DEPARTMENT OF MECHANICAL ENGINEERING
TECHNISCHE UNIVERSITÄT MÜNCHEN

Model-based diagnosis - Application to Matlab Stateflow models

Bachelor Thesis 2016

Pablo Martí Blasco

DEPARTMENT OF MECHANICAL ENGINEERING
TECHNISCHE UNIVERSITÄT MÜNCHEN

Model-based diagnosis - Application to Matlab Stateflow models

Bachelor Thesis 2016

Author: Pablo Martí Blasco
Supervisor: Prof. Dr. Julien Provost
Submission Date: September 8, 2016

I confirm that this Bachelor thesis is my own work and I have documented all sources and material used.
Munich, September, 2016 Pablo Martí Blasco

Content

1. Introduction

Nowadays, the industrial systems are becoming more and more complex and large, what is deriving in bigger problems to control the faults from the systems. The centralized method for systems diagnosis is becoming obsolete and useless against those enormous systems.

However, we cannot get behind because the technology of fault diagnosis is vital in a market with such a fierce competition as the one that we have these days.

In a big company where millions of products are manufactured every day, a breakdown of the system that can stop the production is something that cannot be permitted.

Speaking about costs, a good system of fault detection can save us a lot of money because it is essential for a trustworthy maintenance planning. It will increase the shelf life of the system, reduce the number of breakdowns and help us to actuate where and when a failure occurs.

Therefore it will reduce the costs of production and will decrease the price of our product in the market. That will help us to be more competitive against other firms.

The correct diagnosis of a fault in the system is not just the identification of the fault. It is composed of three steps [17]: first of all, detection, second isolation and in third place identification.

In the area of discrete event systems, different methods of fault diagnosis can be developed. We will be explain some of them along this thesis. We can split these methods in three main groups.

1) Ruled based expert systems use the knowledge of an expert to create logic rules. Those rules are triggered depending on the events that take place in our system (Papadopoulos and McDermid, 2001).

2) Data driven approaches need a data base to store the knowledge about the different faults we can have in the system.

3) Model base approach where a model of the system is built and observed to detect the occurrence of fault.

The approach that I have decided to use is a model base approach. This approach can be divided in another three categories depending on how the diagnoser is built [11].

Diagnosers can be built with a centralized approach where a global model is built and just one diagnoser is needed, a decentralized approach can be built with one global model or with several local models and local diagnosers are needed, with a distributed approach local models are built and local diagnosers are developed to control the system.

After studying those three options I have decided to use a decentralized approach because the centralized one can get too complex if we have a large system.

Once we have built the model we have to build the diagnoser. To build the model, expert knowledge is needed. However, the diagnoser can be built automatically using an algorithm. It is an automatic process that can be programmed.

In this thesis, an algorithm has been developed to change from a nondeterministic model to a deterministic one. The algorithm has been programmed in Matlab language and is applicable for the tool from Simulink named Stateflow.

Stateflow is an environment for modeling and simulating combinatorial and sequential decision logic based on state machines and flow charts. Stateflow allows combining graphical and tabular representations, including state transition diagrams, flow charts, state transition tables, and truth tables, to model how the system reacts to events, time-based conditions, and external input signals.

The algorithm is applicable in non deterministic automatons. An initial state has to be previously defined and the non observable events have to be defined as well and introduced in the program.

It is built to work in Stateflow. Nevertheless, I consider that it should not be hard to modify some commands and make it work in other programs because the main program is defined with standard language from Matlab and not the specific of Stateflow.

1.1 Organization of the document

This document is organized in 7 points:

Point 2: An introduction to fault diagnosis is going to be described , the explanation of what a fault is and basic concepts of diagnosis, including the difference between deterministic and non deterministic automatons.

Point 3: In this point, the criteria to classify the methods of diagnosis are developed and a longer explanation of rule-based expert systems, data driven approaches and model based is provided .

Point 4: As the model based approach is the one that has been chosen in this thesis, a detailed explanation about its types is expounded on this point.

Point 5: The algorithm that has been used to get a deterministic automaton drew from a non deterministic one is explained.

Point 6: Two practical cases are developed in this point to show how the algorithm works.

Attached 1: Code of the program in Matlab language.

Attached 2: Different models built in Stateflow.

2. Introduction to fault diagnosis in DES

In this part I am going to make a brief introduction to fault diagnosis. Meaning I will explain how it works as well as the basic fundaments of the fault diagnosis systems.

The fault diagnosis is based on the action of controlling a system in order to detect when a problem occurs in it, when something unexpected changes the running of our system.

Therefore I consider vital to make clear some points about those faults.

2.1. Faults

We need to start explaining what a fault is and how we can classify them before moving forward into fault diagnosis.

Definition 1. A fault can be defined as an abnormal behavior of the system.[1]

There are several types of faults and they can have different effects in our system. To classify these faults we can follow different principles:

i. The source of the fault: maintenance faults, human operator´s faults, wrong operating conditions faults, design faults, failure of components, software or hardware faults.
ii. The severity of the fault: whether a fault can affect the system but it can keep working with it or if the fault can cause the complete stoppage of the system.
iii. The time pattern of the fault: permanent, intermittent or incipient.
iv. The predictability of the faults: it can be predictable or unpredictable.

To sum up, a fault is an event that will lead the system to a different behavior which is not supposed to have. This new behavior can still accomplish the function of the system or not, depending on the type of fault. But either way it will be a faulty behavior.

2.2 Diagnosis

Now that we have made clear what a fault is we can proceed to explain the basis of diagnosis. We have already explained that fault diagnosis is used to

detect faulty behaviors of a system. However, the whole approach of this idea is not just to detect when the system is not working correctly but to detect, isolate and identify the origin of this failure behavior.

The detection consists on realizing when our system is not working as it should.

The isolation of the fault is to detect where this fail comes from.

The identification of the problem is to detect the severity of the failure.

In this thesis we are going to centre on the first two aspects: detection and isolation of the faults and as it is explained on the introduction, we are going to focus on fault diagnosis in discrete event systems (DES).

When we are working with a DES, it can be defined with Petri Nets, automatons and other definitions. In this thesis we are going to use the automatons to define our DES.

There are non deterministic finite automatons and deterministic finite automatons.

The non deterministic finite automaton is defined by: M = (Q,E, δ ,q0,F)

i. Q is the finite group of states
ii. E is the set of events
iii. q0 is the initial state
iv. F is the group of the last states
v. δ is the transition function

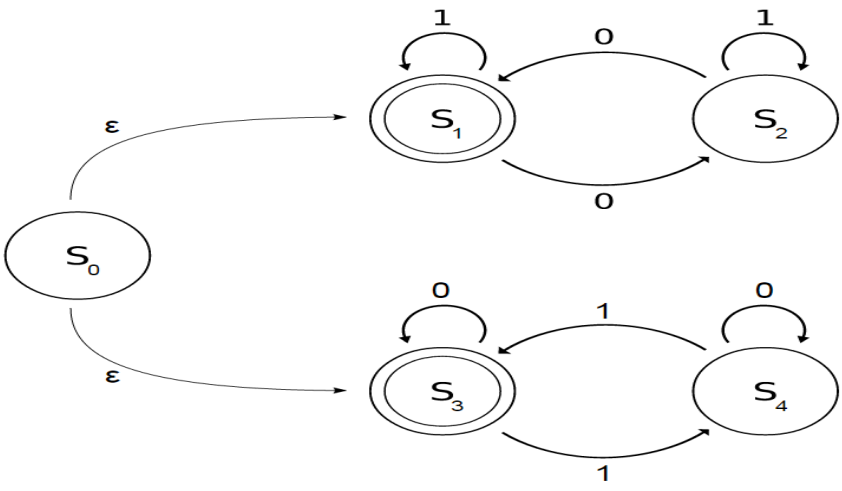

Figure 1

Inside the group of events, we can make two different groups that will be the key for the fault diagnosis. We can divide this group in observable and unobservable events. Faults, within other events, are considered as unobservable events. Otherwise their detection would not be a problem and all of these studies would not make any sense.

In the Figure 1 we can see the observable events as {1,0} and the unobservable event as {ε} and all of them will be included in E.

The aim of fault diagnosis is to detect those unobservable events, so what we need is to change the non deterministic automaton into a deterministic one that will be our diagnoser. The deterministic finite automaton is defined by: M = (Q,E, δ ,q0,F), just as the non deterministic ones.

However, all the events from E are observable. This automaton is what we have to build to be able to detect the faults.

An example can be found in the Figure 2 were we have the observable events {1,0} but we have not any unobservable events as {ε}.

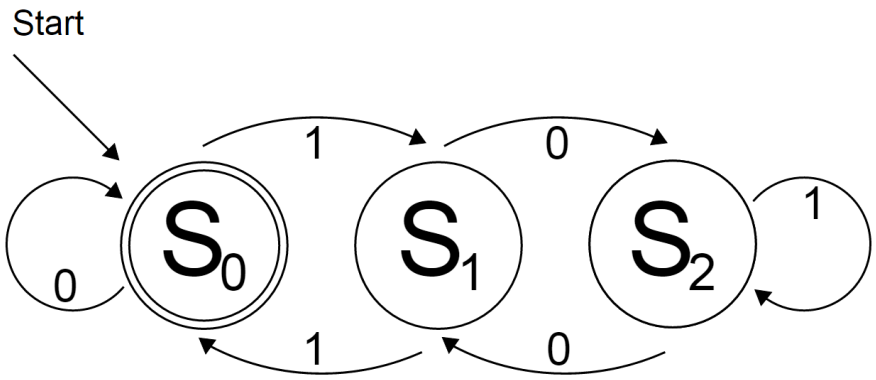

Figure 2

The new states of the observer are composed of one or more states of the non deterministic automaton. To create these new states, we have to look at the former automaton and see which states we can reach when an event occurs and which others we can reach with an unobservable event (The detailed explanation about how to do it can be found in point number 5).

We can classify the states of the diagnoser depending on whether they represent a state of failure, a normal state or an uncertain state:

i. Normal state: if our diagnoser reaches this state we can confirm that the behavior of the system is correct and that no fault has occurred.

ii. Faulty state: when our diagnoser has reached this state it unequivocally means that a fault has occurred and our system is having an abnormal functioning.

iii. Uncertain state: if our diagnoser reaches this state we cannot decide if the system is faulty or not. We have to wait until we reach one of the other type of states to decide whether or not our system is working properly.

Nevertheless, a fault is not always observable. This means that sometimes we will find a fault that we cannot detect with the diagnoser because our automaton keeps running between uncertain states. Therefore we cannot decide if the system is faulty or not. We can draw a new term from this fact: diagnosability. We can find the following definitions about diagnosability [3]:

Definition 2. A fault is diagnosable if it can be detected with certainty within a finite number of observable events after its occurrence. This means that fault f is diagnosable if for every execution trace s of events ending with f, there exists a sufficiently long continuation trace t such that any other execution trace indistinguishable from s * t – that is, that produces the same record of observable events as s * t – also contains f. (J.Zaytoon, S. Lafortune, 2013,p.313)

Definition 3. A fault is n-diagnosable if it can be detected with certainty within a specified number, n, of observable events after its occurrence. . (J.Zaytoon, S. Lafortune, 2013,p.313)

Definition 4. A system is diagnosable if it is possible to detect within a finite delay occurrences of faults of any type using the record of observed events. Alternatively speaking, diagnosability requires that every occurrence of every fault event leads to observations distinct enough to enable unique identification of the fault event within a finite delay. . (J.Zaytoon, S. Lafortune, 2013,p.314)

Theorem 1. The system is diagnosable if and only if there are no f-indeterminate cycles in the diagnoser for any fault type f. . (J.Zaytoon, S. Lafortune, 2013,p.313)

3. Classification of diagnosis methods

In the last years the fault diagnosis is becoming a more and more important field in automatic systems. Therefore the studies about the techniques of fault diagnosis have surprisingly increased since the late ninety's and the beginning of the new century.

