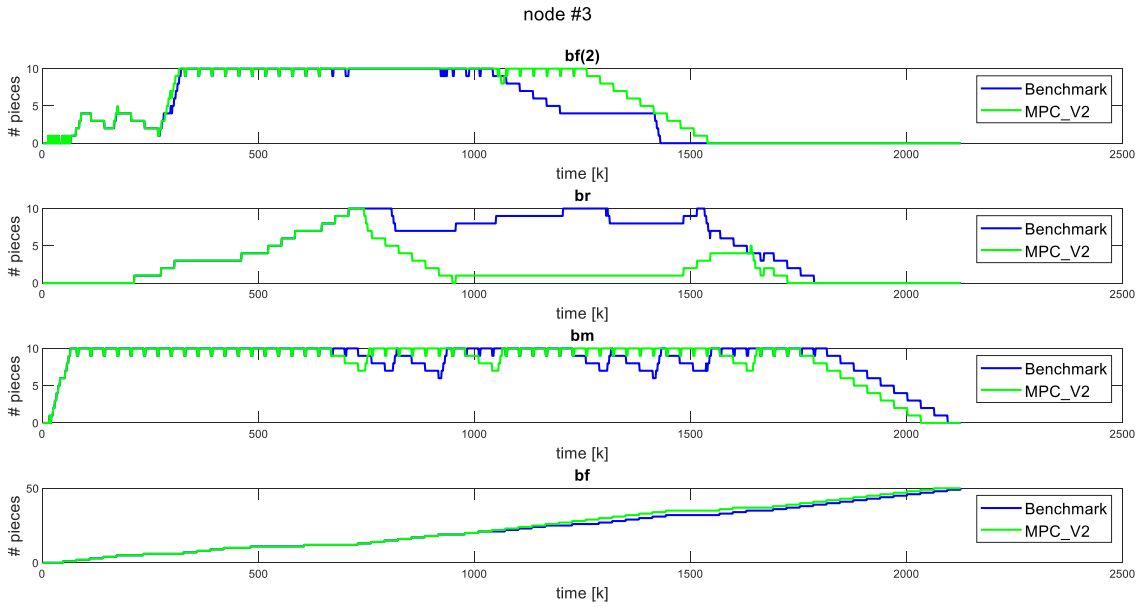

Figure 38: MPC_V2 VS Benchmark, Node#3

4.2.3 MPC_V3 VS Benchmark

Results of the test are plotted in Figure 39, Figure 40 and Figure 41.

The behaviour of MPC_V3 is quite similar to the one of MPC_V2, also in this case the MPC algorithm can optimize production and save the same time as the other approaches.

It's understandable the reason why this approach and the previous have the same behaviour, this happens because there are not situations in which θ has to control different nodes.

Even if seems that the three approaches have the same or similar behaviour will be demonstrated that in other situations the control actions of the three algorithms are different between each other.

