EXPERIMENTS AND TECHNOLOGIES FOR DECENTRALIZED TEMPERATURE CONTROL

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Master of Science in the

Graduate School of The Ohio State University

By

Nicanor Quijano, B.S.

* * * * *

The Ohio State University

2002

Master's Examination Committee:

Doctor Kevin M. Passino, Adviser

Doctor Stephen Yurkovich

Approved by

Adviser

Department of Electrical Engineering

ABSTRACT

Centralized and decentralized controllers are developed here for a temperature control. First we show how to use the National Institute of Standards and Technology (NIST) Real-Time Control System (RCS) methodology in National Instruments' (NI) LabVIEW for a temperature control experiment consisting of 4 zones, each of which has a temperature sensor and heater. Then using the same concepts as before, but using dSPACE, we implemented different centralized and decentralized controllers using a grid consisting of 16 zones, each of which has a temperature sensor and heater. The physical layout of the temperature control grid experiment results in the heater from each zone affecting the temperature in some neighboring zones. Also, disturbances from ambient temperatures and wind currents have a significant impact. To illustrate the operation of the experiment, we developed resource allocation algorithms for both decentralized and centralized cases.

EXPERIMENTS AND TECHNOLOGIES FOR DECENTRALIZED TEMPERATURE CONTROL

By

Nicanor Quijano, M.S.

The Ohio State University, 2002

Doctor Kevin M. Passino, Adviser

Este trabajo está dedicado a mis padres, mi hermana, mi familia, mi asesor, y todos mis amigos tanto acá como en Colombia y el mundo, ya que me ayudaron y me apoyaron durante todo este período.

A ellos y a las personas que ya no me acompañan en mi vida les dedico este trabajo.

ACKNOWLEDGMENTS

I would like to extend my sincere thanks to my advisor Professor Kevin M. Passino for his guidance and support throughout my research. I would also like to offer my appreciation to Professor Yurkovich for being on my examination committee. Working with Professor Passino has been a fulfilling experience.

I am grateful to the National Institute of Standards and Technology (NIST) and my parents for giving me the financial support and opportunity to study in the United States.

I would like to thank Carlos, Lilia, Catalina, Andrea, my family, friends (in Colombia, around the world, and here) for always being there for me when I needed them, and always encouraging and supporting me in my studies.

Also I would like to thank Veysel Gazi, Alvaro Gil, Jorge Finke, Sriram Ganapathy, and Élie Abi-Akel, for their help during the development of the experiment, and their inputs in some of the chapters of this document.

VITA

May 7, 1974	Born - Bogotá Colombia
1997-1998	Electrical Engineering Intern, British Petroleum Exploration Colombia, Instrumentation Dept., Bogotá, Colombia
1996-1999	Teaching Assistant, Pontificia Universidad Javeriana, Bogotá Colombia
1999	B.S. Electronic Engineering, Pontificia Universidad Javeriana, Bogotá Colombia
1999-2000	Instructor, Pontificia Universidad Javeriana, Bogotá Colombia
2000	Traffic Engineer, Capitel-Telecom, Traffic Dept., Bogotá, Colombia
2001-present	Graduate Teaching and Research Associate, The Ohio State University.

FIELDS OF STUDY

Major Field: Electrical Engineering

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

While hierarchical and decentralized control are used extensively in industry, there are not a significant number of experiments in universities to support the training of engineers for the development and refinement of this technology. Here, we introduce an inexpensive experiment for decentralized temperature control and compare two software/hardware systems that support decentralized controller development and are generally available in a university engineer program (or at least which are often not cost-prohibitive). These are LabVIEW and dSPACE. In support of an educational methodology and as a general design approach for decentralized control we use National Institute of Standards and Technology (NIST) Real-Time Control System (RCS) methodology.

We implemented two different kind of experiments. First of all, using LabVIEW we develop an experiment consisting of 4 zones, each of which has a temperature sensor and heater. These zones are divided into pairs, and two NI Real-Time (RT) cards are connected, one to each pair of two zones. For this case we develop a hierarchy consisting of three different levels, where the higher one will be the supervisor, and the data will be transmitted using the TCP/IP protocol.

Based on the same RCS concepts, we developed another experiment, but now we used another data acquisition tool: dSPACE. The experiment consists of 16 individually controlled zones (each zone consists of 1 heater and 1 sensor) in a regular grid with lights for heaters and an analog sensor. The inputs and output of the 16 zones are interfaced to two different DS1104 dSPACE interface cards, and we will try to regulate the temperature to be uniform across the grid but with some fixed value, dividing the grid into four different "superzones" (each "superzone" has four heaters and four sensors). To show centralized and decentralized control, we developed some resource allocations algorithms. For that we use different controllers for each zone and a network with random but bounded delays for neighbor-zone sensed information, and delayed control inputs. To achieve decentralization we set up a "communication topology" that allows a "local controller" for each zone to sense the temperatures in neighboring zones (middle zones have more neighbors than edge zones). The goal is the same as in the centralized case, to achieve the highest possible uniform temperature. In both cases we have disturbances such as the ambient temperature, wind currents, and as we will see, inter-zone effects.

1.2 Literature Overview

Since most of the work was done using RCS methodology, the main book that was used is [1]. From there, we found all the information necessary to implement all the blocks in LabVIEW and dSPACEincluding the details on the pre-process, decision process, and post-process. Most of the these characteristics are embedded in the block diagrams and models developed in LabVIEW and dSPACE. As we will see in the chapters to follow, some of the RCS characteristics are quite difficult to develop in these two packages. Since the first experiment that we develop was the LabVIEW one, we used [2] to learn how to use the

basic functions of LabVIEW, and [3] clarified the internet interface using LabVIEW. This reference requires a lot of basic LabVIEW knowledge which we acquired from [2].

Decentralized temperature control is mostly studied in the semiconductor processing area. Chemical and electrical engineers develop decentralized controllers that lead to a uniform temperature on a plate for instance. In [4], the system is designed for chemically amplified photresists. The module is comprised of 49 individual heating zones which are controlled independently with separate temperature sensing, actuation and feedback control mechanisms. One supervisory control strategy is applied to coordinate each zone. Also it has an in-situ chill mechanism that is on the backside of the heating elements, which helps to prevent transport effects from the substrate to another chill plate, since during this transfer, the temperature can fluctuate and we can have a uncontrolled situation. The main goal of the design was to reduce nonuniformities during the transients. That is important since in the standard approaches, the stable temperature was achieved by having a tight spatial temperature uniformity across the plate surface. But the conventional approach to baking may be unable to meet such tight tolerances because of an inability to control transient deviations. Also we need a large amount of time to cool-down or heat-up the conventional plates to new set points. They used an a priori calibration to determine the appropriate plate temperatures to achieve precise photomask temperature uniformity.

Another important paper that was found was [5]. Here the experiment that was conducted involved decentralized control of wafer temperature for multizone rapid thermal processing systems. The system is configured as a in a pattern of multiple concentric lamps. The multiloop structure is achieved by taking the lamps paired individually to the sensors, and with a decentralized controller. The idea is to maintain a constant and repeatable steady-state condition, but in rapid thermal processing to achieve a zero-tracking error at steady state, we need a controller that does not make the closed-loop system unstable. The solution

that was implemented is to use a PID structure. First, they tune the proportional gain having the other two off, and after a while the integral action was added to eliminate offset, and the derivative was added to reduce overshoot.

Finally, a third related article developed in [6] describes a multizone space heating system, that maintains a desired temperature in different zones. Each zone has its own desired temperature, and the controller that is studied in this paper is a robust decentralized one. The experiment has two zones, that are controlled by 5 different controllers in a decentralized configuration.

1.3 Thesis Summary

We developed different kinds of temperature control experiments to study the behavior of several centralized and decentralized control strategies. Different technologies, such as LabVIEW and dSPACE were used. First of all we implemented an example to show how the concepts in RCS are applicable in LabVIEW. Then, based on the same methodology, but without using the same hierarchical structure, we developed centralized and decentralized experiments to show some differences between both technologies. In particular, we studied several "resource allocation" strategies for temperature control. The presence of the network, delays, and disturbances combine to make the temperature control experiment useful in both research and a graduate level laboratory.

CHAPTER 2

LabVIEW-RCS

Here we show how to use the National Institute of Standards and Technology (NIST) Real-Time Control System (RCS) methodology in National Instruments (NI) LabVIEW for a temperature control experiment consisting of 4 zones, each of which has a temperature sensor and heater. These zones are divided into pairs, and two NI Real-Time (RT) cards are connected, one to each pair of two zones. The hierarchical controller that is developed is divided into three levels, where the lowest level has two single-input single-output temperature control loops implemented in the NI RT cards. The second level has two computers each of which communicates with an RT card. The highest level is a computer that is connected to each of the middle-level computers and it serves as a "supervisory" controller and user-interface. The communications between each of the layers of the hierarchy is done via TCP/IP. This experiment shows one example of the use of the RCS methodology in LabVIEW and we show how to specify an RCS-style control module and RCS-style NML communications. The standard LabVIEW environment provides for an RCS-style "design tool" and a remote RCS-style "diagnostic tool" is implemented via the new LabVIEW internet capabilities. While we refer to the resulting software and methodology as "RCS-LabVIEW," we clearly have significant additional work to complete in order to fully realize such an objective. For instance we need to generalize beyond a single example and study

incorporation of other RCS concepts. We need to evaluate, develop, and implement other distributed control experiments and develop remote internet interfaces.

2.1 Introduction

2.1.1 Temperature Control Experiment Hardware and Challenges

The experiment that was developed is a simple temperature controller that allows for a hierarchical and decentralized control design and implementation. Our intent was to keep the experiment as simple as possible but still rich enough to demand the use of, and exercise RCS-LabVIEW concepts and software.

The hardware that was used for the plant is shown in Figure 2.1. It is a very simple hardware, since we need only four sensors, and four actuators. The values for the LabVIEW pins might be changed according to the channels that you selected in the "measurement and automation software" (MAX for short and for the rest of this document). For this specific case, we selected analog channels numbers 0 and 1, and according to the 68-PIN E Series these correspond to pins 68 and 33 respectively. For the digital outputs, we defined in MAX one digital port, and we used pins 52 and 17 to activate the actuators. If you are using another card, you have to use the proper NI pins. The actuators are the lamps, and those are turned on or off depending on the value that the base of each of the Darlington devices has (if the value is 0 then the lamp will be off, but if it is 5 volts, then the lamp is on). These Darlingtons are integrated in the DS2003 chip. We need to use these elements because the amount of current consumed by each of the lamps is 100 mA, and this current cannot be provided by the card. The sensors are National Semiconductors LM35CAZ temperature sensors, and these convert from degrees Celsius to volts. The data sheets are given at the web site `http://eewww.eng.ohio-state.edu/~passino/RCSweb/`.