There are different ways to classify these methods depending on the criteria we choose for it.

Some of these criteria are exposed on the article [3] :

i. Respect to fault compilation
ii. Respect to the modeling formalism
iii. Respect to fault representation
iv. Respect to the decision structure

Another classification is proposed (Papadopoulos and McDermid, 2001) on which we can classify most of the approaches that have been developed until now, which is the following:

i. Rule-based expert systems
ii. Data driven approaches
iii. Model based

These methods will be explained in more detail in the following pages.

3.1 Rule-based expert systems

These sytems are built with the knowledge of an expert about the system.

In the rule-based system, rules like IF-THEN-ELSE and an inference motor are used to get some conclusions of the process. These kind of systems are easy to build and implement for small systems. However they can become too big and complex if the system is more complicated.

The way a rule-based systems works is as follows.

The expert has to define the rules that will rule our system. They usually are IF-THEN-ELSE but can work as well with the type AND-OR. Once the system is built and we start to run the system, some rules will be triggered, which, at the

same time, can trigger other rules. In the end we will arrive at a point where no more rules can be triggered and the resulting information that we get is what will tell us whether or not a fault has occurred.

A graphic technique can be used in this kind of method named fault tree. The fault tree is made of nodes that are distributed in levels, representing the failure on the top and different sub failures behind, what will define the shape of a tree.

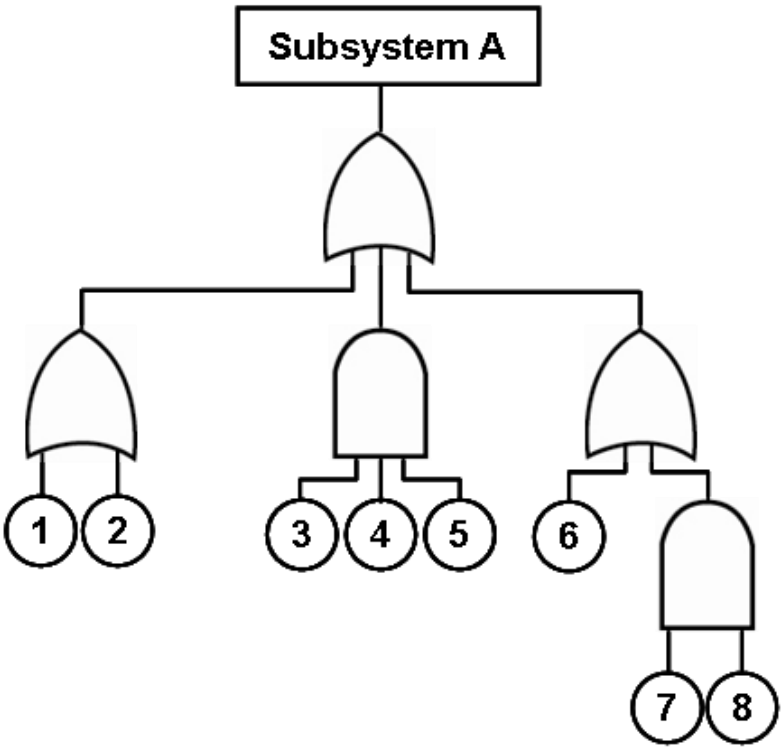

Figure 3

The different levels of the tree are connected with logic doors AND-OR. The functioning of this method is based on going from the top of the tree, where we have a complex event, to the lowest part. To go through the tree we use the logic doors until in the end we find a basic event that does not need other events to be explained, so we can define exactly what is happening in our system.

3.2 Data driven approaches

For this kind of methods what we need is an amount of historical registries of the system making to make the diagnosis. (Dash and Venkatasubramanian, 2000) We store in a knowledge data base the trends that we measure in the sensors when a fault occurs. With that, we will be able to identify if a fault is taking place in our system.

The way to proceed with this method is as follows:

i. First of all we need to create our data base of the faulty behaviors.
ii. Run the system.
iii. Record the new measurements of the sensors from the system.
iv. Compare the measurements of the sensors with our data base of faulty behaviors.

By comparing the current behavior of the system with the data base that we have stored, we will be able to identify if the system is running correctly or, in case that it is not, which is the fault that has occurred.

When we program a method like this one we have to make a compromise between robustness and time for the detection. The main idea has to be something in the middle where we do not use a chain of too long data, thus we are not able to detect the failure in time to react against it, but we cannot use a chain too small, hence we can take a normal behavior as faulty because we have not checked enough data.

One of the most used approaches for this method is the Neuronal Nets. They have very good properties for the diagnosis of failures, since they learn to diagnose failures thanks to the training of data, being also tolerant to the noise and to have good capacity of adaptation online.

3.3 Model based approaches

The general description of these methods (Matthias Roth. 2010) is that we compare the system with an expected behavior of the system defined by the model we have created.
To use this method, depth knowledge of the system is needed.

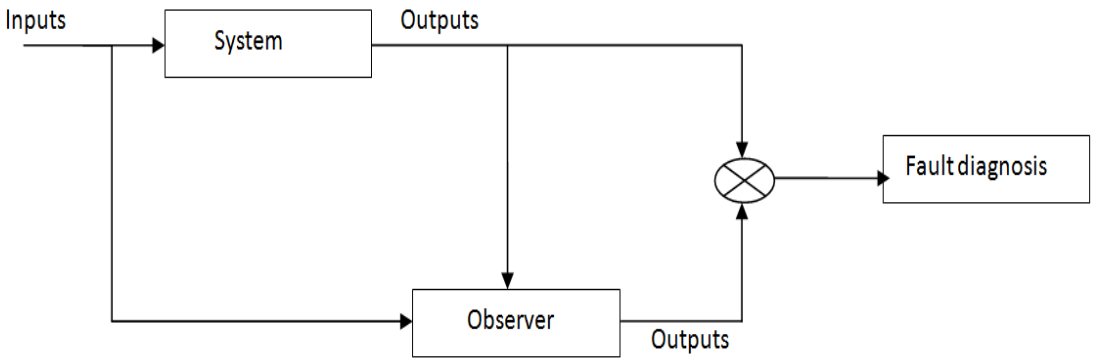

Figure 4

The different outputs between the system and our model will help us to decide if the system is working correctly or if we have a faulty system. Taking into account that not every difference between these signals has to be a fault, we have to be able to distinguish whether the difference is caused by noise of the system or a fault.

Various techniques are used to study those differences. In this thesis we are going to use a diagnoser.

Diagnosers are used to recognize those deviations. There are different types of diagnosers depending on the part of the system that we are observing or the connection and interaction between the observers.

In the next point, the different type of observer we can create is explained, as well as when we have to use them and their advantages and disadvantages. This part is explained in more detail because it is the approach that has been chosen to solve the practical cases that will be explained in the last point of the document.

4. Model Based approaches

The model based approach can be divided in two main groups:

i. Methods where we create a model that includes the faulty behavior of the system.
ii. Methods where only the fault-free model is built.

In this thesis we are going to use the first category that is showed above.

The model based approach needs three different structures: the model, the observer and the diagnoser. When we start with this method the first thing we build is the model. It represents the whole system, and it can be built following several criteria.

This model will have two types of events: observable and non observable events. Faults are considered non observable events. Therefore, this model will be a non deterministic finite automaton.

However, with this model we cannot see the faults because they are unobservable, so we need to build a deterministic automaton where we have not any of those events. The deterministic automaton that is built, drew from the non deterministic one, is named diagnoser.

The construction of the diagnoser can be made by an automatic algorithm and that is the biggest aim of this thesis. The algorithm has been developed using the language of Matlab and can be applied in the tool of Simulink named Stateflows. The algorithm will be explained in the last point.

Different algorithms can be used to build the observer and some references can be found in: [12],[13],[14].

Finally we need the structure that will be the system in charge of detecting whether or not a fault has occurred.

In the model base approach we use a structure named diagnoser that is able to make the diagnosis of the system and identify the faults. (Moamar Sayed-Mouchaweh,2014) The diagnoser can be built in three different ways.

i. Centralized
ii. Decentralized
iii. Distributed

Structures made by a centralized approach are characterized for having a global diagnoser that reaches the whole model.

When we build a decentralized structure we have to build several local diagnosers. This diagnoser cannot be connected with each other. However we have a coordinator that receives the information from those local diagnosers and make the final decision: if a fault has occurred or not.

In a distributed structure, what we build are several systems, instead of the global model that we used for the previous methods. Once that those subsystems are built we create a diagnoser for each of them that will make a local diagnosis of the system.

4.1 Centralized Diagnosis

As we have explained before, to apply a centralized diagnosis three main elements are needed:

i. Global model
ii. Global observer
iii. Global diagnoser

The procedure of this system can be explained following the next steps. First of all, as we want to do a centralized diagnosis, we will need to build the centralized model of the system. That means that the automaton we have to build includes all the process that takes place in our system, all the actuators from our system and all the sensors used on it.

This first model will be a non faulty model and then we will add the faulty behaviors to create the faulty model. Faults are considered non observable events. Therefore, this model will be a non deterministic finite automaton.

As we have built just one model for the whole system, the observer that we have to build will be a global observer as well. However, this new deterministic automaton is supposed to be much simpler than the model because it is made joining the states from the model to suppress the unobservable transitions we had.

As a consequence of having a global observer, the diagnoser will be a global structure as well.

When we use this approach the biggest problem we have is that, as we are using one single automaton to describe the entire process, it can become an enormous automaton if we are analyzing a complex system. Furthermore, we can have problems of robustness because if one part of the model fails it can cause the breakdown of the whole structure. Another problem we can find is the

difficulty to add new actualizations to the structure, because changing just a part of the real system is usually translated into changing the entire model we had.

Thus this approach is recommended for small system and systems that are not going to be developed in the future.

Behind we can see the schema of a centralized diagnoser.

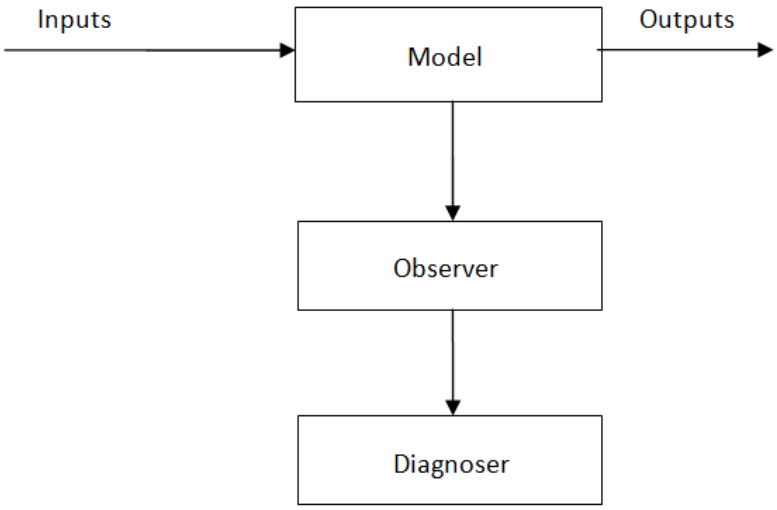

Figure 5

4.2 Decentralized Diagnosis

Before, the problems with a centralized structure have been exposed. Thus we need another method that will help us when we have to deal with a complex or large system. The decentralized diagnosis has been developed in order to overcome those problems.