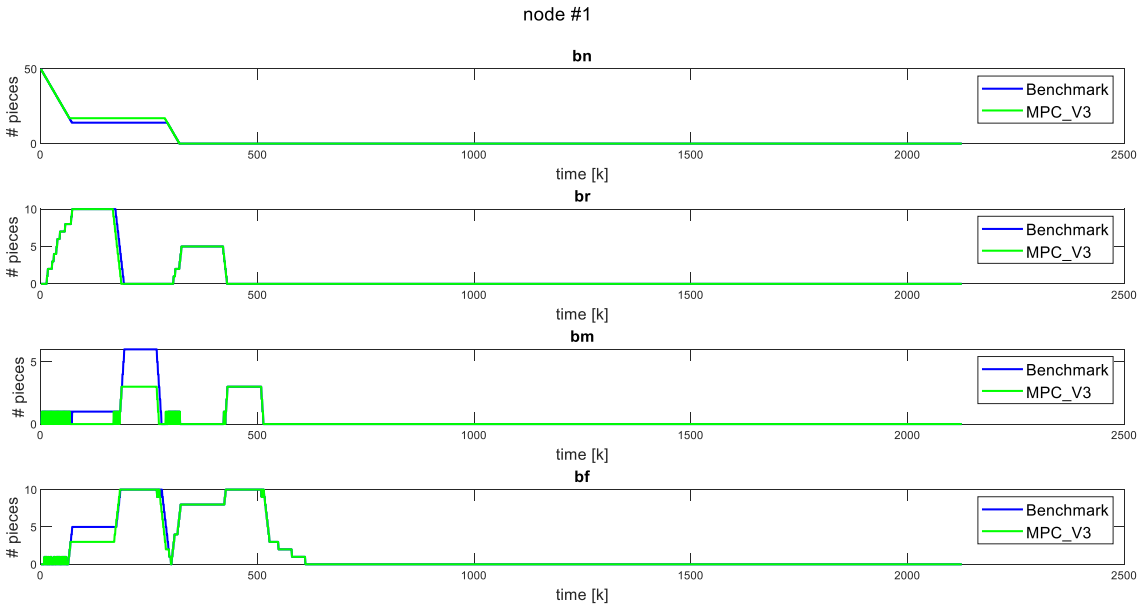

Figure 39: MPC_V3 VS Benchmark, Node #1

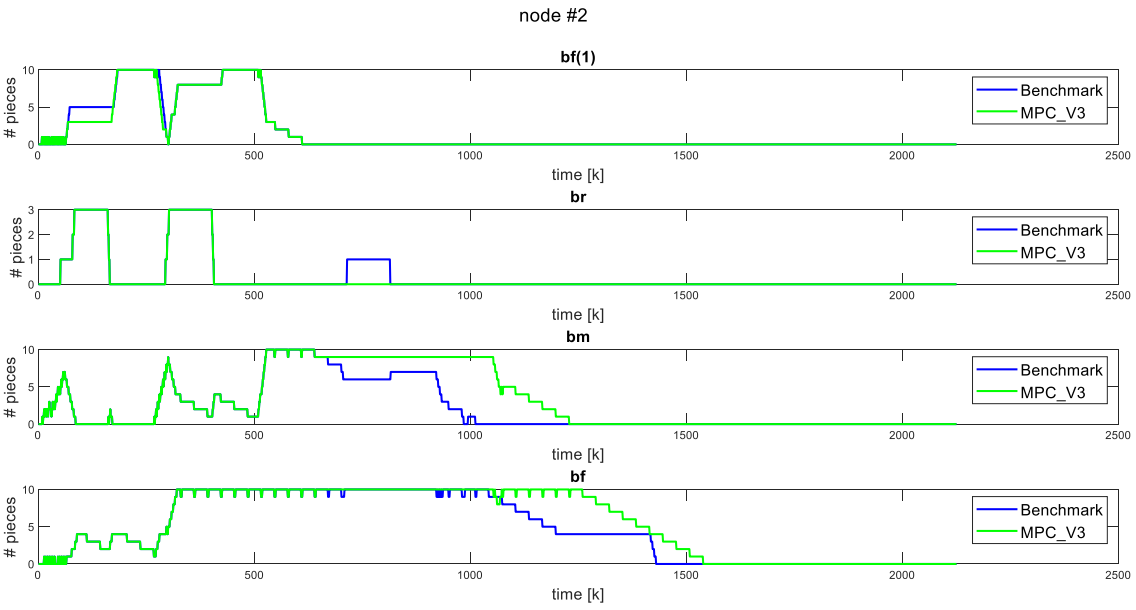

Figure 40: MPC_V3 VS Benchmark, Node #2

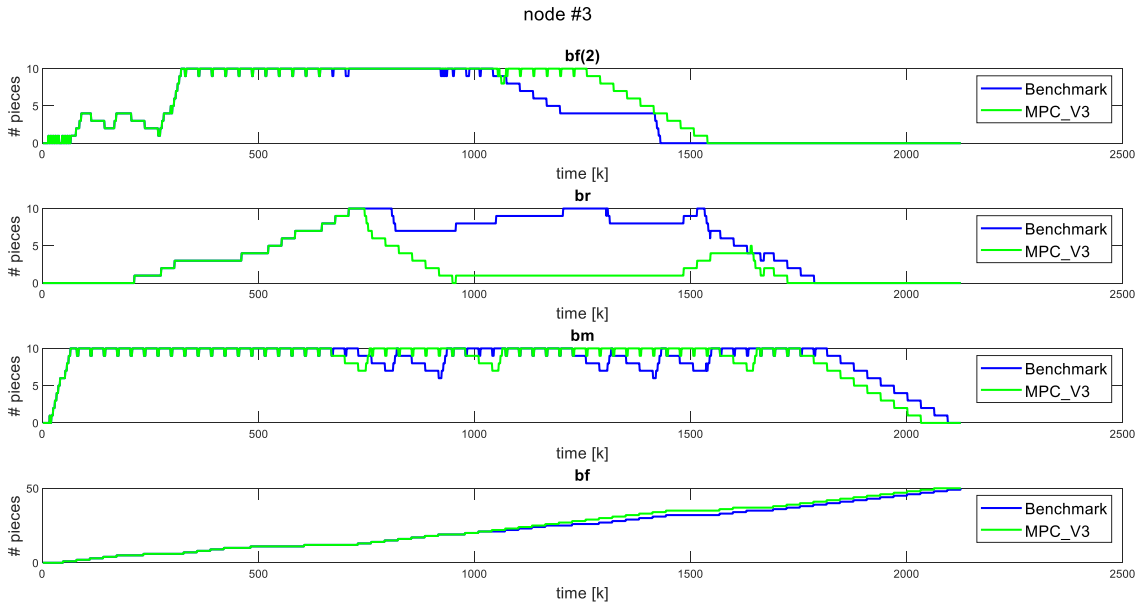

Figure 41: MPC_V3 VS Benchmark, Node #3

4.3 Evaluation MPC performance

In this section will be presented the last comparison, the target was to evaluate the influence of MPC algorithms with many tests.

To do this analysis have been evaluated the following simulation:

- Prediction horizon: 4
- $\theta = 3$
- Number of nodes: 4
- Number of parts: 50
- Buffer dimensions: 5 parts
- Time machining raw parts: [1, 1, 1, 3]
- Time machining fault parts: [1, 1, 1, 3]
- Time testing: [1, 1, 5, 30]
- Probability test passed: [70%, 80%, 75%, 70%]
- Moving time from T to *br*: [10, 10, 10, 10]
- Setup time: [100, 100, 100, 100]

This test was repeated 100 times, every time the heuristic algorithm generated a random sequence of testing results and after having simulated its process the three approaches was tested using the same sequence previously generated.

Table 5: Comparison between approaches, time to complete simulations

	BENCHMARK	**MPC_V1**	**MPC_V2**	**MPC_V3**
Total time (Average)	2300	2270	2298	2200
Improvement (%)	100%	-2%	0%	-4%
Variance	28000	28000	30000	22000

Table 5 reports general results of this test, in detail the first raw indicates the average time that every approach needed to complete a simulation. The second raw indicates in percentage the difference between benchmark and every MPC.

The last raw correspond to the variance between the one hundred tests executed on every approach.

Is evident that MPC_V1 and MPC_V3 provide an improvement, even if in limited percentages, on the other hand MPC_V2 do not returns an interesting result.

Other statistical analyses are now presented to understand better the results, starting from Table 6 than analyses more specifically the timings to conclude the simulations.

The three rows of the table indicate, on the 100 tests, how many simulations provided a better, a worse or the same time against benchmark. The first column of every approach divides the 100 tests in the three categories just explained while the second one contains the average of the increase or decrease from benchmark timings.