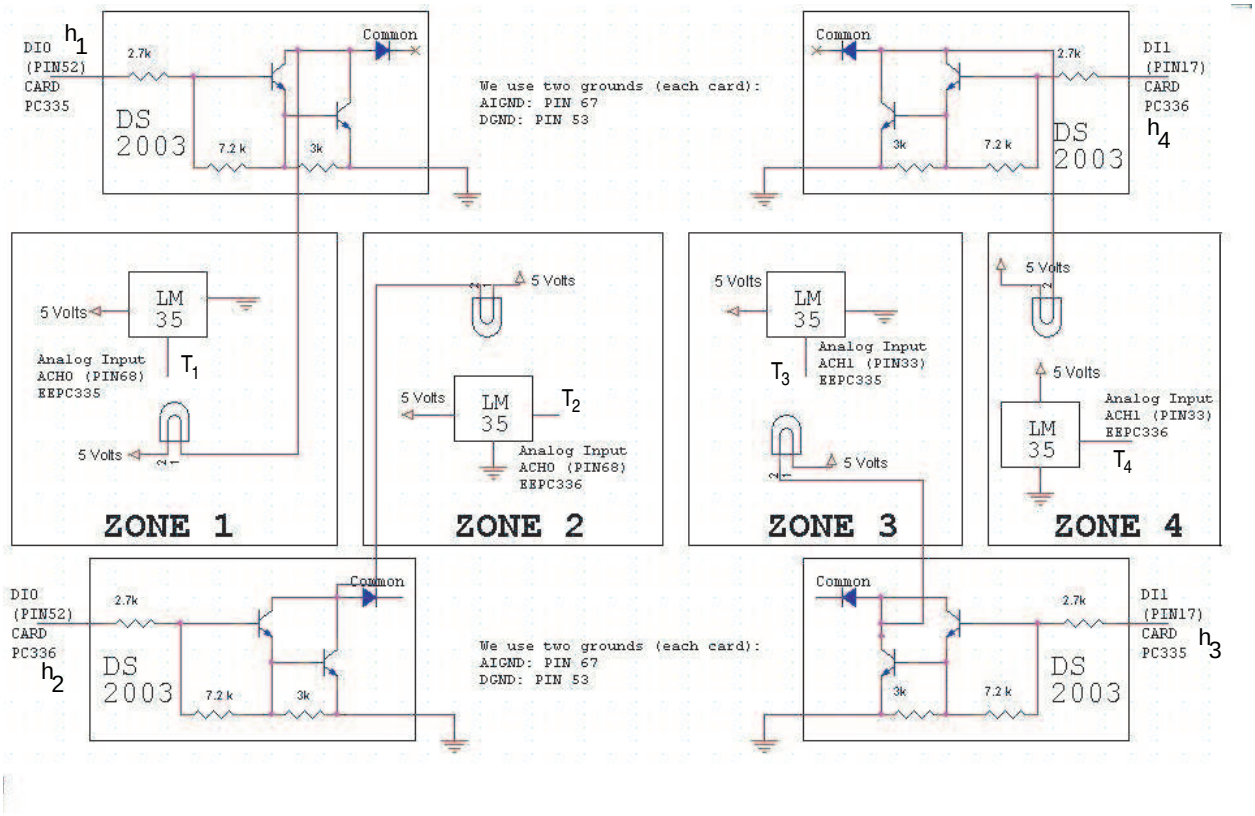

Figure 2.1: Hardware for the temperature control experiment.

The idea is then to try to control the sensed temperature T_i, $i = 1, 2, 3, 4$, for each of the zones described in Figure 2.1 using the control output h_i, $i = 1, 2, 3, 4$.

There are several possible control objectives:

1. Control each zone to a specified desired temperature T_i^d, $i = 1, 2, 3, 4$. The T_i^d could be constants and we could study a temperature regulation problem. Alternatively the T_i^d could be signals and we could study tracking. In both cases disturbances such as cool air currents are easy to induce and disturbance rejection capabilities of the compensator can be studied.

2. Control T_i of a zone i to track T_j of zone j, $j \neq i$ (i.e. let $T_i^d = T_j$, $i \neq j$). In this case T_i^d could be a constant or a time-varying signal. It is also possible to have sequences of zones tracking each other or a group of zones tracking the average temperature of another set of zones. Coupling such as in these cases over multiple zones naturally demands the use of hierarchical control.

2.1.2 Functional Architecture

Three levels in the hierarchy were implemented, where each of the levels has its own communication protocol, and the supervisor can take some control actions. Each of the levels follows the philosophy implemented by RCS, in the sense that each level has a pre-process, decision process, and a post-process. Since we were using NI RT cards, we needed to implement one protocol to communicate the lower level with the next layer of the hierarchy. For that, we used the standard TCP/IP protocol. For the middle level and the supervisor, we had two options for the communications: TCP/IP or the NI "DataSocket" connection.

The architecture that we used is shown in Figure 2.2. Since we had two computers (called EEPC335 and EEPC336 in Figure 2.2 and in the remainder of document), the supervisor is on one of these computers (EEPC335) because we do not have more than two LabVIEW 6*i* licenses. Each of these computers has their own NI RT card. This card is part of the lower level and to be able to manipulate the data from this level we need to make a TCP/IP connection between this level and a middle level that might control the experiment, and transfer the data to the supervisor. Each temperature controller in the lower level senses the temperature of two zones, and depending on the desired temperature value the actuators are turned on or off. The value that is sensed here is transmitted to the next levels using the TCP/IP protocol. The controllers that are running in this level are downloaded to the NI RT card, and they receive the information from the upper levels using also the TCP/IP

protocol. All the details of this level are going to be provided in Section 2.2. To make

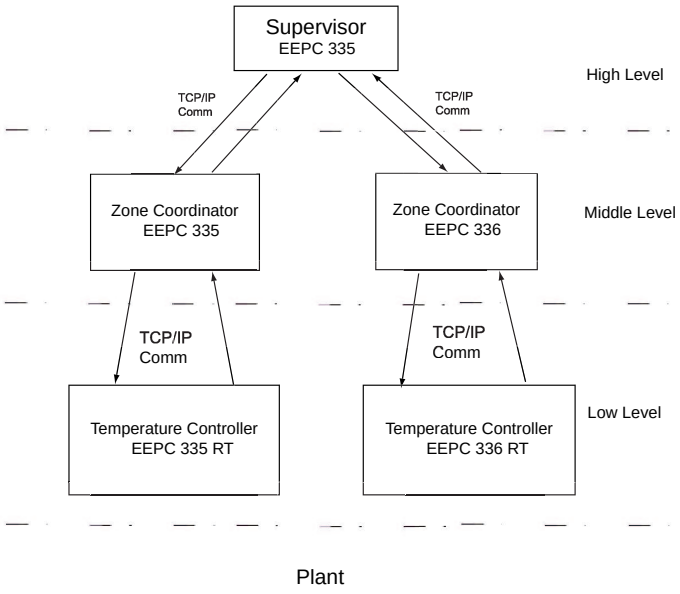

Figure 2.2: Architecture for the temperature control experiment.

everything consistent, both computers have the same LabVIEW code. The only difference is that in each of the computers we have to be aware of the URLs (Uniform Resource Locators, because it depends on which computer we are working on), and the port that we want to use to transfer the information (to avoid any "listening" conflict). More information on ports and URLs is given in Section 2.1.3. These ports and URLs are located in the upper level. First, we have a middle level where we coordinate two zones in each computer. This coordinator receives the information that comes from the lower level using the TCP/IP connection, and sends the data to the supervisor with the same protocol. The T_i and T_i^d for each zone are transmitted to and from this level. Then the supervisor reads the temperature values from both of the computers, and based on the information received and the user command inputs, it takes appropriate control actions.

2.1.3 Communications Overview

TCP stands for "transmission control protocol" and IP stands for "internet protocol" [1] [7, 8, 9]. TCP breaks the data that we want to transmit on one side of the transmission, and on the other side reassembles the information. Each of the broken data (sometimes called datagrams) is routed by the internet protocol.[2] One of the most important features is the IP address. This address (fixed in the computers used in the EE Department of The Ohio State University) is used to recognize the source/destination of the data that is being transmitted. Sometimes instead of using the IP address we use the Domain Name Service (DNS). In this case it is useful to use the DNS because is easier to remember the name of the computer where we are working. If the application is on the same computer, we can use "localhost" as the IP address for that computer. Another important feature in TCP/IP is that this protocol is connection-based. That is why we need to use an specific port on the remote machine. There are three kinds of ports: well-known, registered ports, and dynamic and/or private ports. The first ones go from 0 to 1023 and are used for specific services. The most common example is port 80 that is used by default by web servers to listen and return packets. Other examples can be port 21 for FTP connections, or port 23 for a TELNET connection. The registered ports go from 1024 to 49151 and those ports are the ones that we use to establish communication in LabVIEW because they can be used by specific applications. Finally, the dynamic and/or private ports go from 49152 to 65535 and are normally used for specific applications to avoid any conflict with the other ports (as in the case of TCP/IP) [3].

In order to discuss communications in LabVIEW first we need to introduce some basic elements in the NI language. For those who are not familiar with LabVIEW there are three

[1] For an introduction to TCP/IP see `http://www.yale.edu/pclt/COMM/TCPIP.HTM`

[2] For more information see the TCP tutorial by Charles L. Hedrick of Rutgers University at `http://oac3.hsc.uth.tmc.edu/staff/snewton/tcp-tutorial/sec2.html`

basic expressions that we need to know: front panel, block diagram, and virtual instrument. The term virtual instrument (VI) is the expression that LabVIEW uses for "programs." These instruments use a graphical programming language to develop the front panel, and the block diagram. The front panel of a virtual instrument is a set of controls and indicators. In this window, we can supervise and control whatever we programmed in the block diagram which visually resembles a computer program flowchart, and which corresponds to the lines of text found in text-based programming languages [2]. In LabVIEW we can implement standard TCP/IP communications. We can also use a "DataSocket" to transmit data. The DataSocket is a web programming technology that is designed to simplify data exchange between computers and applications and avoids complexity of low-level TCP programming. It converts the data for transfer so you can transmit different data types. LabVIEW Real-Time 6*i* supports the DataSocket read and DataSocket write. The DataSocket server runs on the host computer, while the read and write can be on the host or the embedded controller. In real-time applications however, you cannot use front panel binding for your DataSocket communications. To associate a DataSocket to any variable we have two options: we can perform a right-click in the front panel under the instrument that we want to associate and we give an specific address called `dstp`. Another way to read and write data using a DataSocket is in the block diagram. Here we use the programs that LabVIEW has to transmit the information. In both cases the most important element is the `dstp` address that we associate with each of the elements that we want to transmit and/or receive. In our case, this method seemed to be one good option to transmit and/or receive the data from one computer to another. The problem that we found is that since we already have a TCP/IP connection implemented in the lower level (because we need that to transmit the data that comes from the RT card and we cannot use DataSocket to share this information) when we used several DataSocket connections at the upper levels we had a significantly larger delay

than the one that we have right now. The delay when we used the DataSocket approach was so significant that, for example, displays on the front panel were largely out of sync with the flashing of the lights as they turned on and off by the controllers.

2.1.4 Design and Diagnostic Tools

The RCS diagnostic tool is a Java based graphical program that an operator can use to connect to a running RCS application from a (possibly) remote host. Any web browser that supports Java (e.g., Netscape or Internet Explorer) can view the diagnostic tool. It also can be viewed as a stand-alone application provided that the Java Development Kit (JDK) for the given platform is installed in the system. The main objective of the diagnostic tool is to allow the operator to read state values of the plant and send commands to controllers using the diagnostic interface. In other words, it provides a high-level interface to the control routines. Before sending the command messages, plotting the fields of the status messages, or viewing the current state of execution (in the state table), the operator can modify (or set) the fields of command messages. The architecture file (which is a text file that provides the controller structure, the command status, and TCP communication port numbers for accessing the auxiliary buffers) allows the RCS diagnostic tool to read the information about the structure of an RCS controller. These architecture files can be generated automatically by the RCS design tool [1]. The RCS design tool is a Java based graphical program that the designer can use to easily layout the modules of his/her controller structure (i.e., "design" or build an RCS application) visually and generate automatically all of the application independent code. This includes the code for the command and status messages to be passed between the modules, the code for the RCS modules, the scripts needed by the RCS diagnostic tool, and all the scripts that are needed for compiling and running the application. The user can

add or remove modules modify the hierarchy, and set up communication channels between modules using this design tool [1].

In LabVIEW these tools (design and diagnostic) can be viewed as the front panel and the block diagram. Once you define in your block diagram the elements necessary to control your plant, you can add modules (in this case we will call these as VIs), set up communication channels, or anything related to the graphical interface that we have to develop any program in LabVIEW. For us, this is transparent, and we do not need to modify the code that was generated once we run the application. Meanwhile, in the front panel, the operator can change the port number that we use to communicate to each level, he/she can add graphical tools (such as plotters) to see what is happening in the lower levels if he/she wants. Besides, the operator can change the elements necessary to define different control algorithms, and can see what is the status of the message that we are sending.