In a decentralized diagnosis instead of building a huge global diagnoser we create several smaller diagnosers where each focuses on a part of the system; therefore it is simpler and can be modified easier without changing the whole model.

A decentralized approach is based on the idea of using local diagnosers to identify the failures of the model. Two ideas have been developed to carry out a decentralized diagnosis [11]:

i. A global model with local observers.
ii. Local models with one observer for each.

Several new concepts have to be explained to reach a complete understanding of this new approach.

Definition 5. A fault f occurring on a subsystem is locally diagnosable if there is a finite number of observations from the subsystem after the occurrence of f, so that we are sure that f has effectively occurred on the subsystem [15].

Definition 6. A subsystem is locally diagnosable if every fault occurring on that subsystem is locally diagnosable in the subsystem [15].

Definition 7. A fault is decentralized diagnosable if we can diagnose its occurrence with the collaboration of local diagnosers after a finite gap of time of its occurrence (Moamar Sayed-Mouchaweh, 2014).

We are going to start with the explanation of the global model with local observers because it is more similar to the one that has been explained before.

4.2.1 Global model with decentralized diagnosers

To apply this method the first thing we have to do is to build the global model of the process. It will be the same model as the one we create for a centralized diagnosis so until this point both methods are completely equal.

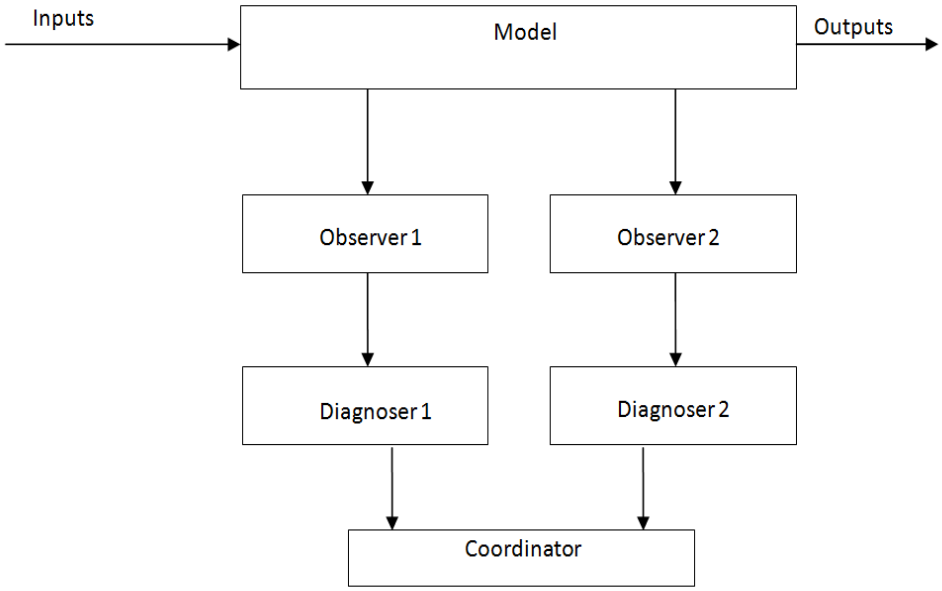

Figure 6

What will make the difference is the way to build the diagnosers. Now, instead of using just one for the whole system, various observers have to be developed. All of them will focus in a part of the system whereas none of them will have a complete vision of the model. They are not able to communicate with each other.

However, if it is necessary, we will create another system named the coordinator whose work is to use the information from the local diagnosers to reach a conclusion about the occurrence of a failure.

4.2.2 Local model with decentralized diagnosers

In this approach the steps that we have to follow change from the very beginning. Now, instead of building a global model we will have to build several local models.

Each of these models will be able to represent a part of the system and all of them together will be a representation of a real process. No global model is needed. Just with the local models we have to be able to solve the problem of diagnosability.

The second step is to build a diagnoser for each of these local models. With the diagnosers we can identify the faults that occur during the functioning of the system.

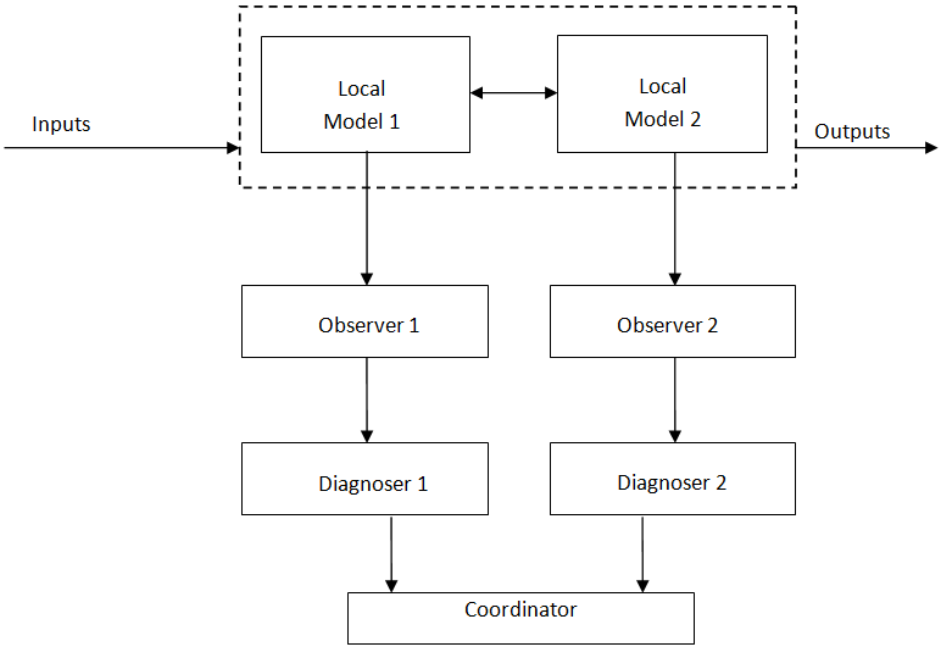

Figure 7

There is no communication between the local diagnosers. The ideal scenario would be that without anything more than the local diagnosers we could be able to identify every faulty behavior of the system.

However, that is not always possible and sometimes we need to use a coordinator to reach the conclusion whether or not a fault has happened.

4.3 Distributed Diagnosis

This point explains the basic notions of the distributed approach. This last approach can seem very similar to the distributed diagnosis based on local models (J. Kurien, X. Koutsoukos, F. Zhao, 2002).

The model that we have to build is the local models of the system. As it has been explained before, the local models focus on a part of the system and cannot show the whole process that we are studying.

The second step will be to build the diagnosers. They will be local diagnosers, one for each local model.

Until this point it can seem that we are describing the process to build a decentralized diagnoser. However we will see the difference between them in the next step.

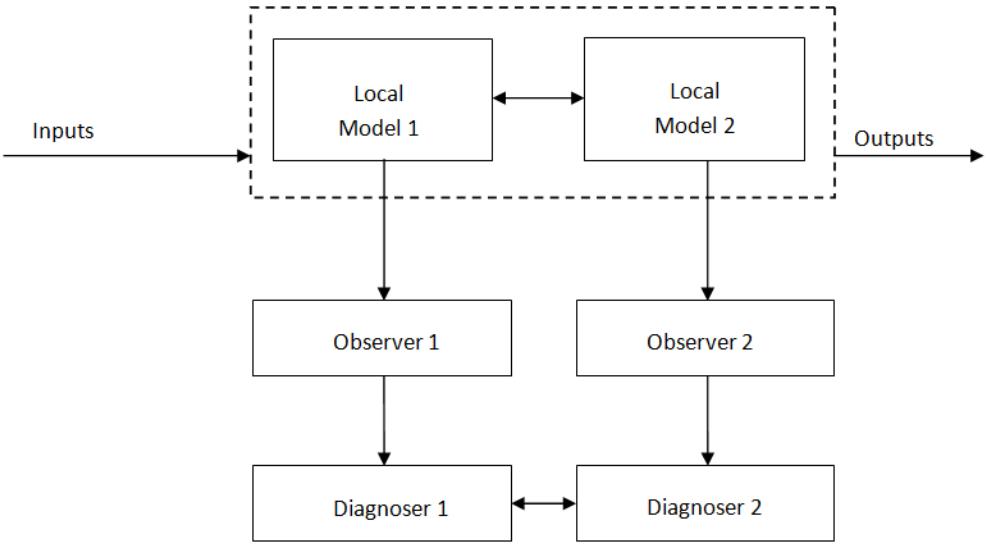

Figure 8

Once we have created the diagnosers we are not going to build the coordinator, as we should have done if we were building a decentralized diagnoser. Instead of that, in this method we have to develop the communication between the diagnosers because, on the contrary of the decentralized approach, the local diagnosers are able to communicate with each other.

This approach is characterized for being more flexible than the others; however, we have to reach a compromise between creating too many connections between the diagnosers that can lead us to a very complex system, and developing a connection too simple that can create a problem to detect a failure.

5. Algorithm

Along this thesis several approaches for fault diagnosis have been explained. Due to a personal preference I have chosen to work with a model base approach. As it has been explained before, the basis of this approach are three different structures: model, observer and diagnoser and in some cases the coordinator.

When these structures are represented by automatons we have explained that we can have two types of automatons:

i. Non deterministic finite automaton.
ii. Deterministic finite automaton.

The difference between them is that in the first one we have unobservable events whereas in the second one all the events are observable.

The model that we build from a system is considered a non deterministic finite automaton because it includes the faults that can take place in our process and these faults are considered as non observable events.

On the other hand, the structure that we need to develop for the diagnoser is a deterministic finite automaton where the non observable events will not appear.

The following algorithm has been developed to change from a non deterministic finite automaton to a deterministic finite automaton.

The method that we are going to use is an algorithm in which we will fill a table draw from the non deterministic finite automaton with the information that is needed to build the deterministic automaton.

On the first line we will compile all the observable events from our system.

On the first column we will gather the new states that will be part of the new automaton.

In the matrix between them we have to write the successors that will have the state of the first column in case of firing the event on the first line.

Now that we have a general idea of how the table will look like , we can explain how to fill it in more detail.

Considering a non deterministic finite automaton, it is defined by:

M = (Q, Σ , δ ,q0,F)

i. Q is the finite group of states
ii. Σ is the set of events
iii. q0 is the initial state
iv. F is the group of the last states
v. δ is the transition function

The steps that have to be followed in this algorithm are:

i. Build a table with one column for each e ∈ Σ
ii. In the first line we leave a blank space and we write all e ∈ Σ
iii. In the first column of the second line, we write the initial state I = E({q_0}), this corresponds to all the states that I can reach from q_0 with ε^* (all the non observable events).
iv. We fill every square from that line with $U_{r\in I}$ E(δ(r , a)), what means with all of the states that can be reached from I with $e_i\varepsilon^*$ (being e_i the event of the column i).
v. The new states that have appeared in the table have to be gathered in the first column.
vi. We jump to the next line and we do the same as we have done with the state I but with the new state R from the first column. We fill the line writing in each column e, $U_{r\in R}$ E(δ(r , a)). What means all of the states that we can reach from R with $e_i\varepsilon^*$.
vii. The step "v." and "vi." have to be repeated until we finish and run out of new states.

The process is showed in the following example.