Table 6: Differences of time between benchmark and every approach

	MPC_V1		MPC_V2		MPC_V3	
	%	**Average difference**	**%**	**Average difference**	**%**	**Average difference**
# of tests better time/100	45	203	47	155	55	170
# of tests worse time/100	50	120	53	130	36	80
# of tests same time/100	5		0		9	

Table 7: Quartiles of the differences of time between benchmark and every approach

	Quartiles				
Approaches	**0%**	**25%**	**50%**	**75%**	**100%**
MPC_V1	-464	-99	-1	167	582
MPC_V2	-401	-132	-28	130	542
MPC_V3	-272	-49	15	143	458

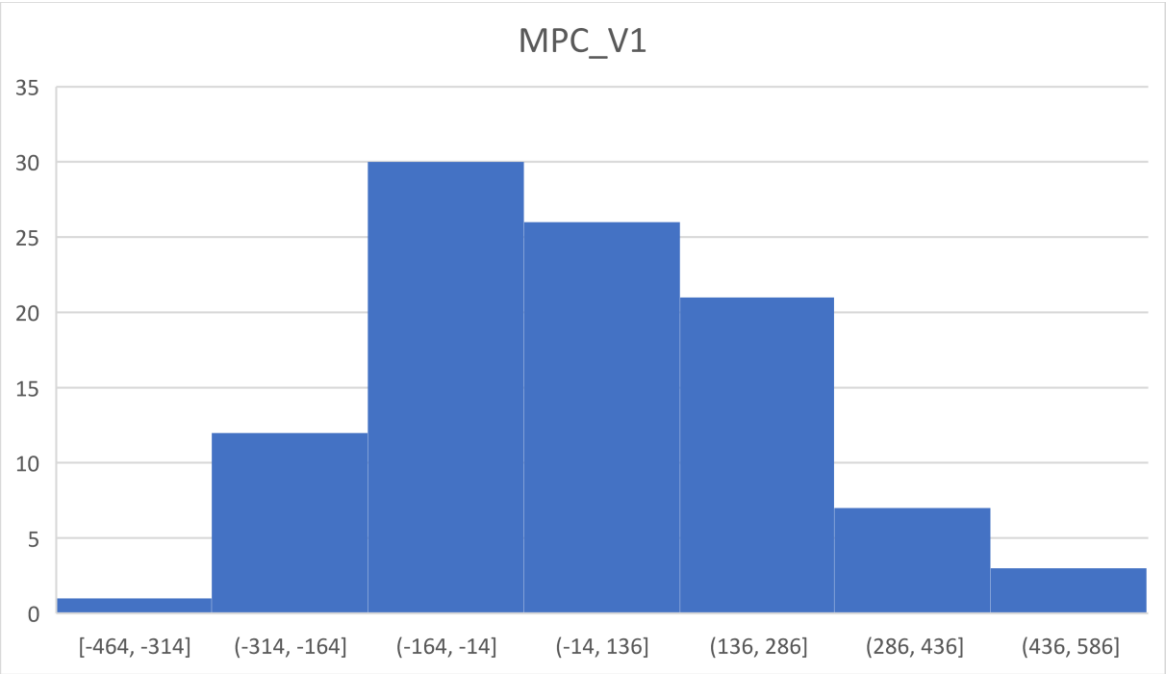