2.2 Lower Level

In this section we explain how to build an RCS-LabVIEW controller implemented for the lower level. In this level, we use the same philosophy of RCS described in [1]. The block diagram and the front panel for the lower level are shown in Figures 2.3 and 2.4, respectively. As we can see, this level consists of two main VIs: One called "Temp" that will be in charge of the acquisition of data, and another called "Comm" (which stands for communications) that will transfer the information from one layer to another, using the TCP/IP protocol. This VI has only one element that we might want to change in its front panel, as we can see in Figure 2.4. The value of the port for this case is 2055, and this value will be the one used to send and receive information to and from the NI RT card. The value of this port is related directly to the VI "Comm" that we will explain later.

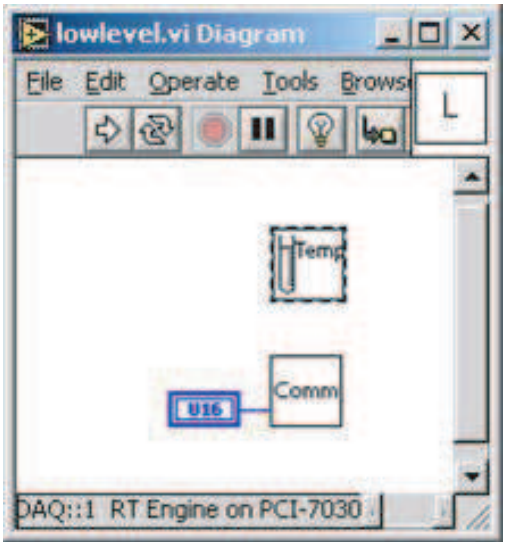

Figure 2.3: Block diagram for the lower level.

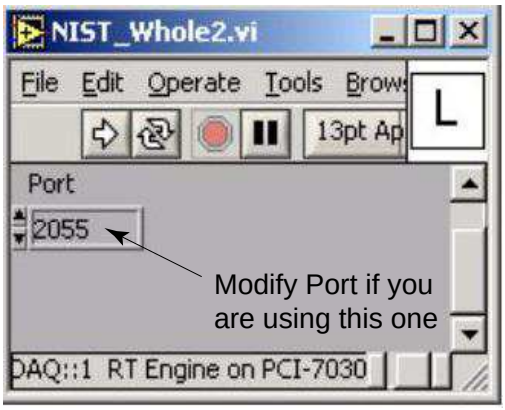

Figure 2.4: Front panel for the lower level.

2.2.1 Data Acquisition

The block diagram for this part is shown in Figure 2.5. We have three different blocks in the while loop called Pre-Proc, Dec Proc, and Post-Proc, because the functions that they have are similar to the methodology applied by RCS. The initial conditions are outside

the loop because we need to initialize some elements before we start the main loop (e.g., the shift register). As we know [1] the pre-process is normally used for retrieving inputs and performing simple conversions or any needed filtering on the data needed every cycle and often by multiple commands, the decision process usually calls one of the command functions based on the current command type, and the post-process is normally used for writing outputs and performing simple conversions on the outputs needed every cycle by multiple commands [1]. The other elements in this block diagram are a stop button that cancels the operations whenever the user wants, and a clock that allows us to control the loop operation (notice the timer functionality here is analogous to the RCS-TIMER and that we can set different sample times for each module based on its timing requiremets.) In this case, we tested the experiment with an iteration time of 100 ms. The front panel for this system is shown in Figure 2.6. This front panel was designed to analyze the information that comes from the card. The boolean elements represent the lamps, and the thermometers indicate the temperature values acquired by the card. The graphs will show the temperature plotted vs. time and in the form of an "intensity graph." The intensity graph is an element where the value that comes from the sensed temperature is scaled, such that it has a color that corresponds to the scaled value.

Now that we understand the basic elements in the data acquisition program, we need to explore in detail each of the VIs. The initial conditions are values that should be defined at the beginning of the main loop (the block diagram is shown in Figure 2.7). Some of these values might change during the process, but there are others that do not. To give some examples, in Figure 2.8 we can see that we define the number of the device, and the name of the digital channel. In this case, we defined in MAX a digital port that will be a digital port to be written called LAMP. The device that we need to use is # 2 for EEPC336 and #

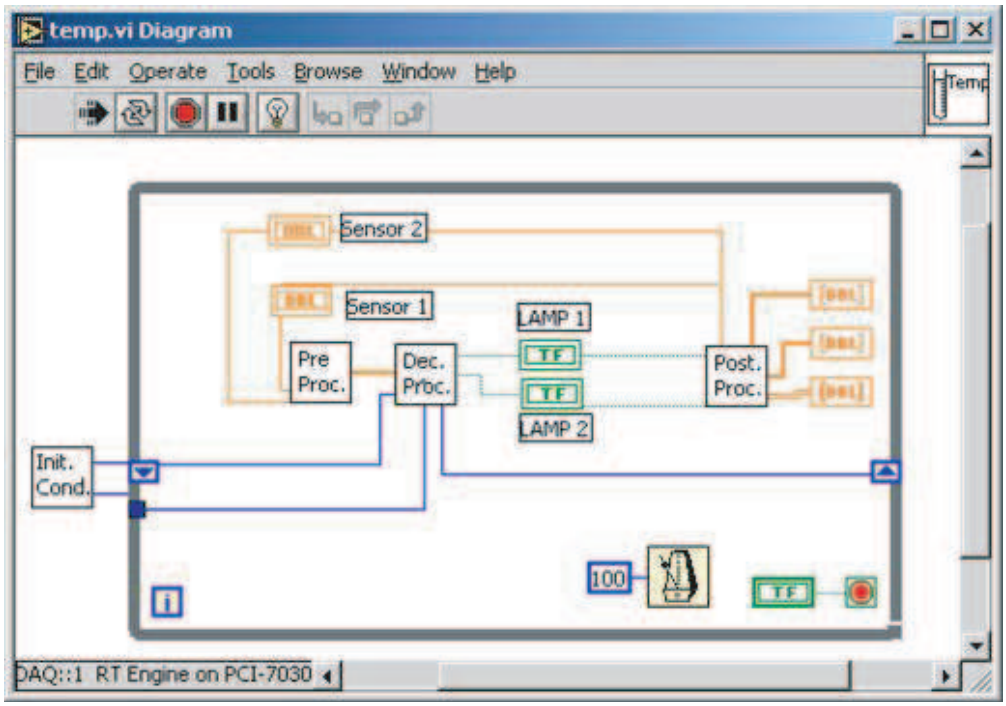

Figure 2.5: Block diagram to acquire the data and process it.

3 for EEPC335. These elements can be changed according to the configuration defined by the user.

For the pre-process block in Figure 2.9, since we wanted to acquire the data that come from two different sensors, and we wanted to retrieve these data at the same time, we decided to use the Multiple Channel, Single-Point Analog Input block as we see in Figure 2.10. The user has to define the channels and the number of the device, since these might change according the application. For instance, in our case, we need to define device # 2 for the card that is used in EEPC336, but in EEPC335 we need # 3. As we can see, in MAX we defined AI0 and AI1 as Sensor0 and Sensor1 respectively. To avoid any noise due to the line or the way that we twisted these cables, we decided to use the averaging VI. For this case,

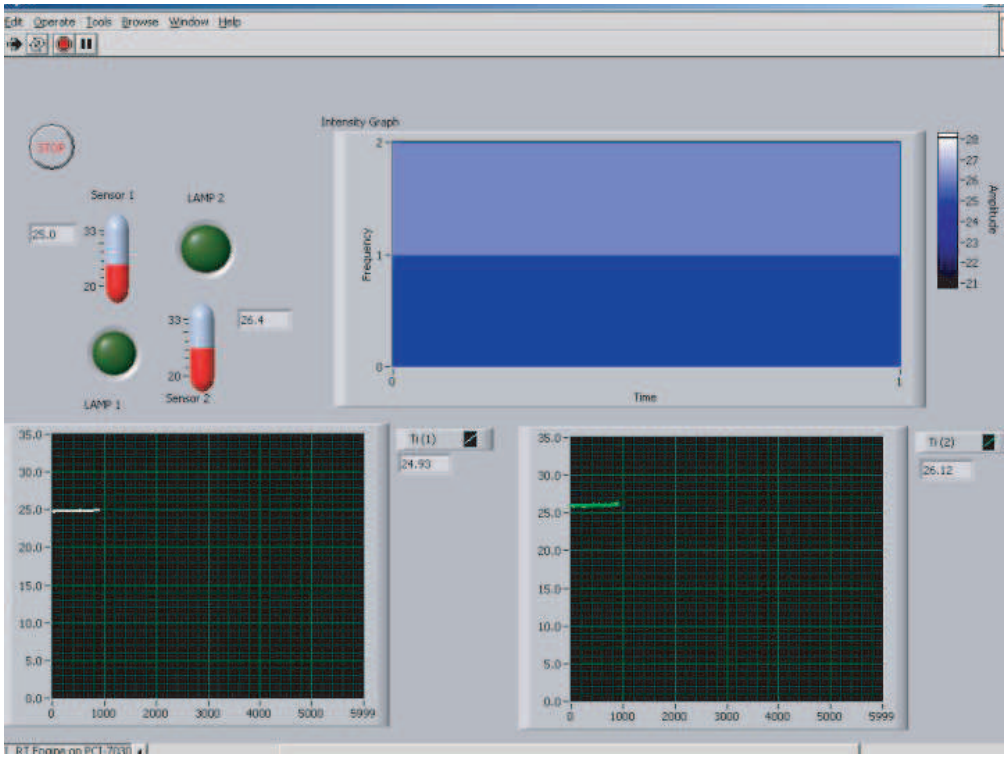

Figure 2.6: Front panel to acquire the data and process it.

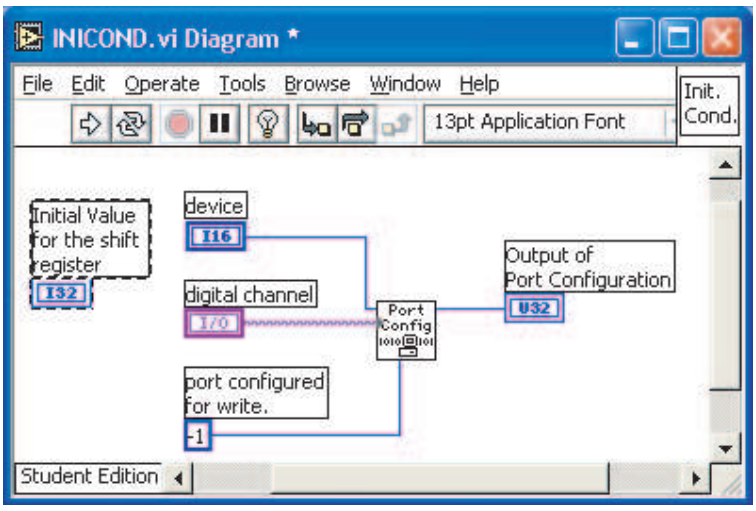

Figure 2.7: Block diagram for the initial conditions.

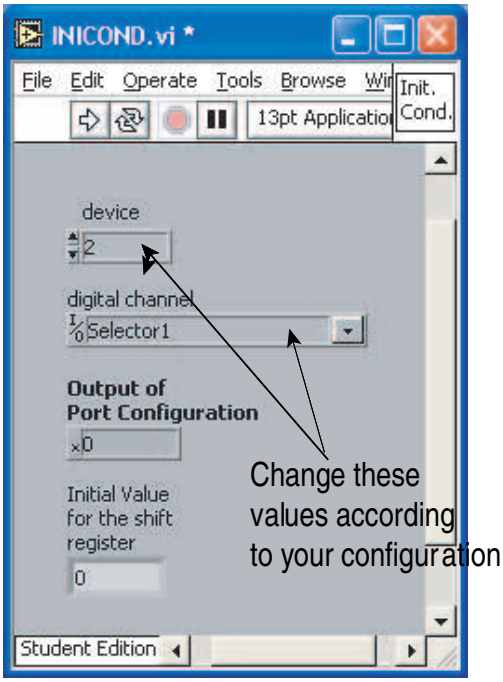

Figure 2.8: Front panel for the initial conditions.

the outputs of this VI are the values of sensors 1 and 2, and one array that has the value of both, and would be useful in the next VIs.