5.1 Example

We have the automaton M:

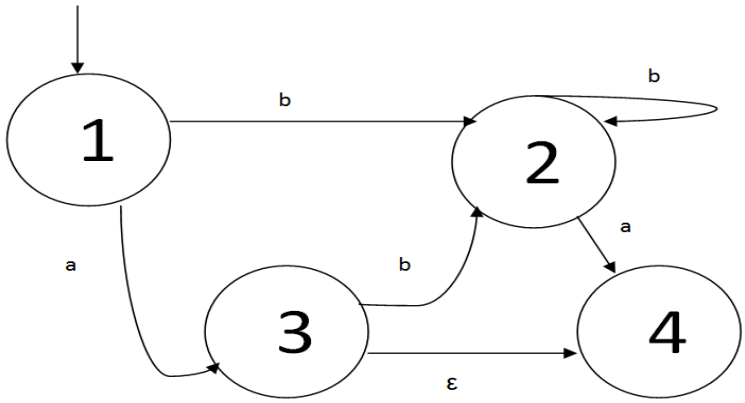

Figure 9

The initial state is 1. And we have the events {a, b, ε}. The events a and b are observable however ε is a non observable event.

Now we apply the algorithm to create the table:

	a	b
{1}	{3,4}	{2}
{2}	{4}	{2}
{3,4}	{}	{2}
{4}	{}	{}

The new deterministic automaton that we get is this one. More complex examples will be showed in the practical cases.

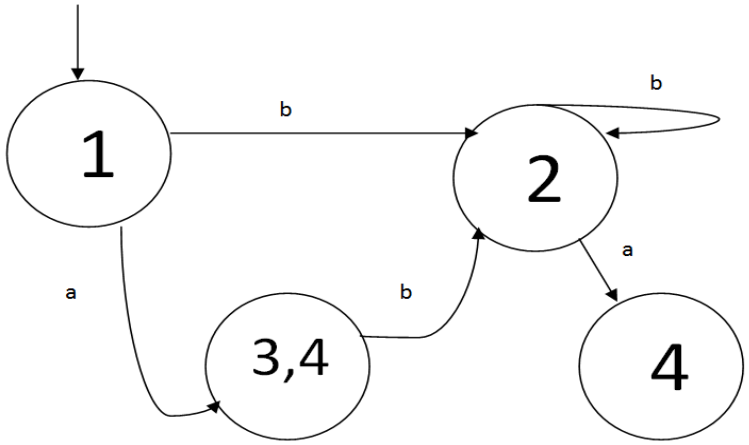

Figure 10

6. Practical cases

6.1 Automatic deposits.

6.1.1 Description of the process

The practical case that is going to be explained consists of the system to control two deposits that are connected with each other by two pipes. Those pipes can be regulated by two independent valves. Furthermore we have one drain in each deposit ruled by two valves, one for each, that can work independently as well. The deposits are filled with a pump that pours the water in the first deposit.

In this case we have two deposits as the ones showed in the figure 11. The process that is being performed follows the next steps:

1. All the valves are closed except V12h that is opened.
2. The pump starts working providing the deposits with a continuous flow of water.
3. The deposits are filled so the level detectors are turned on.
4. The deposit one is being filled so the first level detector that appears is L11.
5. When the water reaches the level of the valve the second deposit is filled. Then we will see the level detector L21.
6. We keep filling the second deposit so the L22 is reached.
7. When the water is at the same level in both deposits the level increases at the same time in both of them. Thus the next level reached is L12.
8. After L12 we will see the level detector L23.
9. Finally we reach LC that means that both deposits are full.
10. All the valves are opened except V12h that is closed.

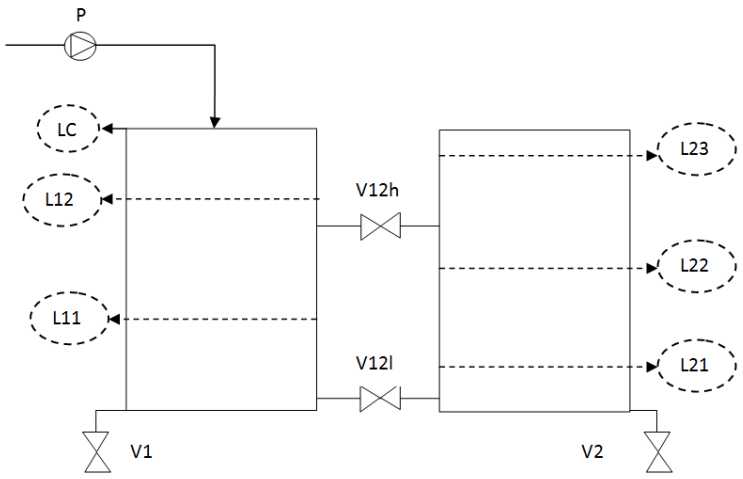

Figure 11

6.1.2 Fault free model

First of all we are going to build an automaton with the fault free behaviour, based on the process described above. The automaton could be smaller but the idea is to show clearly the steps from the process.

This automaton shows the whole process so it is a global model of the system under study.

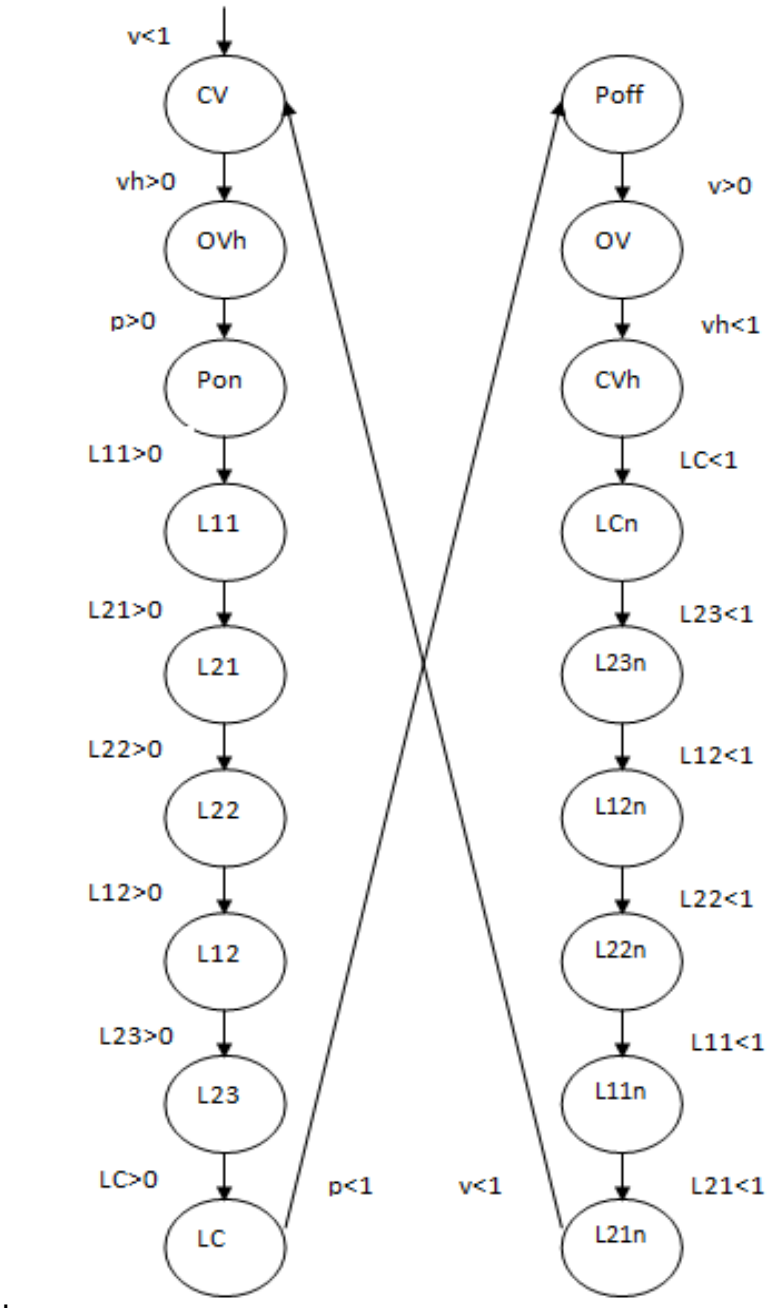

Figure 12

6.1.3 Description of the faults

To determine the failures that we have to identify, first of all we have to define those faults and the boundaries that we are going to apply for the simplicity of the system.

The faults are:

1. f1: failure of L22, it can get stuck on, showing that the water has reached that level.
2. f2: failure of L22, it can get stuck off, showing that the water has not reached that level.
3. f3: failure of the pump, it can get stuck during the process.
4. f4: failure of V12h, it can get stuck closed when we try to open it.
5. f5: failure of V12l, it can get stuck opened when we try to close it.

The rest of the components of the system can be considered reliable.

6.1.4 Diagnosis.

As we have seen, the automaton that we have built before is a global model. It is not specially big or complicated. In fact, it works following a sequential behaviour that is quite simple. However, if we build the faulty global model for this system it becomes much bigger and complex.

Thus, we have decided to change to a decentralized approach . Three local models will be developed for this system focusing on the detection of the faults that can occur in the system:

1. One for the detection of the failure of the level detector L22 (f1, f2).

2. A second local model to detect the failure of the valves (f4, f5).

3. The last one to control the pump (f3).

The models are depicted in the next page

Diagnosis L22

Fault free model L22

With this local model we are going to define a diagnoser to control the behaviour of the level detector L22. In this non faulty model the parts of the system that we are controlling are just the pump P and L22.

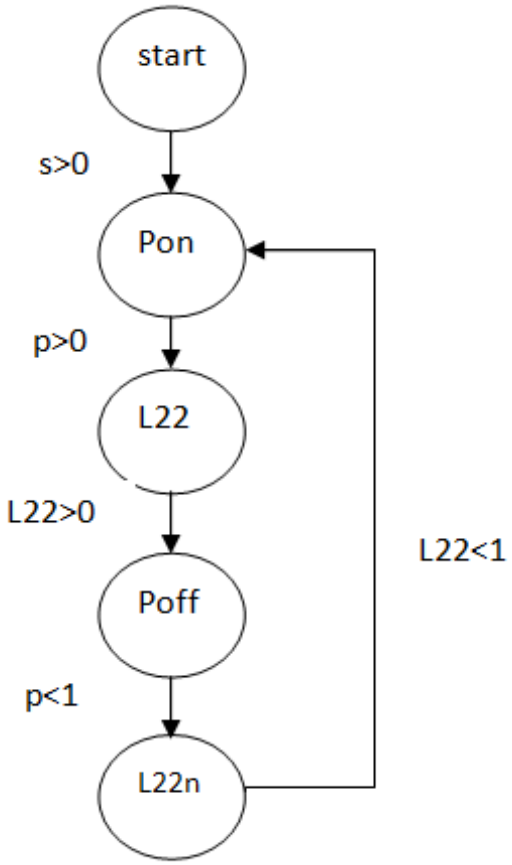

Figure 13

Faulty model L22

The faulty model is built including the detectors L22, L23. In this model we can see that the faults f2 and f1 are being checked. As we have explained before the faults are considered non observable events.
Therefore the type of automaton that we have is a non deterministic automaton.
The states that we have in the middle represent the normal behaviour of the system whereas the states in both sides are faulty states.