Figure 42: Distribution of the difference of time between benchmark and MPC_V1

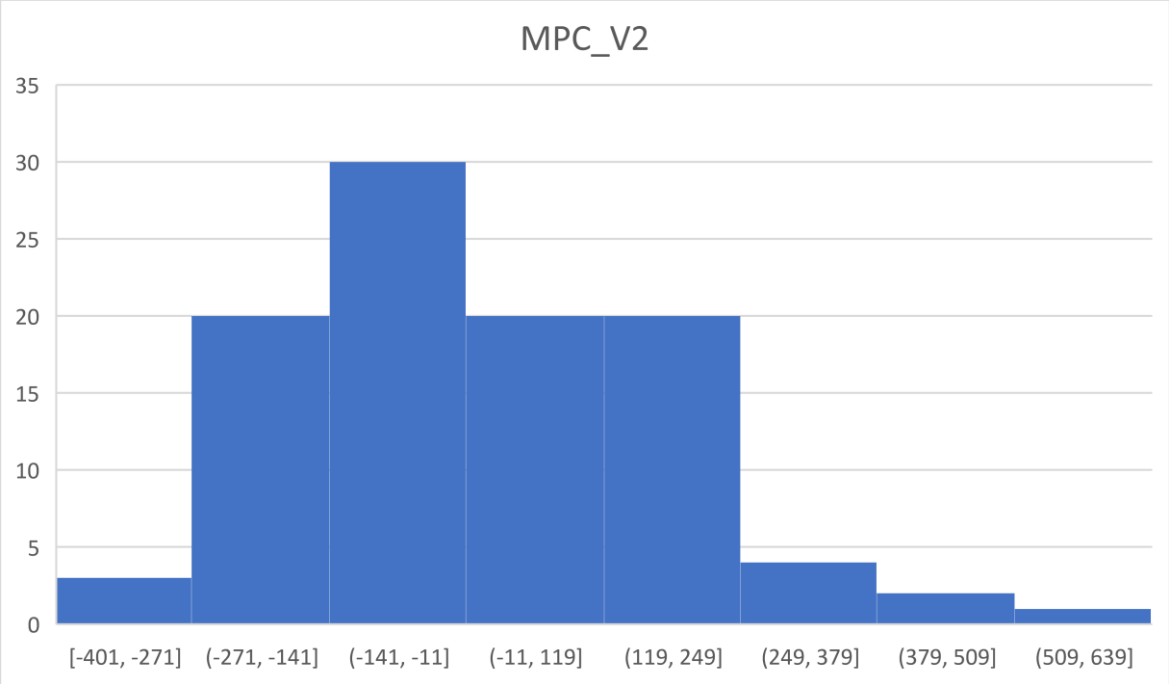

Figure 43: Distribution of the difference of time between benchmark and MPC_V2

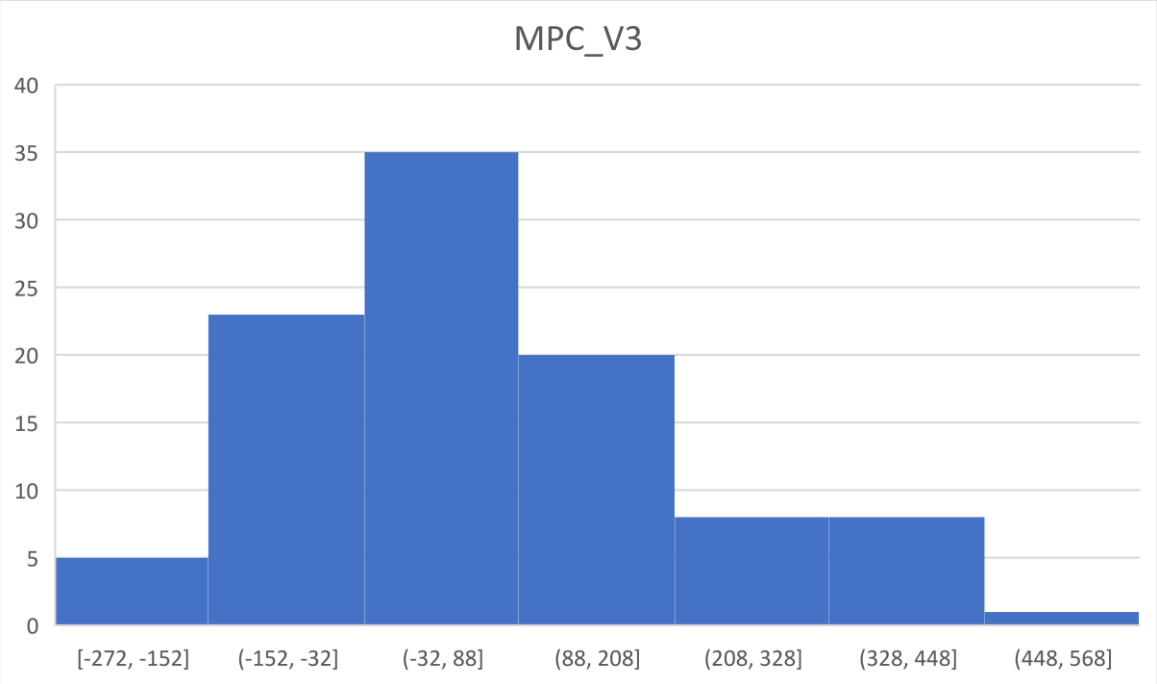

Figure 44: Distribution of the difference of time between benchmark and MPC_V3

Table 7 divides in quartiles the differences between the time returned by heuristic and the time of every approach, so a negative result means that heuristic was faster than MPC, instead positive result means the opposite.