In the decision process we need to implement the controller. In this case, we implemented a simple on/off controller. As we can see in Figure 2.11 we take the value of a global variable called SET POINT and we subtract that value from the one sensed. This value comes in an array from the pre-process, and the output of the subtraction gives a value that could be positive or negative. If the value is positive, the value will be a 1, otherwise it will be 0. This value goes to a comparator where we see whether the value that we got is positive, and if it is we need to update the digital channel. For that we used the VI called "Port Write" that writes to a port if a switch has changed state. The decision that is made goes directly to the port defined in the initial conditions (Figure 2.8). To avoid delays when we update the digital port, we store the value that we changed in the digital channel in a shift register.

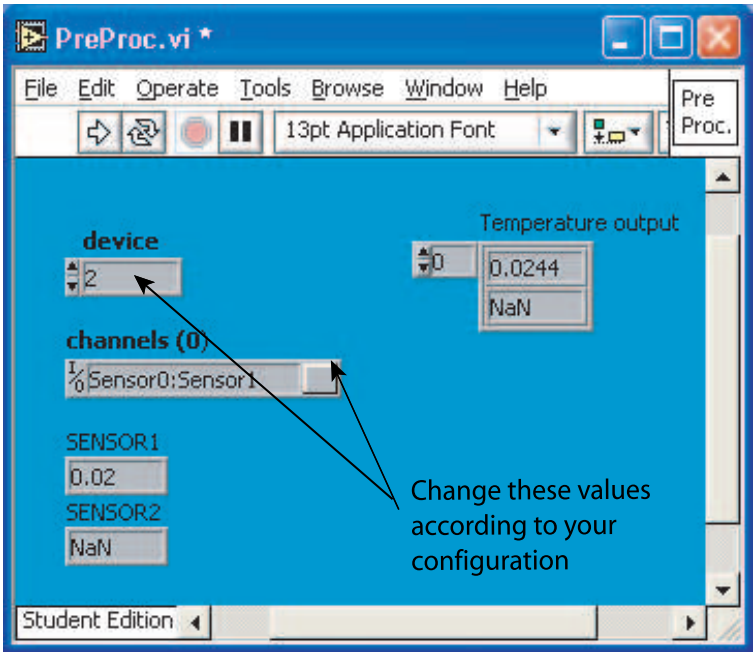

Figure 2.9: Front panel for the pre-process.

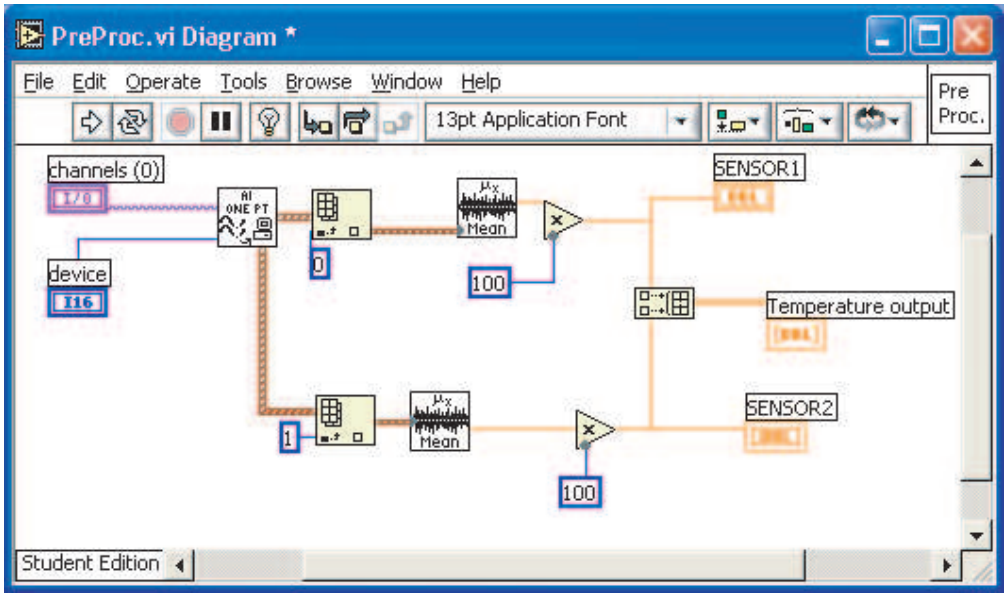

Figure 2.10: Block diagram for the pre-process.

Next time, the value that is stored in the shift register will be compared with the new one, and if it is different, the port will change, otherwise it remains equal. The front panel for the decision process is shown in Figure 2.12. To use the decision process in Figure 2.11 we implemented a subVI where the outputs are the state for the two lamps and the input is the array that comes out from the comparison. This subVI allows us to turn on/off the lamps, and is called "Array Vector."

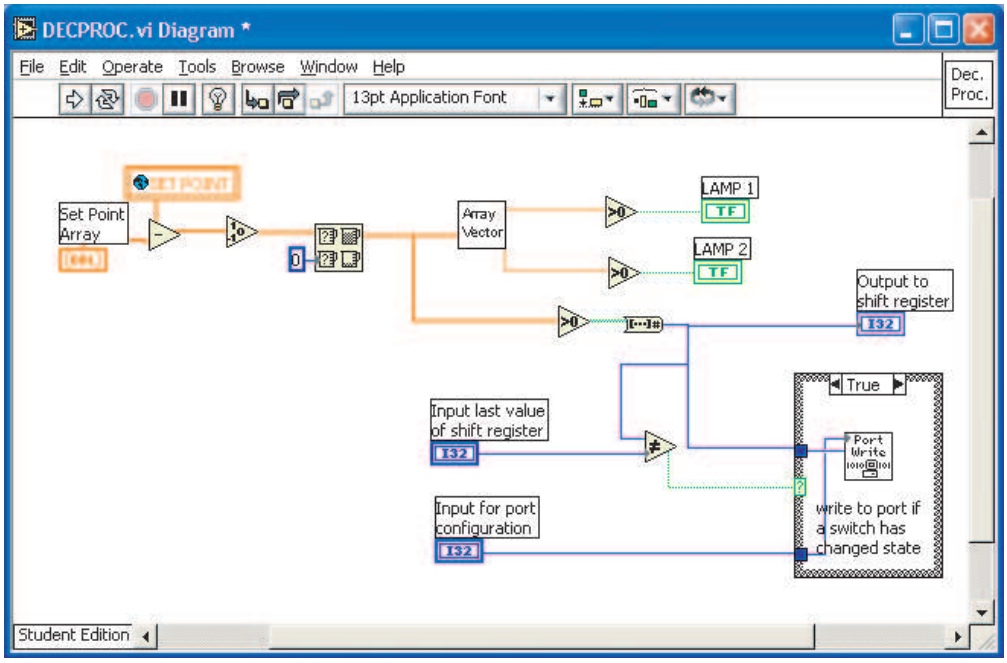

Figure 2.11: Block diagram for the decision process.

Next consider the post-process of Figure 2.13. Here, since we needed to perform simple conversions and write some outputs [1], we decided to store in a global variable called "Cluster" the information that comes from the sensors and the status of the lamps. For those who are not familiar with LabVIEW, a cluster is a fixed collection of data elements of mixed types [2]. In this case, as we can see we will send the information of the temperature

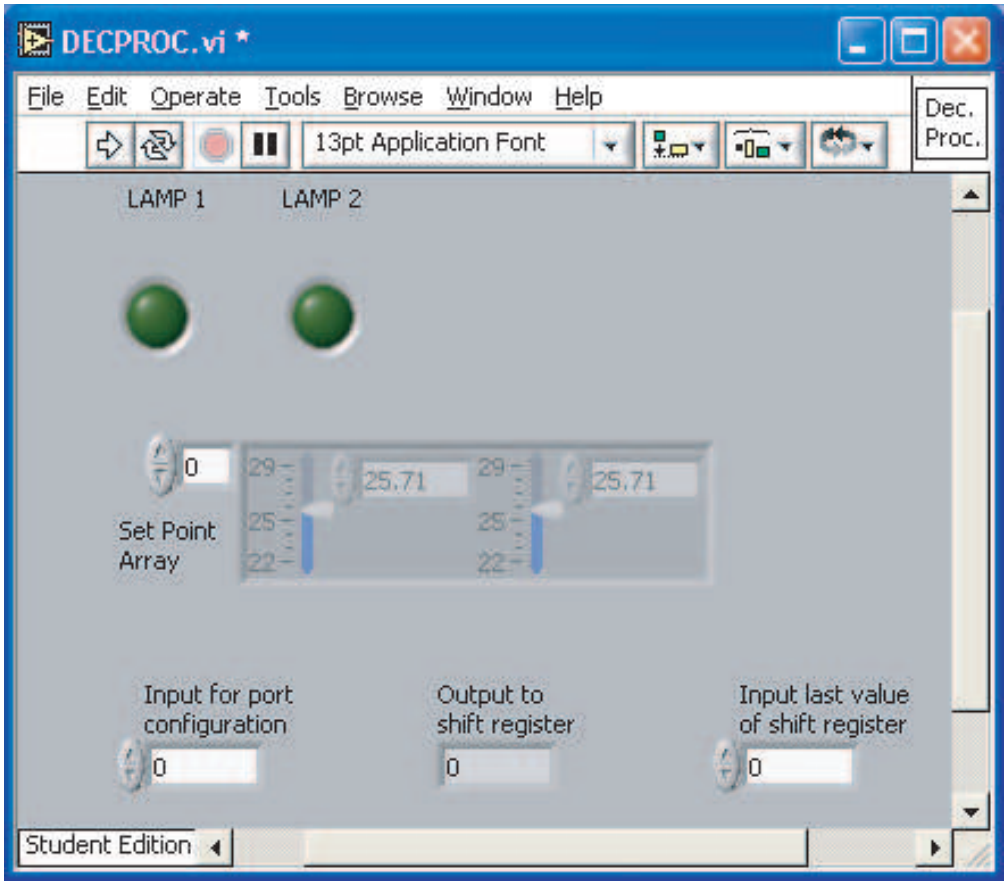

Figure 2.12: Front panel for the decision process.

sensed and the status of the lamps using the VI called "Comm" such that the TCP/IP can transmit the information to the next level. Also in Figure 2.14 we see that there are two charts and one intensity graph. These three elements are here to observe and analyze the data in the lower level, if we define different set points in the global variable SET POINT defined previously. Note that there is a connection between the cluster and RCS status messages. Similar to RCS messages it can have data fields of different types, and the SET POINT correspond to what is called a command message in RCS.