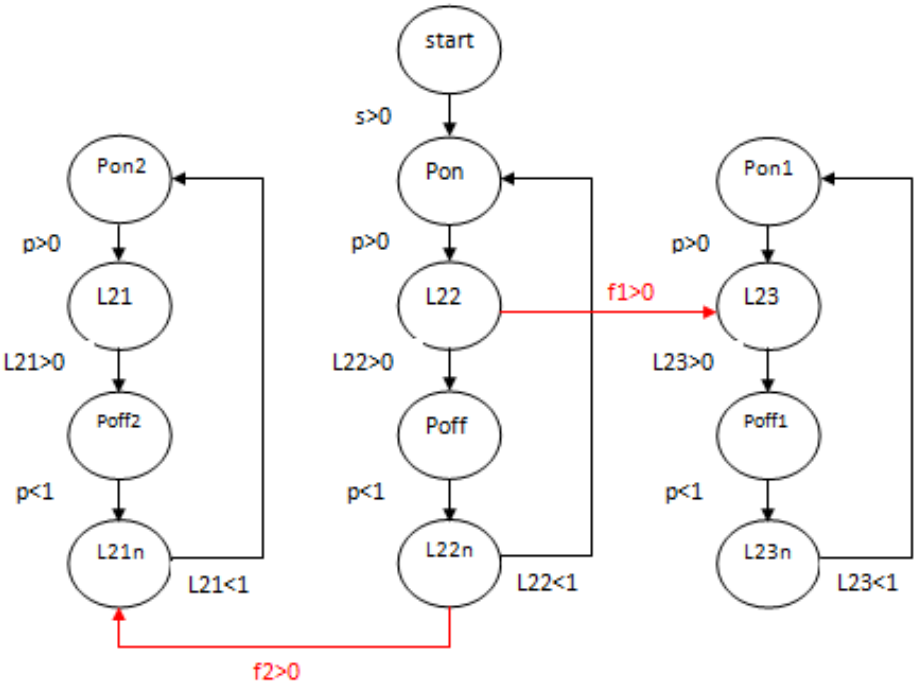

Figure 14

Table L22

Applying the algorithm that has been described in the point 5 to the model that we have built in Stateflow we will get the following table as an output. With the table we can easily create the diagnoser.

In the table we will have the observable event from this automaton on the highest line:
Σ={[b>0], [l21<1], [pm>0], [l23>0], [pm<1], [l22<1], [l23<1], [l21>0], [l22>0]}

In the first column we have the new states for the deterministic automaton:
Q={start, Pon, L22, Poff, L22n, Pon2, L21, Poff2, L21n, Pon1, L23, Poff1, L23n}

In the matrix we have the transitions between the states. We can see the transitions because the state in the first column is the source, the state in the matrix is the destination and the event above the destination is the event that fires the transition.

	[b>0]	[l21<1]	[l22<1]	[l23<1]	[pm>0]	[l21>0]	[l22>0]	[l23>0]	[pm<1]
Start	Pon								
Pon					L22,L23				
L22,L23							Poff	Poff1	
Poff									L22n, L21n
Poff1									L23n
L22n, L21n		Pon2	Pon						
L23n				Pon1					
Pon2					L21				
Pon1					L23				
L21						Poff2			
L23								Poff1	
Poff2									L21n
L21n		Pon2							

Diagnoser

In the figure 15 we can see the diagnoser that has been built using the data from the table. The states with the N are the normal states, are those states that confirm that our system is working correctly.
The states with the U sae those states where we cannot say whether or not the functioning is faulty.
And the states with the F are those states that represent a faulty behaviour of the system.
With this diagnoser we can determine when the level detector is faulty. Furthermore we can recognise if it is the fault type f1 or f2. The states on the right represent the states of the fault 2 and the states on the right are the states of the fault 1.

So we can confirm that this local system is diagnosable because all the faults can be detected.

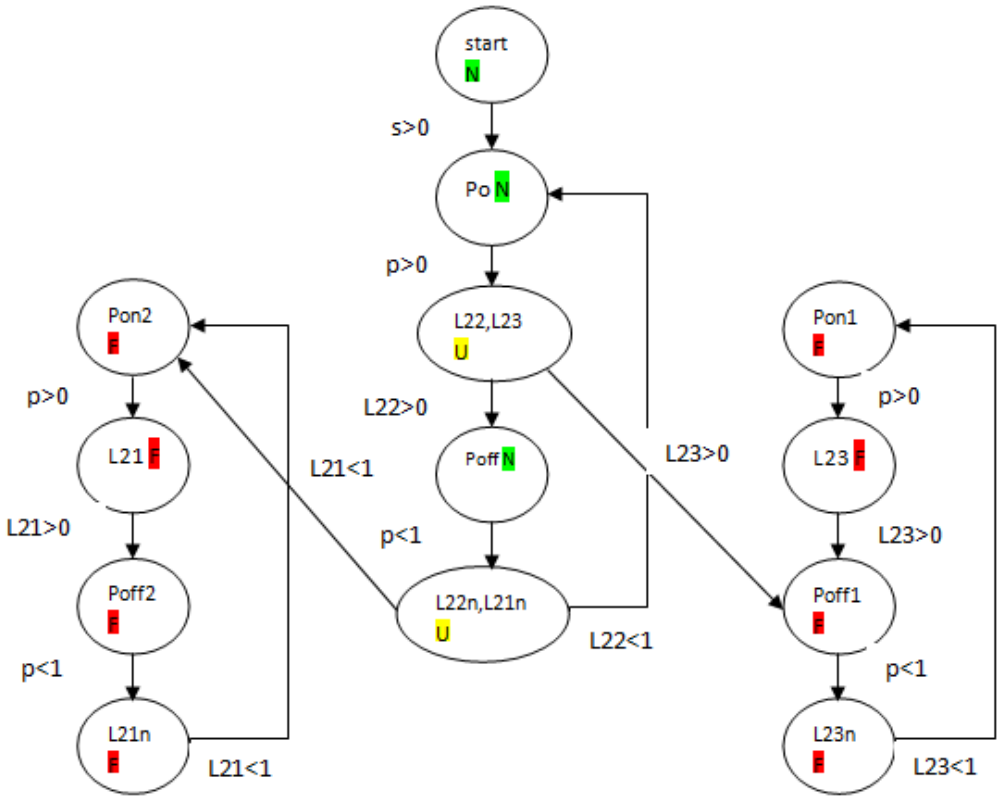

Figure 15

Diagnosis Valves

Fault free model Valves

With this local model we are going to define a diagnoser to control the behaviour of the valves between deposits. In this non faulty model the parts of the system that we are controlling are the pump P, L11, L21 and L12.
The use of the visor L22 is avoided because it is not trustworthy.

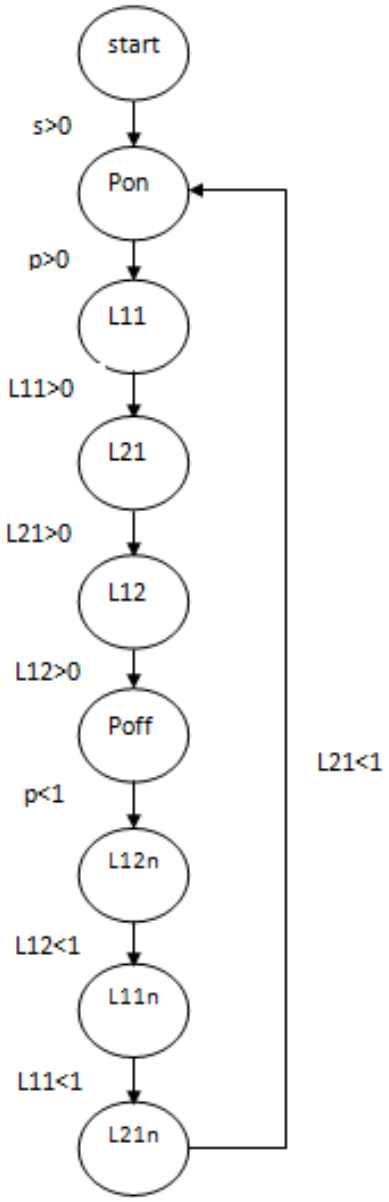

Figure 16

Faulty model Valves

In this model we can see that the faults f5 and f4 are being checked. As we have explained before in the faults are considered non observable events.

This is a non deterministic automaton so we will apply the algorithm to get the new diagnoser for this model.

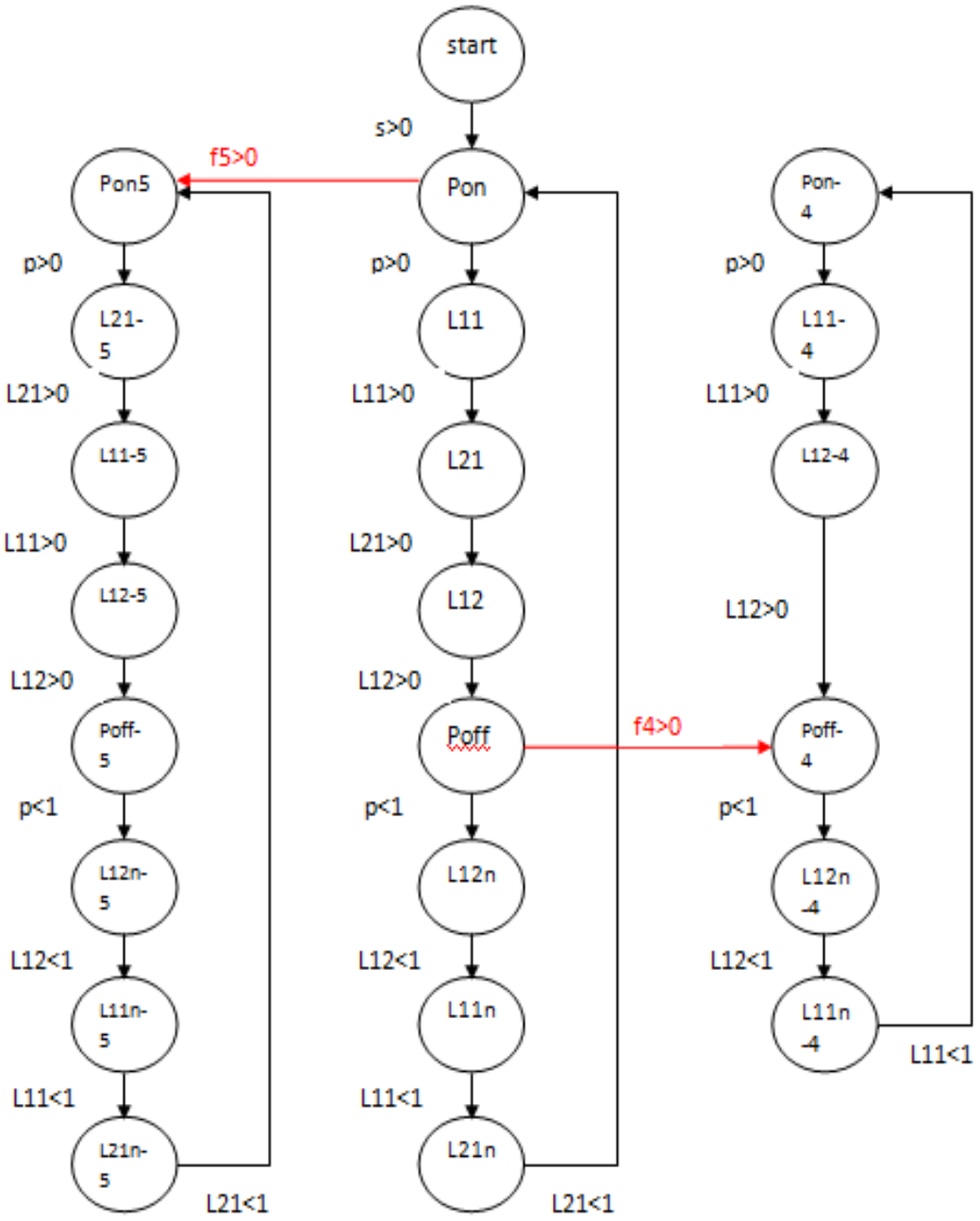

Figure 17

Table Valves

The table that we will get from the algorithm this time will be much bigger because now we have the double number of states. But it will work the same way.