The three histograms (Figure 42, Figure 43, Figure 44) contain, for every approach, the distribution of the difference between production time of benchmark and production time of MPC.

Table 6 highlights that 50% of MPC_V1 tests spends less or the same time than heuristic, MPC_V3 instead has this percentage equal to 64%.

Looking to Table 7 MPC_V1 presents positive and negative peaks wider than MPC_V3, but Figure 42 and Figure 44 show that on average MPC_V3 returns better times respect to MPC_V1.

Table 8 presents statistics on setups executed on average by every approach; obviously the type of control that characterize MPC_V1 and MPC_V2 brings to provoke more setups than heuristic. On the other hand, MPC_V3 demonstrate to be able to do not waste time in setups as expected.

Table 8: Statistics on setups

	HEURISTIC	**MPC_V1**	**MPC_V2**	**MPC_V3**
Average # of setups	35	38	36	31
Improvement (%)	100%	+9%	+3%	-11%
Variance	34	56	40	35
Max-Min value	49-20	51-18	50-19	46-14

Despite MPC_V2 should return better results than MPC_V1, is evident from all the results that the procedure of generating a variable predicted sequence of possible θ it's not a good solution.

In conclusion, the approach of controlling individually every node is a more robust solution than control the nodes all together, but is important to highlight that in some situations, manage the production in a synchronous way could return a better result.

Controlling the nodes with the same θ means emptying buffers *br* at the same time and most important means executing the setup in a coordinated way, sometimes this control technique permits to avoid bottlenecks.

4.4 Possible future works

Starting from the results just explained, in future could be interesting to proceed with the following improvements:

- Design a new version of the algorithm that units the approaches of MPC_V1 and MPC_V3; this new technique could be useful to control the nodes in an autonomous or in a coordinated way depending on whether one or the other approach is better every instant for the plant.
- A second improvement could be designing an algorithm that has the task to control all the plant and not only variable θ, obviously the level of complexity could increase but the results can be even more evident.
- Another interesting step could be considering a multi-product production with deadlines to respect and eventually also considering decay of raw material.

Bibliography

[1] E. Angel, E. Bampis e F. Kacem, «Energy Aware Scheduling for Unrelated Parallel Machines,» in *In Green Computing and Communications (GreenCom)*, IEEE International Conference , 2012, p. 533–540.

[2] K. Fang, N. Uhan, F. Zhao e J. W. Sutherland, «A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction,» in *Journal of Manufacturing Systems*, 2011, p. 234–240 .

[3] A. Cataldo, A. Perizzato e R. Scattolini, «Production scheduling of parallel machines with model predictive control,» in *Control Engineering Practice*, 2015, p. 28–40.

[4] T. Ávila, Á. Corberán, I. Plana e J. Sanchis, « The stacker crane problem and the directed general routing problem,» in *Networks*, 2015, p. 43–55.

[5] A. Cataldo e R. Scattolini, «Dynamic pallet routing in a manufacturing transport line with Model Predictive Control,» in *IEEE Transactions on Control Systems Technology*, 2016, pp. 1812-1819.

[6] K. So e C. Tang, « Optimal Operating Policy for a Bottleneck with Random Rework,» in *Management Science*, 1995, pp. 620-636.

[7] L. Magni e R. Scattolini, Advanced and Multivariable Control, Bologna, Italy: Pitagora Editrice.

[8] E. F. Camacho e C. Bordons, Model Predictive Control, Springer, 2 edition, 2007.

[9] F. D. Vargas-Villamir e D. E. Rivera, «A model predictive control approach for real-time optimization of reentrant manufacturing lines,» in *Computers in Industry* , 2001, p. 45–57.

[10] J. B. Rawlings e D. Q. Mayne, Model Predictive Control: Theory and Design, Madison, Wisconsin, USA: Nob Hill Publishing, 1 edition, Aug 2009.