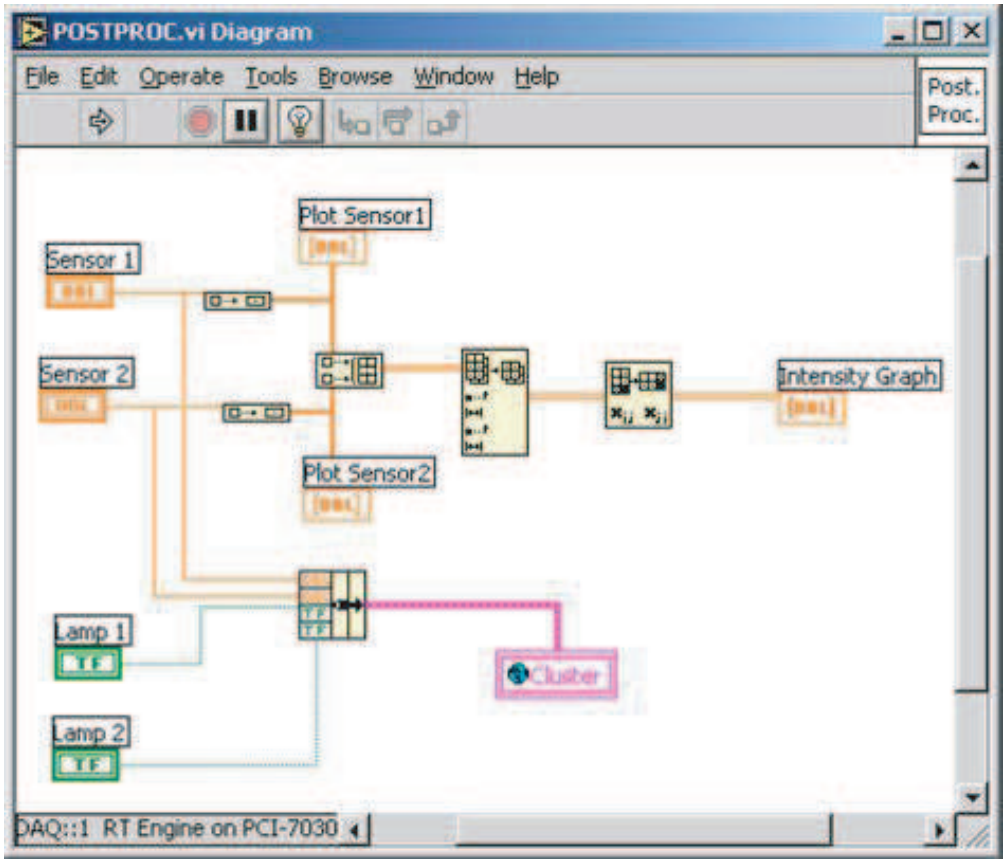

Figure 2.13: Block diagram for the post-process.

2.2.2 Communications

The NI RT card cannot run more than one program. Therefore, we need to find a way to transmit the information that we had acquired, so it can be manipulated and visualized by the higher levels. In LabVIEW there are different methods to transmit data, but some of them (e.g., DataSocket) does not work using NI RT cards in the LabVIEW version that we have (6.0.3). That was the main reason to choose the TCP/IP protocol to send the data from one level to the next one.

To develop the block diagram that you find in Figure 2.15, we used the structure of a single data server. We modified the original VI that comes in the LabVIEW examples by

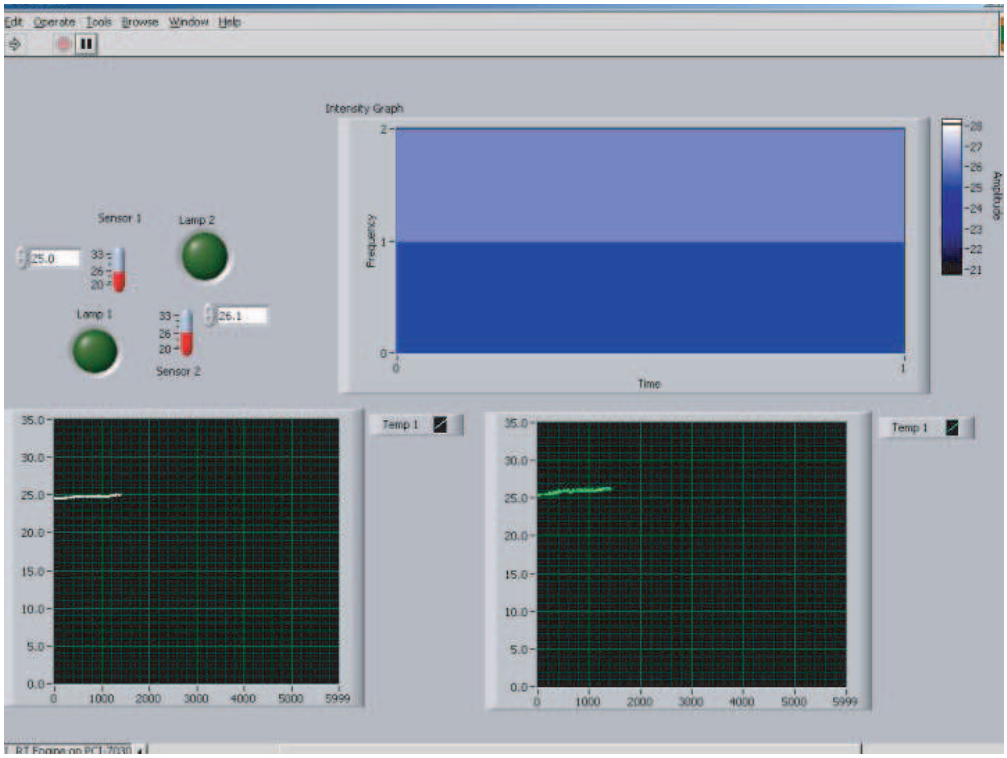

Figure 2.14: Front panel for the post-process.

adding some elements that allow it to read the information that comes from the middle level. There are two main loops. The outer while loop listens for incoming client connections, until an error occurs, and the inner while loop sends data to a client as long as it is connected or an error occurs [3]. The outer loop has three important VIs: TCP listen, TCP Close connection and the status element that allows it to stop the VI when an error is detected. The main block here is TCP listen. This element listens to whatever you have on the port specified by the user. In this case, as we can see in Figure 2.16, we used port # 2055 to establish communication between the lower level and the middle one. The user should know that if he/she wants to add another listener, the port of the new one must be different from the one used before. As we saw in Section 2.1.3, there are different ports, but the most useful ones for this kind of application are the ones that are in the range of 1024 to 49151, because

those numbers correspond to registered ports, and may be used by specific applications. This outer loop keeps lower level communication running continuously because we did not specify any timeout. Another VI that is basic in this configuration is "NO EOC Error.vi." This VI is here to allow the outer loop to detect when the client finishes the connection because we want to reset the server to continue listening [3].

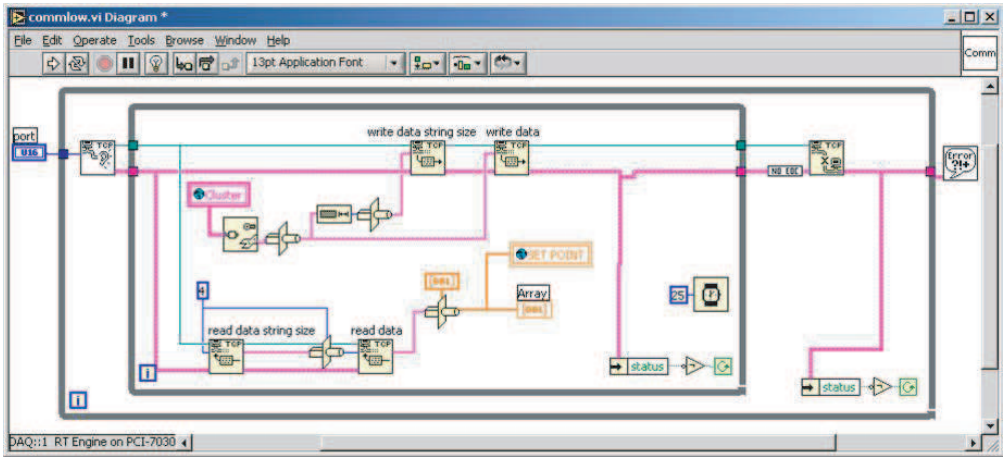

Figure 2.15: Block diagram for the transmission of data.

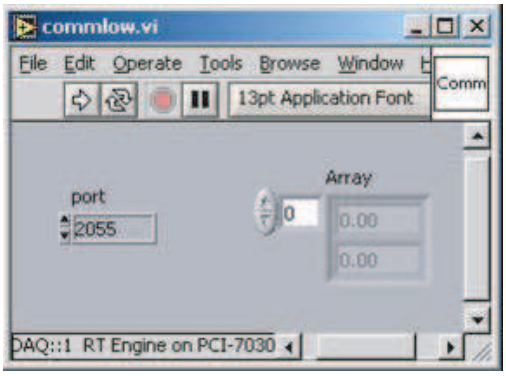

Figure 2.16: Front panel for the transmission of data.

As we can see, inside the inner loop we have two main steps. The first one, in the top part of the inner loop of the block diagram shown in Figure 2.15, is the one that allows the user to write the data from one level to the other. We use two VIs called "TCP Write" that acquire the information on the number of bits that are going to be transmitted and the information that we will send. In this case, we want to transmit the global variable "cluster" defined in the post-process. Since the VI "TCP Write" accepts strings, we need to convert this cluster into a "flattened string" (this conversion allows it to transmit the cluster in one datagram as a string). That is why we used the VI called "Variant to Flattened String" that converts the cluster that we formed in Figure 2.14 into a string. Once the variable is converted, it passes directly through the VI called "Type Cast" that again converts the data that comes from the flattened string, so that we can transmit the data to the client. The second one, in the bottom part, of the block diagram, is where the data is read from the middle level. Here we are reading the desired temperature T_i^d that would be stored in a global variable called "SET POINT." The idea is practically the same as in the previous case, but instead of converting the data into strings, we need to convert the data into an array. That is why the "Type Cast" VI has a dummy variable that says that the data that is coming is an array. This VI will behave as a server, since it is always listening to the port specified by the user. The middle level will have a similar VI, but instead of listening all the time, it will behave as a client.

This communication scheme has some similarities with NML in RCS. As we mentioned before, the clusters are elements that can store any type of variable, such as booleans, integers, doubles, or even strings. These clusters are similar to the messages in NML, since the NML provides a mechanism for handling multiple types of messages in the same buffer [1]. Another similarity that we can see is the encoding and decoding of the message. In RCS the NML server can encode and decode messages on behalf of remote processes [1]. Here, the

encoding part is done using the subVIs that we studied before to transmit the information via TCP/IP, and the decoding depends on the type of message that we will send. The problem, as we will see in Section 2.3.1 is that in the decoding part, the size of the cluster is defined at the beginning and cannot be changed dynamically. Therefore, the decoding part is more complex, and as we mentioned before, depends on the message that we want to transmit.

2.3 Higher Levels

2.3.1 Middle Layer

As we mentioned before, the main objective of the middle level is to acquire the data that comes from the lower level using TCP/IP connections, share this information with the supervisor, and process commands from the supervisor. In this level, the user does not have to interact with any data acquisition device. As we can see in the front panel of this middle layer in Figure 2.17, the user will be able to see the data that comes from the sensor T_i, and the status of the lamps, and he/she can change the desired temperature T_i^d of the system. Also, the user can select the type of controller that he/she wants by clicking on the menu that is in the middle of the Figure 2.17. In this case, we have only two kinds of controllers that we will explain later. The button called "OK Button" is the one that allows the user to control the experiment from this level, or if it is ON (by default it is OFF, and the user can recognize that because the button is red) the supervisor will be the one that controls the experiment. The two plotters that we have here allow the user to observe the measured temperatures T_i that comes from the lower level. Since we are following RCS methodology, the middle level is implemented as four different elements that perform the same functions described in Section 2.2. As we can see in the block diagram shown in Figure 2.18 the VI is made by the initial conditions, the pre-process, the decision process, and the post-process,

all of them inside the main loop (the experiment will stop when an error occurs). Now we will explain each of these subVIs.