In the table we will have the observable event from this automaton on the highest line:
Σ={[s>0], [l21<1], [l11<1], [pm>0], [l21>0], [pm<1], [l12<1], [l11>0], [l12>0]}

In the first column we have the new states for the deterministic automaton:
Q={ Start, Pon, Pon-5, L21-5, L11,L11-5, L21, L12-5, L12, Poff-5, Poff, Poff-4, L12n-5, L12n, L12n-4, L11n-5 L11n, L11n-4, L21n-5, Pon-4, L21n, Pon-5, L11-4, L21-5, L12-4, Poff-4, L12n-4, L11n-4, Pon-4}

	[s>0]	[l21<1]	[l11<1]	[pm>0]	[l21>0]	[l11>0]	[l12>0]	[pm<1]	[l12<1]
start	Pon, Pon-5								
Pon, Pon-5				L21-5, L11					
L21-5, L11					L11-5	L21			
L11-5						L12-5			
L21					L12				
L12-5							Poff-5		
L12							Poff, Poff-4		
Poff-5								L12n-5	
Poff, Poff-4								L12n, L12n-4	
L12n-5									L11n-5
L12n, L12n-4									L11n, L11n-4
L11n-5			L21n-f5						
L11n, L11n-4			Pon-4, L21n						
L21n-5		Pon-5							
Pon-4, L21n		Pon, Pon-5		L11-4					
Pon-5				L21-5					
L11-4						L12-4			
L21-5					L11-5				
L12-4							Poff-4		
Poff-4								L12n-4	
L12n-4									L11n-4
L11n-4			Pon-4						
Pon-4				L11-4					

Diagnoser

With this diagnoser we can determine when one of the two valves is faulty and we can recognise if it is the high or the low valve. The states on the right represent the states of the fault 4 and the states on the left are the states of the fault 5.

So we can confirm that this local system is diagnosable because all the faults can be detected.

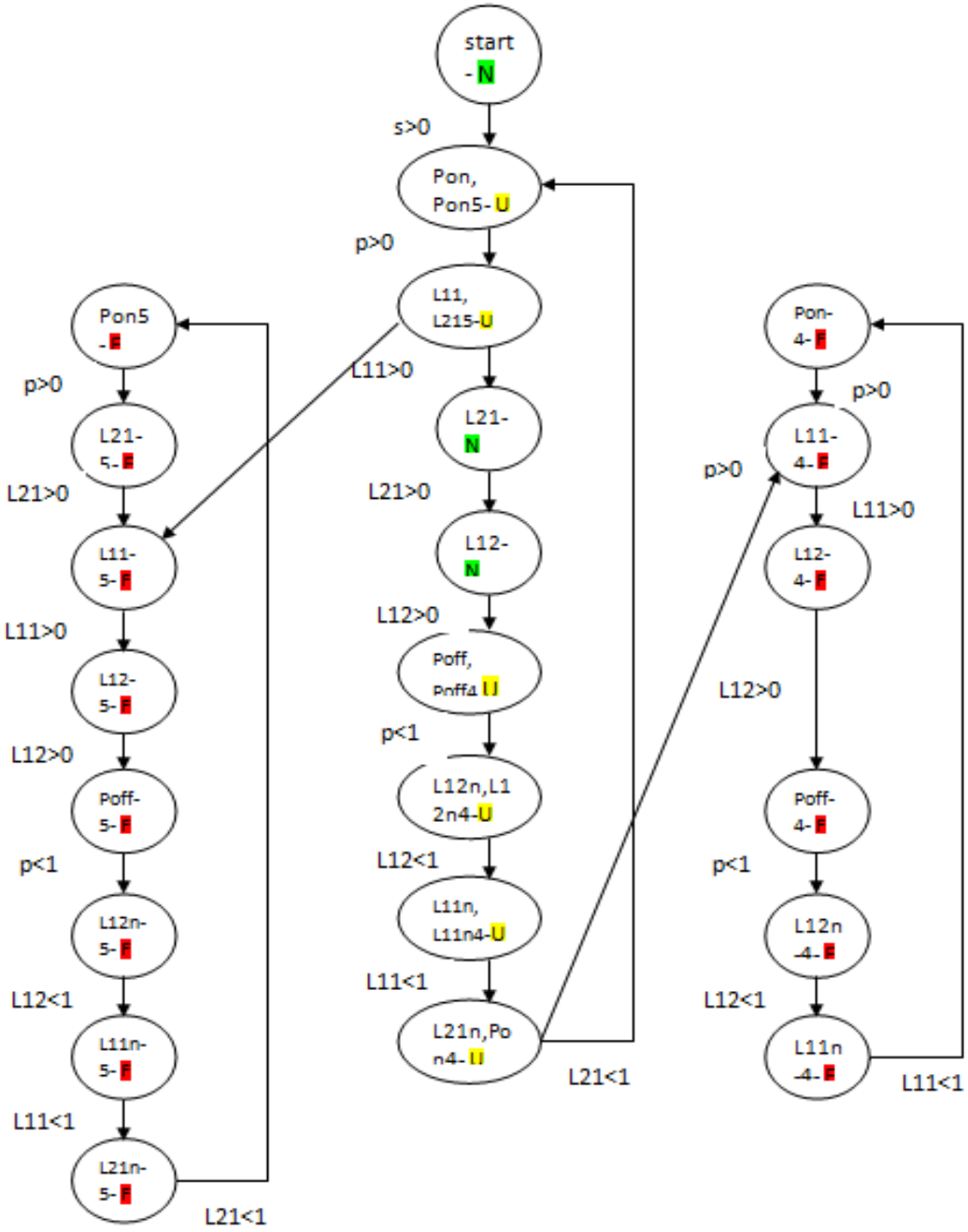

Figure 18

Diagnosis Pump Stuck

Fault free model

In this model, the process of the pump is explained. Just the Pump, and the level detector L11 and LC are used. I have decided to use those detectors because they are not affected by any other faults.

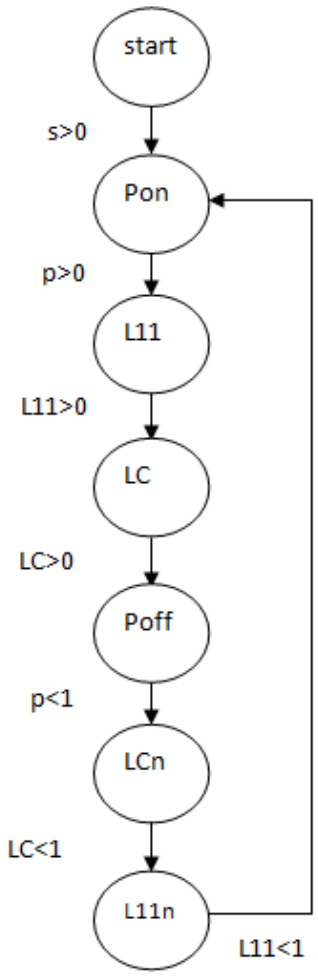

Figure 19

Faulty model

The fault f3 is being checked. This is the fault of the pump, because the pump can get stuck while the deposits are being filled and it can stop the water flow. We can see that the faulty states are located on the right side of the automaton while the normal states are in the middle.

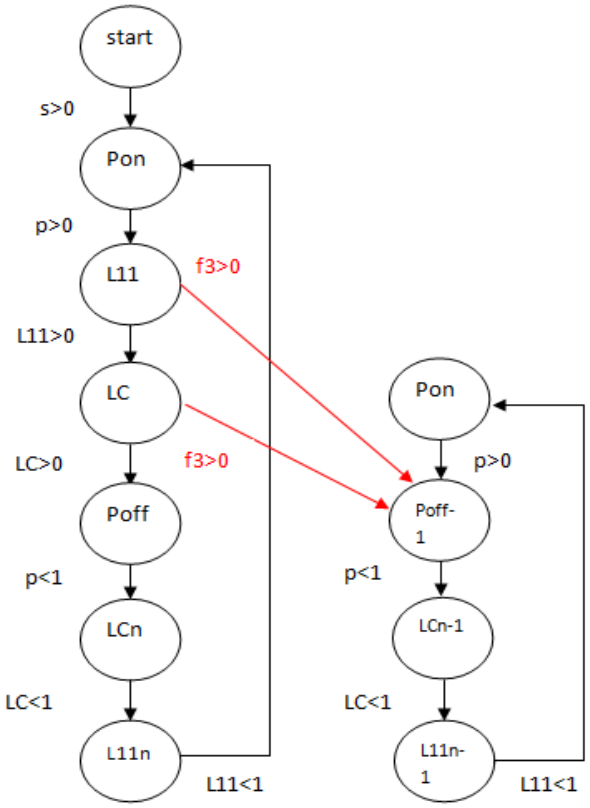

Figure 20

Table

Below we can find the table that the algorithm will give us. As always, the new states are on the left, the events on the top and we can find the transitions with the matrix left in the middle.

	[s>0]	[l11<1]	[pm>0]	[l11>0]	[lc>0]	[pm<1]	[lc<1]
Start	Pon						
Pon			L11, Poff1				
L11, Poff1				LC, Poff1		LCn1	
LC, Poff1					Poff	LCn1	
LCn1							L11n1
Poff						LCn	
L11n1		Pon1					
LCn							L11n
Pon1			Poff1				
L11n		Pon					
Poff1						LCn1	

Diagnoser

With this diagnoser we will be able to identify the behaviour of the pump. As we have done before we have three possible states.
We can see that the loop in the middle is composed by states classified as unknown, or normal. However the loop on the right is composed by faulty states. Therefore we can conclude that the fault is diagnosable because as soon as the process falls in that loop it means that we are having a faulty behaviour.
It can be explained as follows. We start running the middle loop but suddenly while the pump is supposed to be working it get stuck. When this happens the level of the deposits will not change anymore. Therefore the next event that we will see is that the pump will be turned off, because it is programmed to do it and we will identify that the pump is stuck.

Some other approaches can be proposed, such as the use of templates, that will allow us to detect this fault more quickly.

We can conclude that this system is diagnosable.

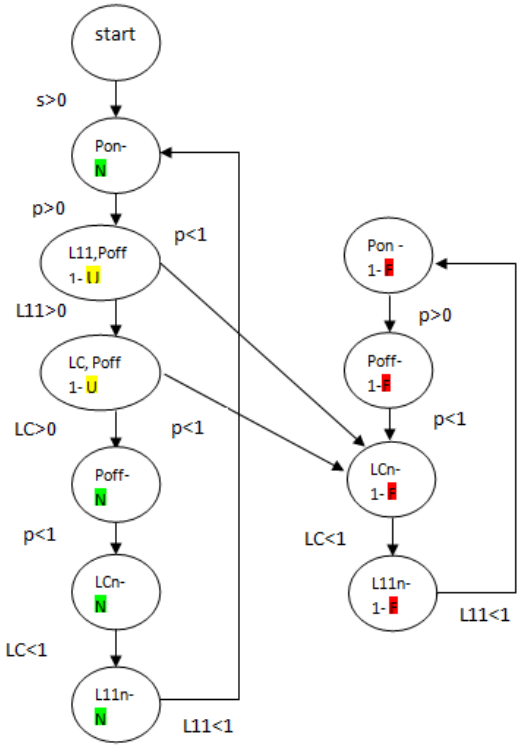

Figure 21

6.2. Heat, Ventilating and Air Conditioning

6.2.1 Description of the process

The practical case that is going to be explained consists of the system to control a HVAC (Heat, Ventilating and Air Conditioning). We will use just the part that is depicted in the picture below.