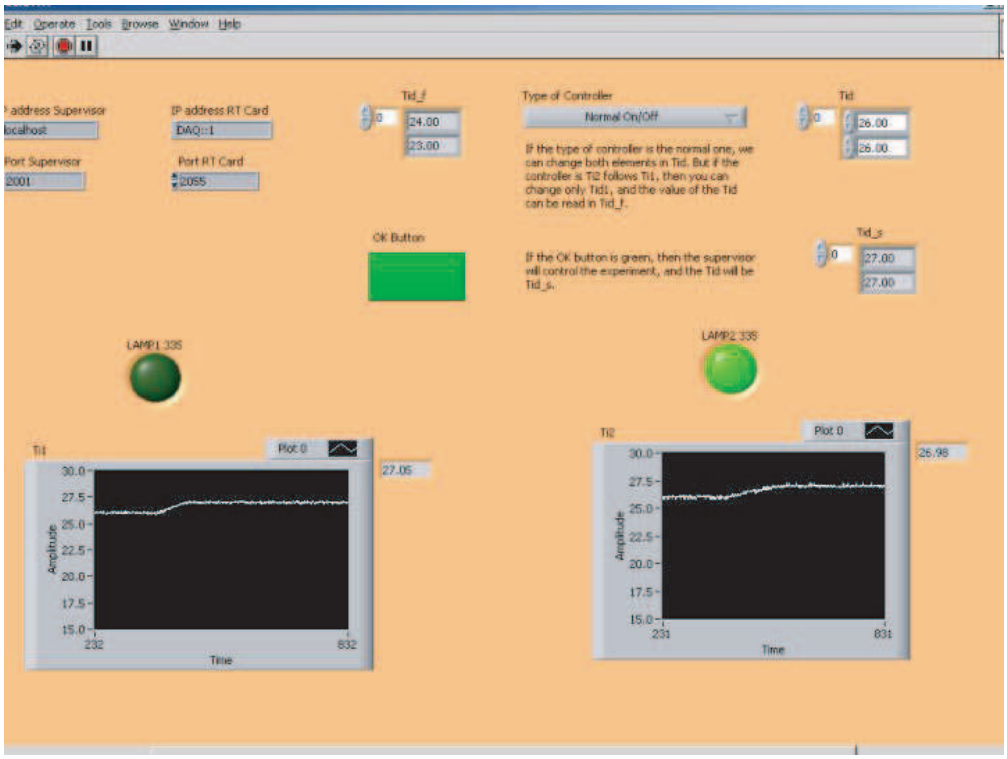

Figure 2.17: Front panel for the middle level.

For initial condition we need to initialize the IP addresses and the ports that the user will use to transmit and receive the data from the lower level, and the supervisor. The block diagram for this case is shown in Figure 2.19. If the user wants, he/she can change the IP address and the ports directly from the front panel as we can see in Figure 2.20. The outputs for this subVI are the connection IDs and the errors that can be generated, and should be listened to each of the elements that belong to the TCP/IP communication. As we know, the port address must be the same as the one that we used in the lower level. That is why we used port # 2055 again. The IP address that we have to use in this case is

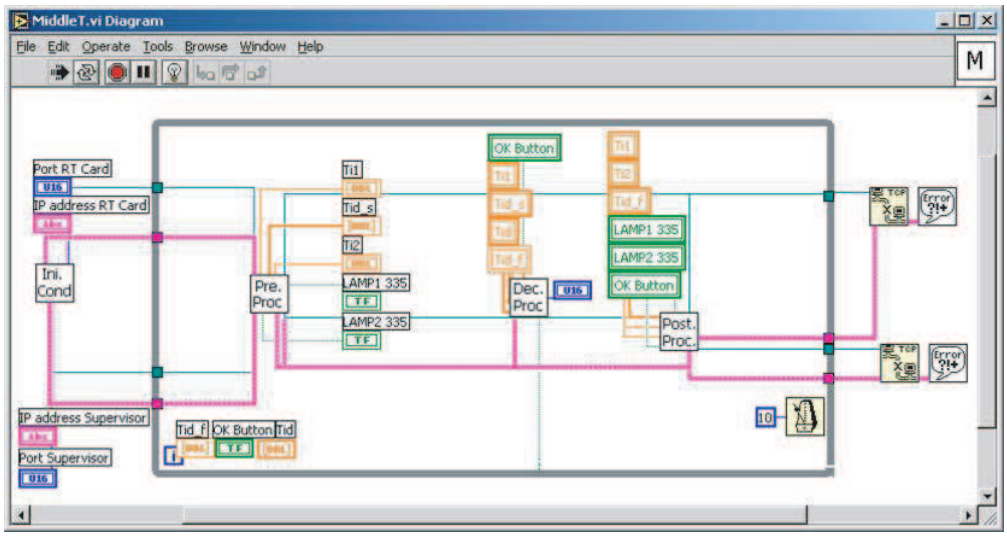

Figure 2.18: Block diagram for the middle level.

DAQ::1 because that is the address that the card uses to transmit information via TCP/IP. Here we need to specify another port and another IP address, since we are going to transmit the information to the supervisor. For that, we selected port # 2001 (if we are working on computer EEPC335, or # 2000 if we are working on EEPC336), and the IP address is "localhost" if we are working on the same computer that will behave as a supervisor. In the other case, we have to know what the IP address is, since as we mentioned in Section 2.1.3 the IP address in the EE Department at OSU are fixed. For this example, the EEPC336 has an IP address equal to "164.107.163.84."

The pre-process in this case listens to the information that comes from the low level and from the supervisor. This subVI will acquire the information of the sensed temperature ("T_i EEPC33Y"), the status of each of the lamps ("LAMPX EEPC33Y"), and will have the desired temperature ("SP Sent") that the user sends when he/she is working with the supervisor as we can see in Figures 2.21 and 2.22. To read the data that comes from the lower level we need to read how many bytes come, and after that we read the data. Once we have

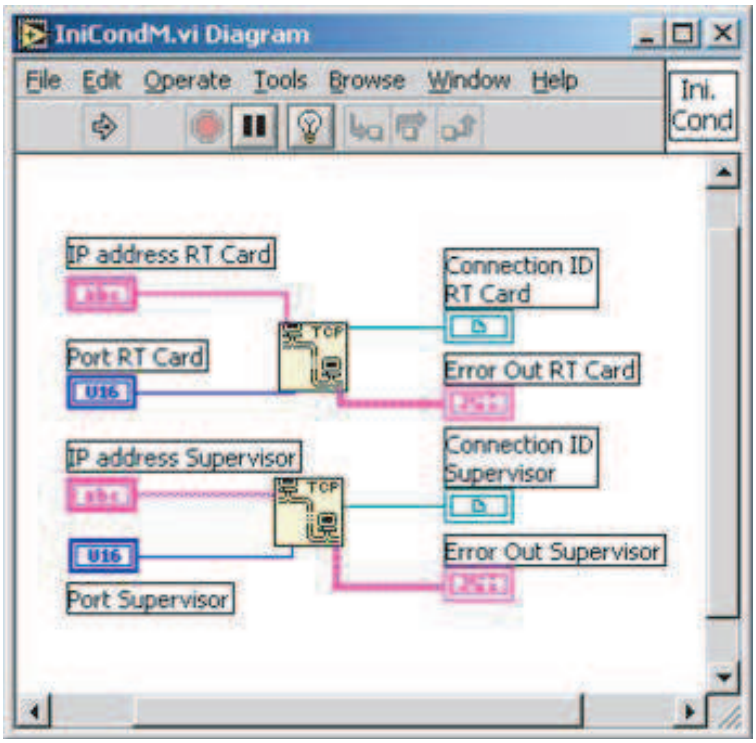

Figure 2.19: Block diagram for the initial conditions of the middle level.

the data (usually it comes in a string form), we need to convert this string into a cluster (since we know that that was the original format used in the lower level to transmit the information.) For that, we need to know the number of elements that we have transmitted, because the VI "Flattened string to variant" needs an index that corresponds to the data that has been transmitted. To know what value we must put here, we developed an extra VI (Figure 2.23) that shows the index array according to the order of the elements that we put in the cluster. Once we convert the data into a cluster, we can extract the four elements that come from the lower level, and work with these values in the other subVIs. On the other hand, we have the data that comes from the supervisor. Since in this case we are transmitting only a simple array, we do not need to convert the data using the same elements as before, but we need to use a dummy variable called "dummy" in Figure 2.21

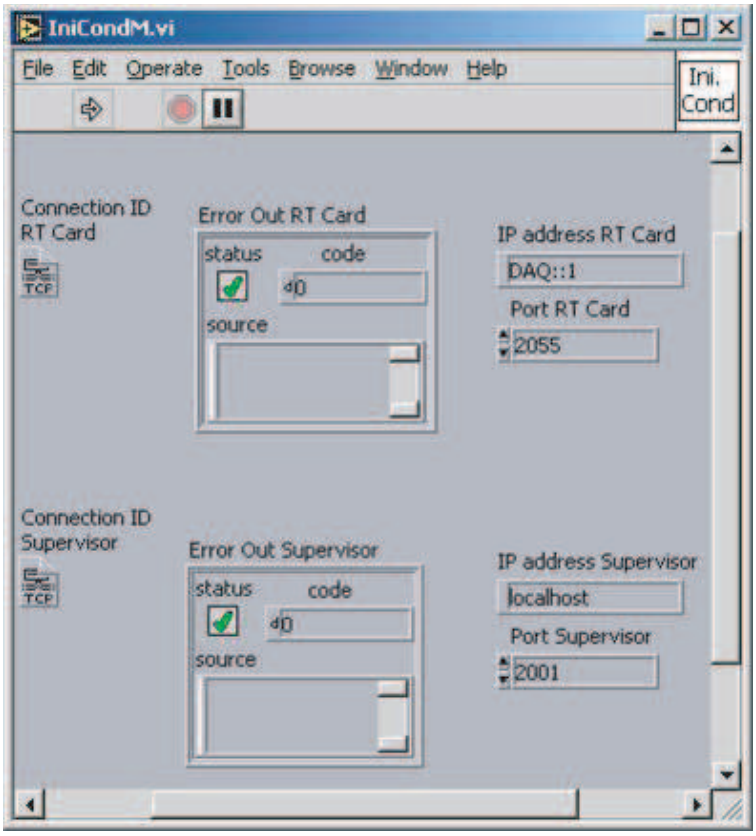

Figure 2.20: Front panel for the initial conditions of the middle level.

that will say the program that we are receiving is an array. This array called "SP Sent" is the desired temperature value that the user is sending from the supervisor level. If the "OK Button" is red, this value does not have any meaning in the control algorithm.

Once the pre-process acquires the data necessary to work in this level, we have to make some decisions. For that we use the VI called "Dec Proc" in Figure 2.18. As we can see in Figures 2.24 and 2.25 this VI receives the information that comes from the pre-process, such as the desired temperature from the supervisor "SP Sent," and other desired temperatures that we will call "SP Local 2" and "SP Local." In the front panel shown in Figure 2.18 we called these variables T_{is}^d, T_{if}^d, and T_i^d respectively. The variable T_{is}^d is the desired temperature when the supervisor is running the experiment. The variable T_{if}^d is the desired temperature

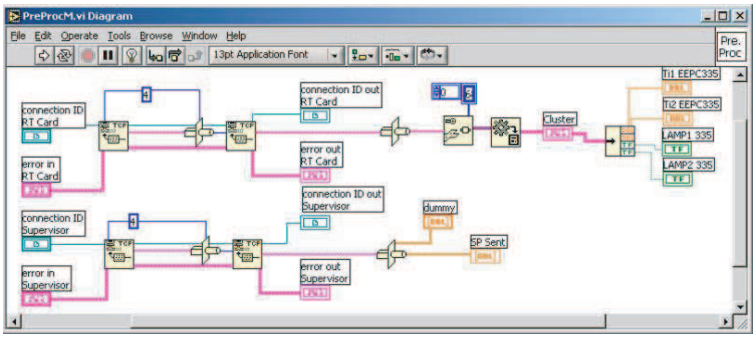

Figure 2.21: Block diagram for the pre-process of the middle level.

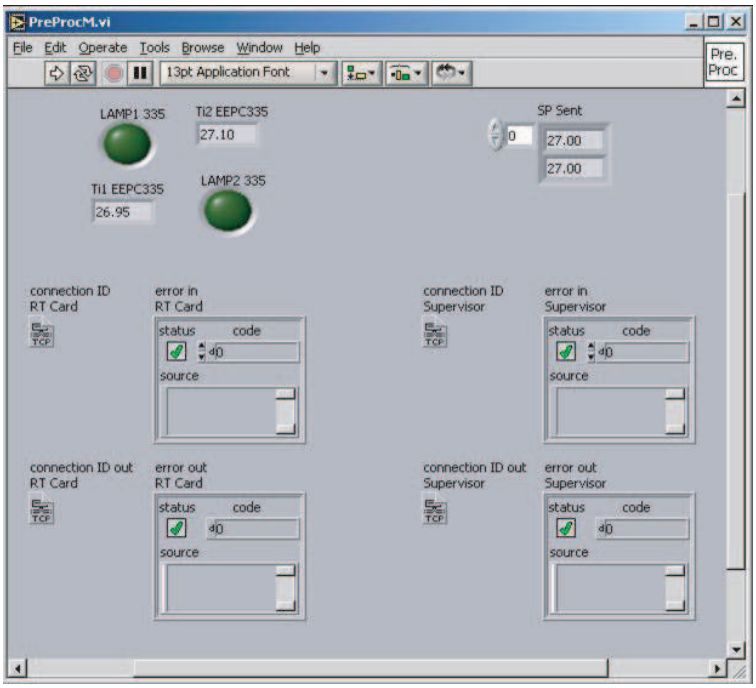

Figure 2.22: Front panel for the pre-process of the middle level.

when the control strategy that we are using in the middle level forces T_2 to follow T_1. Finally, T_i^d is the desired temperature value that we use to change T_1^d and T_2^d if we are using an on/off strategy, or to change T_1^d only if we are using the control strategy when T_2 follows T_1. Besides

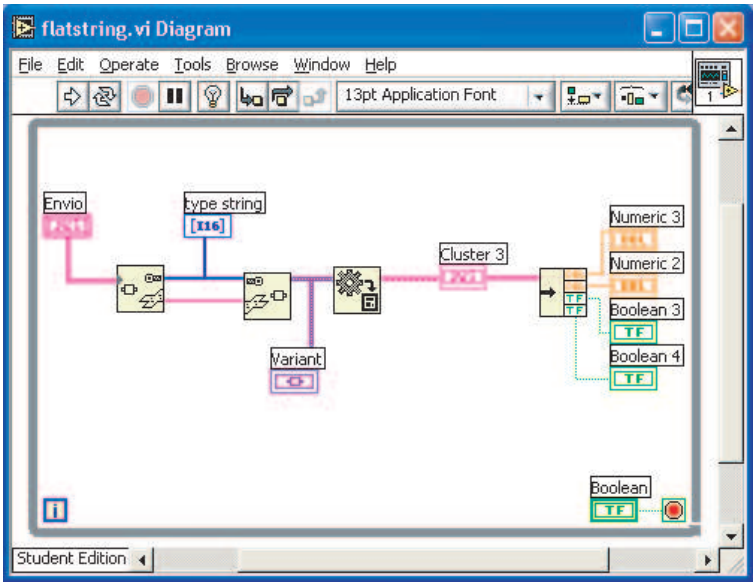

Figure 2.23: Block diagram to know what kind of index we must put according to the data that was transmitted.

these desired values the decision process receives the type of controller that we will use, and the status of the "OK Button."

As was mentioned before, we implemented two different kinds of controllers. The first one is the classical on-off controller that will use as a set point the value that comes in T_i^d. The other controller is used when the temperature of the second zone follows the temperature that we have in the first one and, as we mentioned before, the desired temperature that we will send to the lower level will be the one that comes in the T_{if}^d. For that we implemented a "Case" structure inside an if-else structure where the value of the desired temperature is changed. As you can see in Figure 2.24 the value that comes out from the if-else structure will be stored in the global variable "SET POINT" described in Section 2.2. Note that this is similar to the if-then-else structure which is implemented in the decision process of the RCS library.

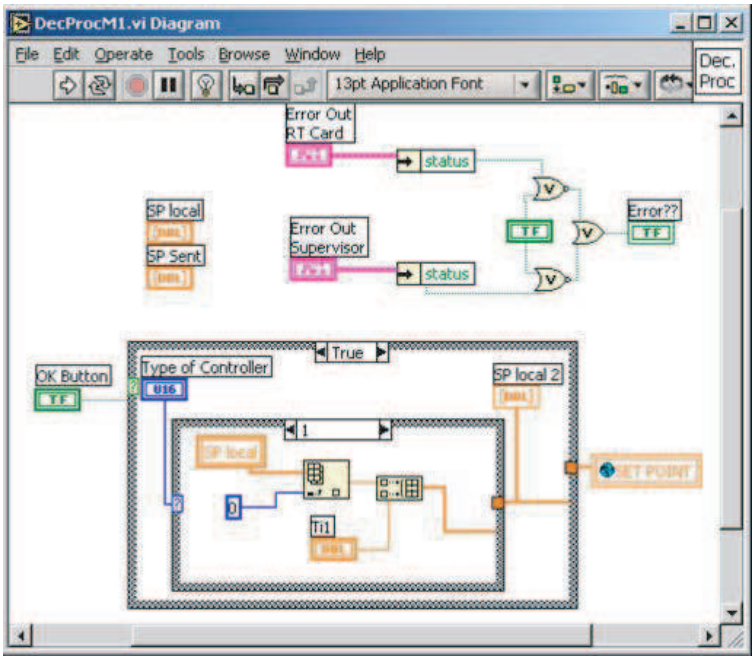

Figure 2.24: Block diagram for the decision process of the middle level.

Finally, when a decision is made, we need to send some data to the supervisor and to the lower level. As we can see in Figure 2.26 the idea of the post-process is to take the value that comes from the pre-process, put it inside a cluster with the status of the "OK Button" and the desired temperature value that comes from this level, and send this data to the supervisor. The variable used for this case is called "Envio", and the way that is sent is shown in the lower part of Figure 2.26. In the upper part of this block diagram we can see the value that will be sent to the lower level. In this case we do not use a VI to convert from a cluster into a string as the one used for sending the data to the supervisor. The reason is that in this case we have only an array to send, but in the other case we have a cluster. In the front panel (Figure 2.27) we can see how the cluster "Envio" is built, and which elements are introduced here.

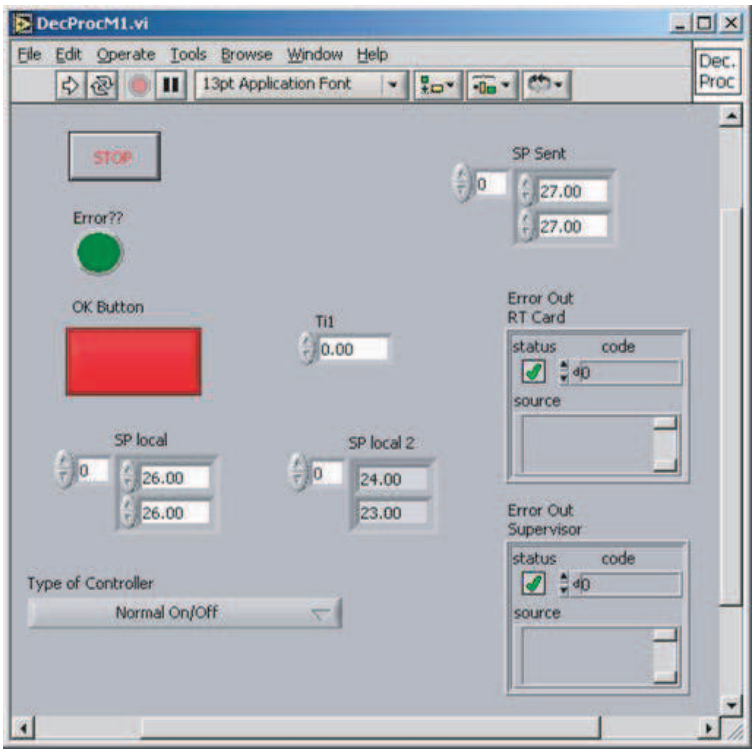

Figure 2.25: Front panel for the decision process of the middle level.

Finally, to close all the connections when we have an error, we need to put two extra elements outside the loop as one can see in Figure 2.18.

2.3.2 Supervisor

The supervisor (which has the block diagram shown in Figure 2.28) uses the data that comes from both modules in the middle level (on EEPC335 and EEPC336). As we can see in Figure 2.29 we supervise the status of each module to set whether we can control the experiment from the supervisor or we can only watch the values that come from each of the sensors. This status is in the upper right part of the front panel and if it is red that means that the supervisor cannot be in charge. Otherwise, if it is green, we can control the experiment from here. When we said that we can control the experiment from the

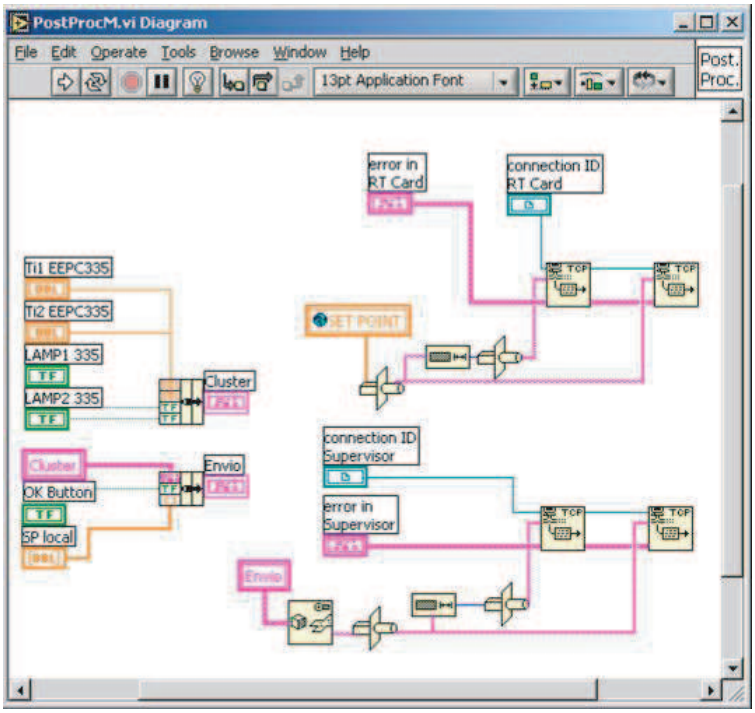

Figure 2.26: Block diagram for the post-process of the middle level.

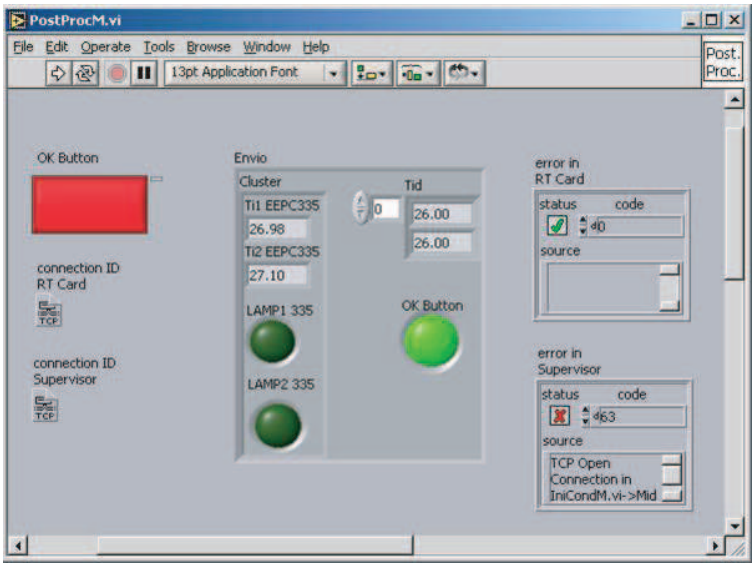

Figure 2.27: Front panel for the post-process of the middle level.

supervisor, we meant that we can change the set points, or we can use another controller that was implemented at that level. To select the control strategy of the supervisor, we need to change the status of the switch called "Change Controller." The idea of this new strategy is to force the set points of EEPC336 to be the actual temperature values of EEPC335. This part of the algorithm is implemented through a "Case" structure.