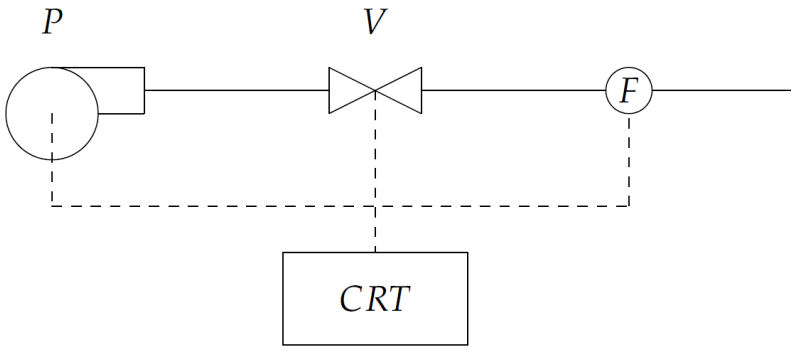

Figure 22

We have three main components:

1. Pump P, that produces the air flow in the piping. It can be switched on and off by the controller CRT.

2. Valve V, that can stop the flow. It can assume two positions, namely completely open or closed.

3. Flow sensor F, that detects the presence of an air outflow in the pipe.

4. Controller CRT, that determines the behaviour the plant by driving the pump and the valve and checking the sensor reading.

The normal process that is going to rule our system follows the next steps:

1. Open valve.
2. Start pump.
3. Stop pump.
4. Close valve.

And it starts again.

6.2.2 Fault free model

First of all I am going to build an automaton with the fault free behaviour, based on the process described above.

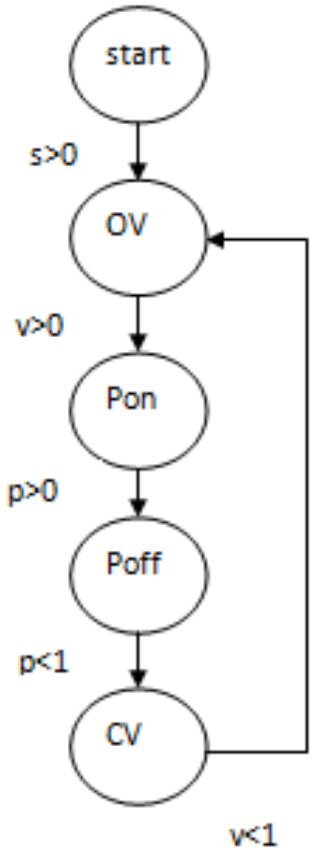

Figure 23

6.2.3 Description of the faults

In general, we will consider all the components as faultless, apart from the valve.

Two different faults can occur:

1. f1: the valve can get stuck-open.
2. f2: the valve can get stuck-close.

6.2.4 Diagnosis.

In this case the model is really simple. Therefore, a global model has been chosen to represent the functioning of the system.

However, two local diagnosers are going to be created, one for each fault, so the identification of the fault will be easier and the diagnosers will be much simpler than one global diagnoser for both faults.

Diagnosis Valve Stuck Open

Faulty model

As we can see the faulty model has two loops. Both of them are similar and related with each other by the faulty event.

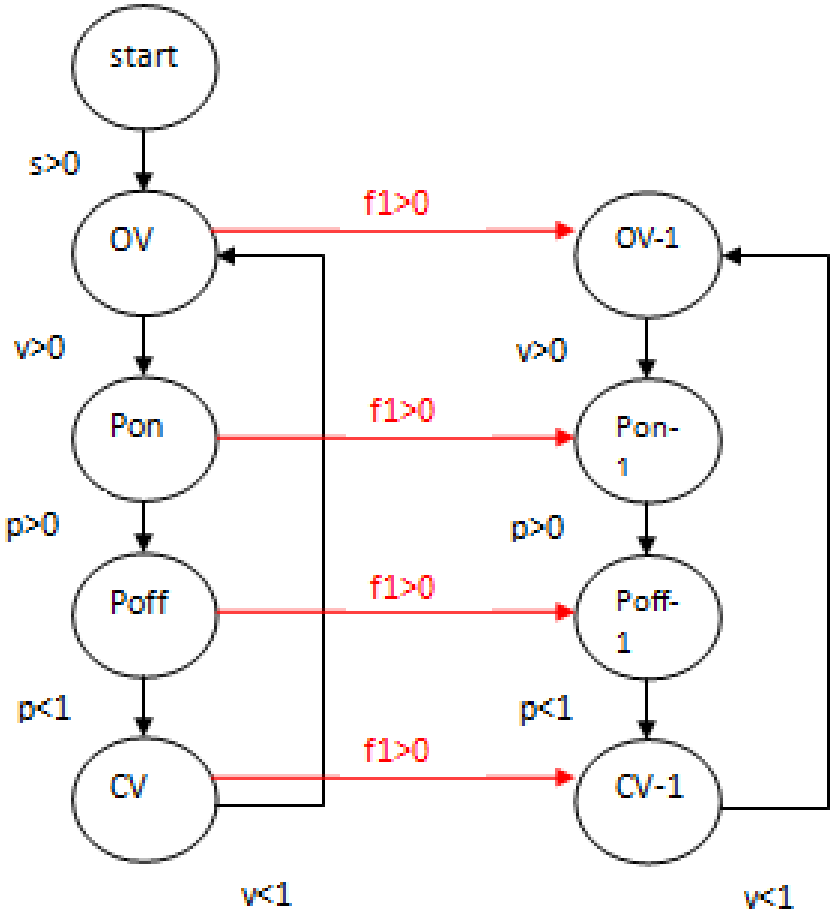

Figure 24

Table

	[s>0]	[v<1]	[v>0]	[p>0]	[p<1]
Start	Ov, ov1				
Ov, Ov1			Pon. pon1		
Pon. Pon1				Poff, Poff1	
Poff, Poff1					Cv, Cv1
Cv, Cv1		Ov1,ov			

Diagnoser

In this diagnoser we can clearly see that all the states that we have ,except for the start, are states of the type unknown. Thus, we will not be able to identify if our system is running correctly or the fault has occurred.
Therefore we can conclude that the fault 1 is non diagnosable.

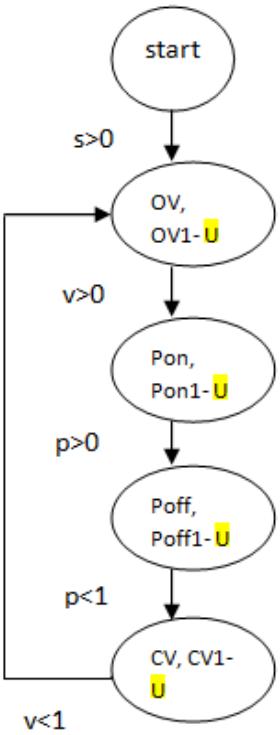

Figure 25

Diagnosis Valve Stuck Close

Faulty model

The faulty model that we build from the previous model is depicted below. We can see that we are using the detector of the flow and it will be the key to identify the faulty behaviour.
We have one more states that have not appeared before an it represents the sensor of the flow.

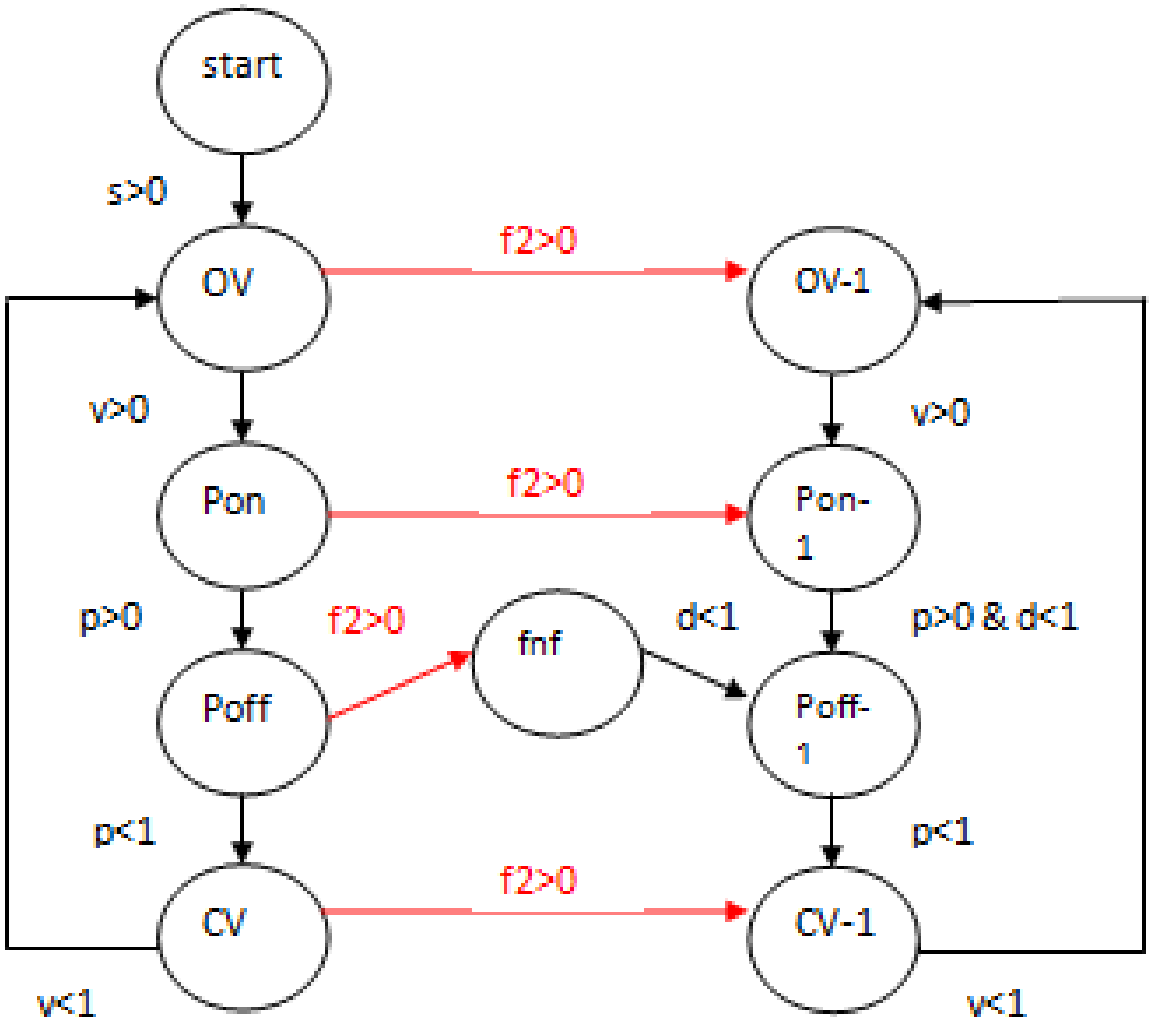

Figure 26

Table

The algorithm will give us the next table.

	[s>0]	[v<1]	[v>0]	[p>0]	[p>0&&d<1]	[d<1]	[p<1]
Start	Ov, ov1						
Ov, ov1			Pon. pon1				
Pon. pon1				Poff, fnf	Poff1		
Poff, fnf						Poff1	Cv, cv1
Poff1							Cv1
Cv, cv1		Ov1,ov					
Cv1		Ov1					
Ov1			Pon1				
Pon1					Poff1		

Diagnoser

The diagnoser for the previous faulty model is depicted below. As we can see we have two types of states: unknown states which can be recognised with a U and faulty states which can be recognised with an F.

Thus, even though we will not be able to know whether our system is faulty or not until we reach the transition [d<1], we will be able to identify the fault.

Therefore we can conclude that the fault 2 is diagnosable.

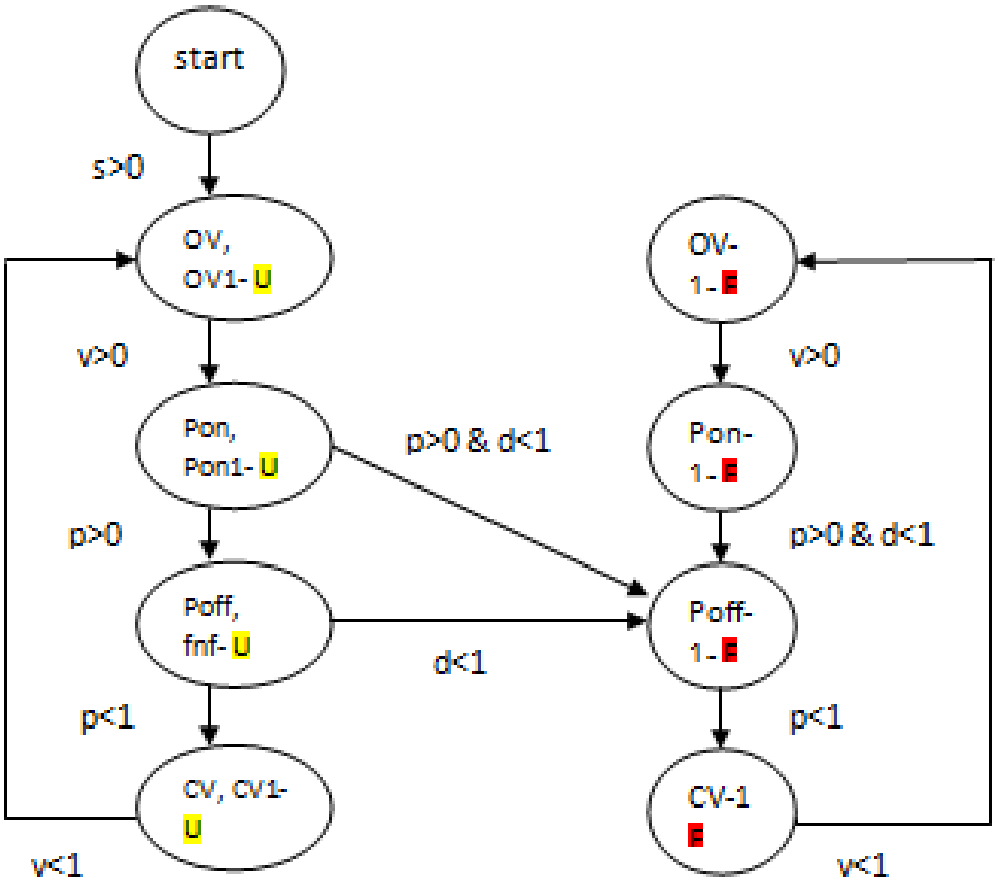
start
s>0
OV,
OV1- U
v>0
Pon,
Pon1- U
p>0
Poff,
fnf- U
p<1
CV, CV1-
U
v<1
p>0 & d<1
d<1
OV-
1- F
v>0
Pon-
1- F
p>0 & d<1
Poff-
1- F
p<1
CV-1
F
v<1

Figure 27

Attached Program

Main code

```
%We are going to use this program to create a table with all the
information that is necessary to build the deterministic automaton.

rt=sfroot;
m = rt.find('-isa', 'Simulink.BlockDiagram','-
and','Name','DepositSystem');
ch=m.find('-isa', 'Stateflow.Chart', '-and', 'Name', 'FaultyPump');
s=ch.find('-isa','Stateflow.State');
t=ch.find('-isa','Stateflow.Transition');
R={};
e={};
f=1;
R{1,1}={'start'};
e=VectorEventos(t,e);
C=cell(length(e));

while f<=length(R)
    C=CreateMatrix(R,C,f,e,t);
    [R]=CreateStates(R,f,e,C);
    f=f+1;
end;
```

Functions

Vector Events

```
%We are going to use this function to create a vector with all the
events
%from our automaton.
function [e]=VectorEventos(t,e)
i=1;
j=1;

%Inside t we have all the transitions from our automaton.
% With this loop we find every Label (events) from those transitions.
while i<=(length(t))
    w=t(i).LabelString;
    z={w};
    o=0;
    switch w
        case '[f1>0]'
            o=1;
        case '[f2>0]'
            o=1;
        case '[f3>0]'
            o=1;
        case '[f4>0]'
            o=1;
        case '[f5>0]'
            o=1;
```

```
        otherwise
            m=0;
            %We have already checked that the event is observable and
now
            %we have make sure that we have not already kept it in the
            %vector of events.
            while m<(length(e))
                m=m+1;
                if strcmp(e{1,m},z)
                    o=1;
                end

            end

    end
    if o==0
        e{1,j}=z;
        j=j+1;
    end
    i=i+1;

end
```

Create States

```
%With this function we create the vector of the new states in the
%determinnistic automaton.
function [R]=CreateStates(R,f,e,C)
j=1;
%With this loop  we go over all the new states in the line of the
matrix
while j<=length(e)
    esta=0;
    if  1> isempty(C{f,j})
        k=1;
        while k<=length(R)
            %With this two loops we compare the state that we have in
the
            %matrix with the state from R.
            in=0;
            m=1;
            while m<=length(C{f,j})
                n=1;
                while n<=length(R{k,1})

                    if strcmp(R{k,1}(1,n),C{f,j}(1,m))
                        in=in+1;
                    end
                    n=n+1;
                end
                m=m+1;
            end
            %with this "if" we check if the state is the sameone as
one that
            %is already in R
            if in==length(R{k,1})
                esta=1;
```

```
                k=length(R);
            end
            k=k+1;
        end
        if esta==0
            n=length(R)+1;
            R{n,1}=C{f,j};
        end
    end
    j=j+1;
end
```

Create Matrix

```
%In this function we are going to create a matrix with all the new
possible
%states for the deterministic automaton.
function [C]=CreateMatrix(R,C,f,e,t)
%We have to fill the line of the matrix with the successors from the
states in the
%array of new states
%With this while we find wich transitions have the state that we have
%as a source. We need to check every state from the cell.
if length(R)>length(e)
    C{length(R),length(e)}=[];
end
a=1;
while a<=(length(t))
    w=t(a).Source.Name;
    n=1;
    in=0;
    while n<=length(R{f,1})
        if strcmp(R{f,1}(1,n),w)
            in=1;
        end
        n=n+1;
    end
    if in==1
        u=t(a).LabelString;
        %once that we have found it, we need to know wich is the event
        %that we´ll lead us to a new state.
        b=1;
        while b<=(length(e))

            if strcmp(u,e{1,b})
                h={t(a).Destination.Name};

                if isempty(C{f,b})
                    C{f,b}=h;
                    l=1;
                    %once that we have found it we storage it in the
cell
                    %and we check if we can go from this state to
another
                    %one  with an unobservanle event.
                    [C]=SearchFail(t,C,b,f,l);

                else
```

```
                    l=length(C{f,b});
                    l=l+1;
                    C{f,b}(1,l)=h;
                    [C]=SearchFail(t,C,b,f,l);

                end
            end
            b=b+1;
        end

    end
    a=a+1;
end
```

Search Fail

```
%In this function we check if we can go from this state to another one
with an unobservanle event.
function [C]=SearchFail(t,C,b,f,l)
a=1;
m=length(C{f,b});

%We find the transitions from the state and we see if the event that
fires
%them is one of the fails (unobservable events).In case that we have
it, we
%write the other event in the cell.
while a<=(length(t))
    w={t(a).Source.Name};
    if strcmp(w,C{f,b}(1,l))
        u=t(a).LabelString;
        switch u
            case '[f1>0]'
                m=m+1;
                c={t(a).Destination.Name};
                C{f,b}(1,m)=c;


            case '[f2>0]'
                m=m+1;
                c={t(a).Destination.Name};
                C{f,b}(1,m)=c;


            case '[f3>0]'
                m=m+1;
                c={t(a).Destination.Name};
                C{f,b}(1,m)=c;


            case '[f4>0]'
                m=m+1;
                c={t(a).Destination.Name};
                C{f,b}(1,m)=c;
```

```
            case '[f5>0]'
                m=m+1;
                c={t(a).Destination.Name};
                C{f,b}(1,m)=c;


        end
    end
    a=a+1;
end
```

Bibliography

[1] Gertler, Janos J. (1998). *Fault detection and diagnosis in engineering systems*. New York:Marcel Dekker. 11

[2] Sampath, M., Sengupta, R., Lafortune, S., Sinnamohidden, K., & Teneketzis, D. (1995). Diagnosability of discrete event systems. IEEE Transactions Automatic Control, 40, 1555–1575.

[3] J.Zaytoon, S. Lafortune (2013). Overview of fault diagnosis methods for Discrete Event Systems. Annual Reviews in Control 37 (2013) 308–320

[4] Papadopoulos, Y. and McDermid, J. (2001). Automated safety monitoring: A review and classification of methods. International Journal of Condition Monitoring and Diagnostic Engineering Management, 4(4):14_32.

[5] Dash, D. and Venkatasubramanian, V. (2000). Challenges in the industrial applications of fault diagnostic systems. Computers & Chemical Engineering, 24(2-7):785_791.

[6] Matthias Roth. Identification and fault diagnosis of industrial closed-loop discrete event systems. Other. Ecole normale superieure de Cachan - ENS Cachan, 2010. English. <NNT : 2010DENS0028>. <tel-00561906>

[7] Cassandra C-G, Lafortune S (2008) Introduction to Discrete Event Systems, 2nd edn. Springer, New York Inc

[8] Isermann R (2005) Model-based fault-detection and diagnosis: status and applications. Annu Rev Control 29:71–85

[9] Jéron T, Marchand H, Pinchinat S, Cordier M-O (2006) Supervision patterns in discrete event systems. 17th International Workshop on Principal of Diagnosis, pp 117–124

[10] Jiang S, Kumar R (2004) Failure diagnosis of discrete event systems with linear-time temporal logic specifications. IEEE T Automat Contr 49(6):934–945

[11] Moamar Sayed-Mouchaweh (2014) Discrete Event Systems. Diagnosis and Diagnosability. London. Springer

[12] Gascard Eric, Zineb Simeu-Abazi. Automatic Construction of Diagnoser for Complex Discrete Event Systems. International workshop on Dependable Control of Discrete systems, Jun 2011, Saarbrucken, Germany. pp.112-1125, 2011. <hal-00676764>

[13] C. Mahulea, C. Seatzu, M.P. Cabasino, and M. Silva, “Fault Diagnosis of Discrete-Event Systems using Continuous Petri Nets,” IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, vol. 42, no. 4, pp. 970 - 984, July 2012.. DOI: 10.1109/TSMCA.2012.2183358

[14] M.P. Cabasino, A. Giua, C. Seatzu, "Diagnosability of discrete event systems using labeled Petri nets", IEEE Trans. on Automation Science and Engineering, Vol. 11, No. 1, pp. 144-153, Jan 2014.

[15] Pencolé Y (2004) Diagnosability analysis of distributed discrete event systems. European Conference on Artificial Intelligence. 2-3

[16] J. Kurien, X. Koutsoukos, and F. Zhao. Distributed diagnosis of networked, embedded systems. Technical report, DTIC Document, 2002.

[17] I. Hwang, S. Kim, Y. Kim, and C. E. Seah. A survey of fault detection, isolation, and reconfiguration methods. IEEE Transactions on Control Systems Technology, 18(3):636–653, May 2010.