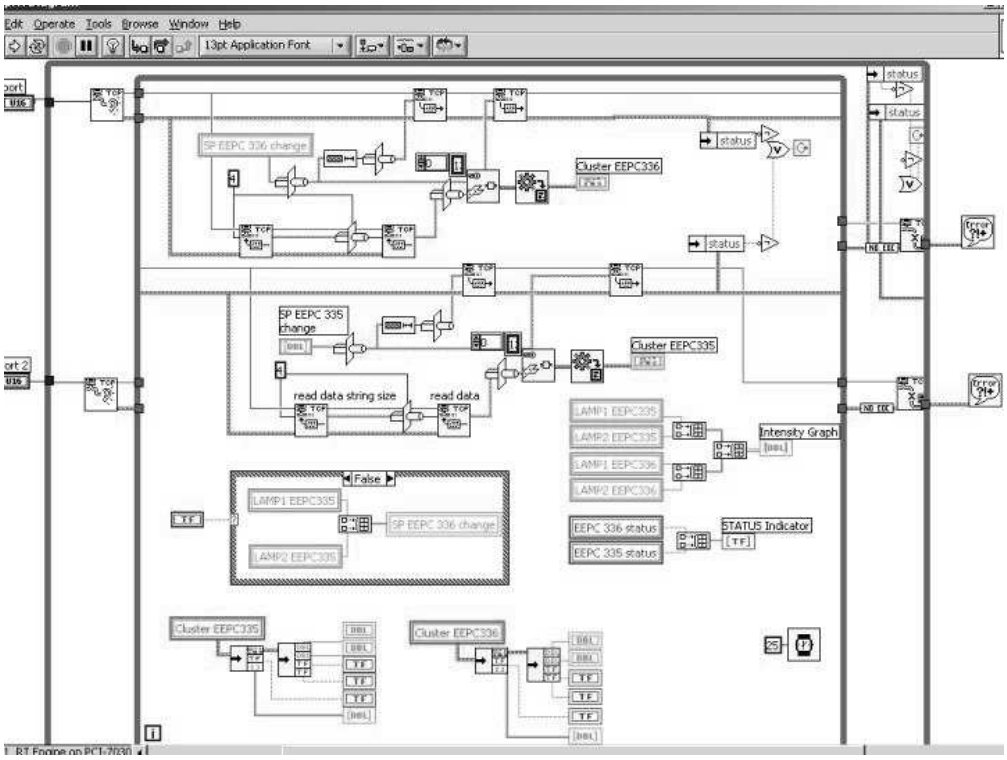

Figure 2.28: Block diagram for the supervisor.

The idea of this VI is to be a server, where we listen to the ports defined by the two computers (in this case we picked ports # 2000 and 2001), and using the same strategy explained in Section 2.2 we transmit the data from one level to the other. Here we defined the index of the VI called "Flattened string to variant" using the same VI shown in Figure 2.23. We unbundle the data that comes from the cluster and we put each element as a double or

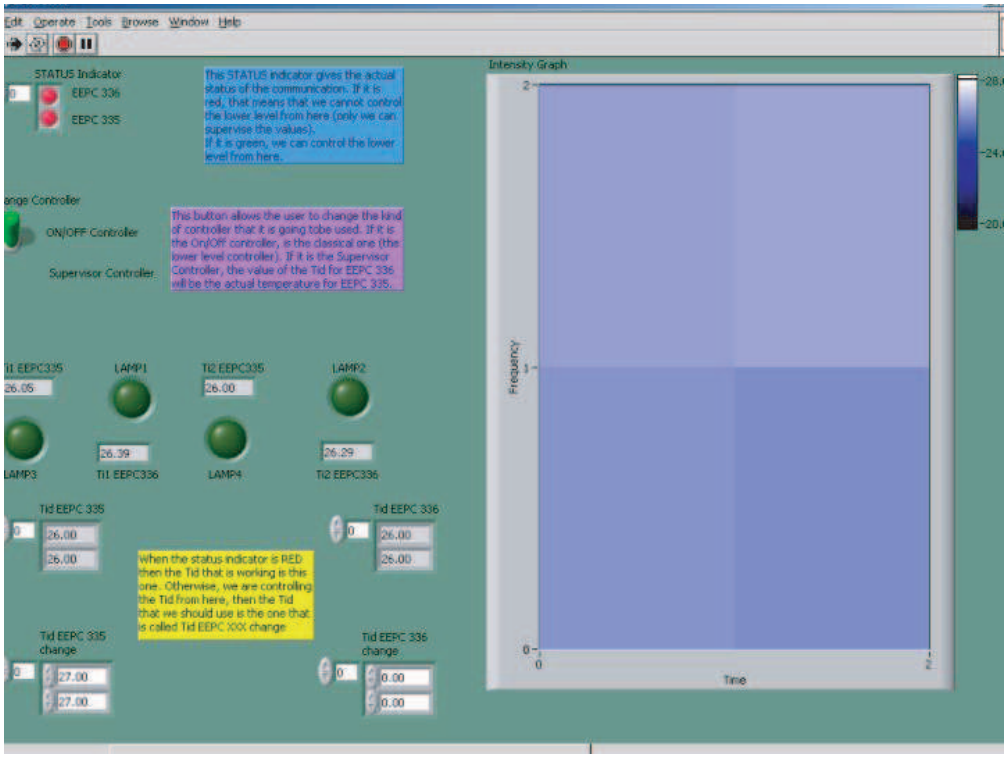

Figure 2.29: Front panel for the supervisor.

boolean. That is why we can see in Figure 2.29 that each lamp is in front of its own sensor as it is physically in the bread board. Finally, we added an intensity graph to show how the temperature has been sensed from each of the four zones implemented in hardware. In Figure 2.29 we can see some of the changes in this graph.

The methodology behind the supervisor is the same that we found in RCS. The difference is that the blocks are all inside a main while loop as we explained before. But as we can see, the pre-process is done when we read the data that comes from any of the computers. Here we use the same strategy used in Section 2.3.1. The decision process is where we see if the system is operating from the supervisor, and if it is, we compare with the actual value of the type of controller that we want. In Figure 2.29 we can see the supervisor is not controlling the system, since the status of each module is off. Finally, the post-process is where we

bundle all the variables, we send the data to the middle level, and we use the intensity graph to see the actual behavior of each zone.

2.4 Operation

2.4.1 Running the Software

To operate correctly all these VIs, we need to follow certain steps. First, we need to run the lower level VI using the NI RT card. For that, first we click under *Operate* the item called *Switch Execution Target*. The window that is going to appear is the one that is in the left of Figure 2.30. Here, we select the NI RT card. Once we are here we need to open the VI designed for the lower level, and run the experiment. Since the NI RT card can run alone, we need to change the status of LabVIEW to be a host (following the same steps as before, but now using the arrow that appears in Figure 2.30 we select the window that is on the right). Here we open the two VIs that implement the middle level and the supervisor. Be sure to first run the supervisor, and after that we can run the middle level because the middle layer works as a client, and as we mentioned before it should be listening the lower level and the supervisor (this is similar to running the servers of the NML buffers first if we use the RCS library.) In both front panels that we have we can change the status of each computer, and the control strategy that we want to use in the experiment.

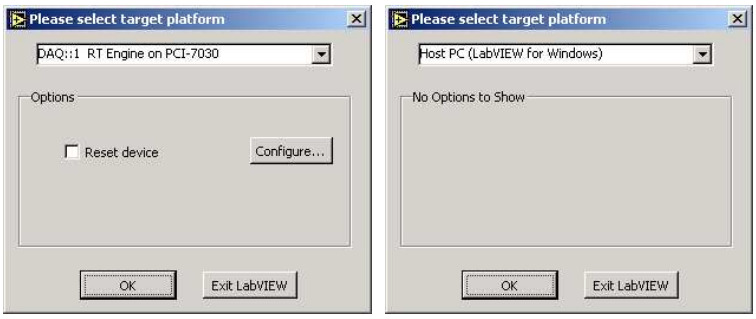

Figure 2.30: Selection of the target platform.

2.4.2 Example Operational Scenarios

To give some examples of the operation of the software, let us take a look at the supervisor. In Figure 2.31 we can see that the experiment is running from the supervisor since the status of both computers is green (upper left part of Figure 2.31). If you do not remember how to operate the system, you have some guides in the front panel. Here we can notice also that the controller that is working is the one that we implemented in the supervisor level. The idea is that the temperature in zones 2 and 4 will follow the ones in zones 1 and 3 respectively. That means that T_i^d for the zones controlled by EEPC336 will be the T_i for the zones in EEPC335. The intensity graph shows how high the temperature of each zone is. Since we need to see the results on the web, we use the web publishing tool that comes in LabVIEW. The supervisor can be viewed using any web browser, however in order to be able to see an update of each of the values, we need to use Netscape, as we can see in Figure 2.32.

2.5 Conclusions

In this experiment we developed a hierarchical controller that follows RCS methodology. In the lower and middle level, we implemented VIs similar to the pre-process, decision process, and post-process discussed in [1]. The information that is acquired is from four temperature sensors (two in each computer called EEPC335 and EEPC336) and four actuators (in this case four lamps that are the heaters of our system). Each computer manipulates an RT card and the hardware is divided into four zones using the scheme shown in Figure 2.1. To transmit all the data that we have in this lower level to the supervisor, we need first to transmit the data to a middle level. This information is passed using the classical TCP/IP connection, using a scheme of client-server. The client in this case will be the middle level, and the server will be the lower level (and the supervisor). In this middle level, the user can control the experiment, and receive all the temperature values acquired by the sensors.

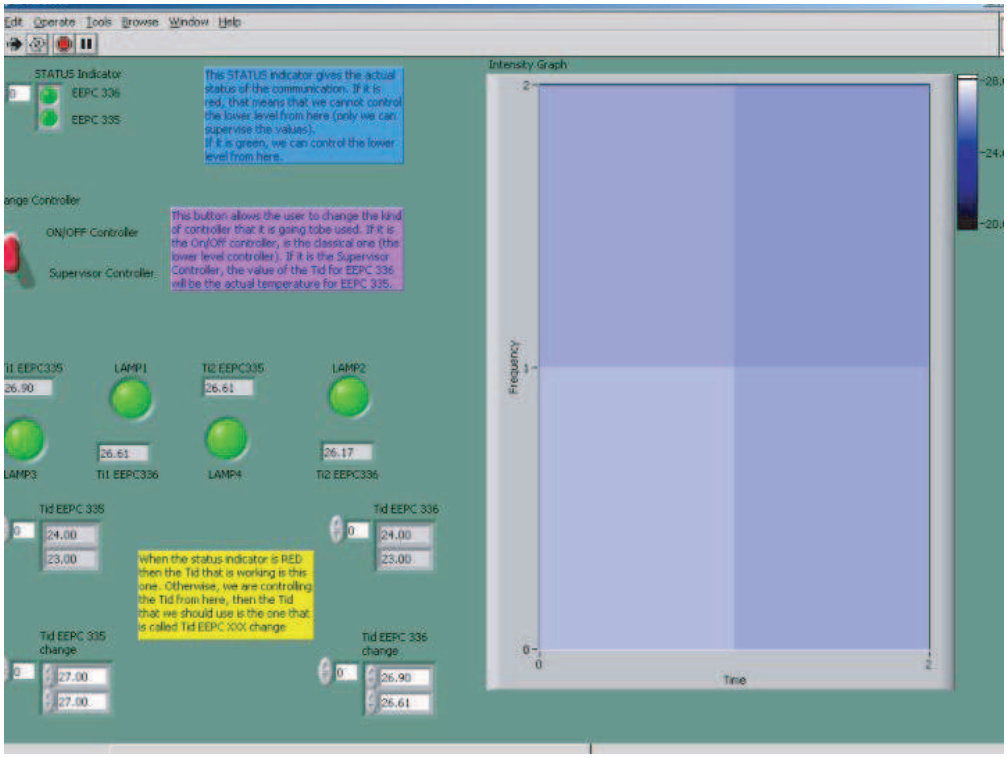

Figure 2.31: Front panel for the supervisor.

This information must be shared with a program called the "supervisor" such that from here we can see the information anywhere (even through the internet). The information is passed to this higher level using the same strategy of client-server, and since the middle level should be a client (because of the methodology implemented in the lower level), the server will be in this case the supervisor. This module, acquires the information that comes from the modules implemented in the middle level, and if the user allows, we can control the experiment from here. Again, the communication protocol used was TCP/IP since the DataSocket implemented by NI presented a lot of delays.

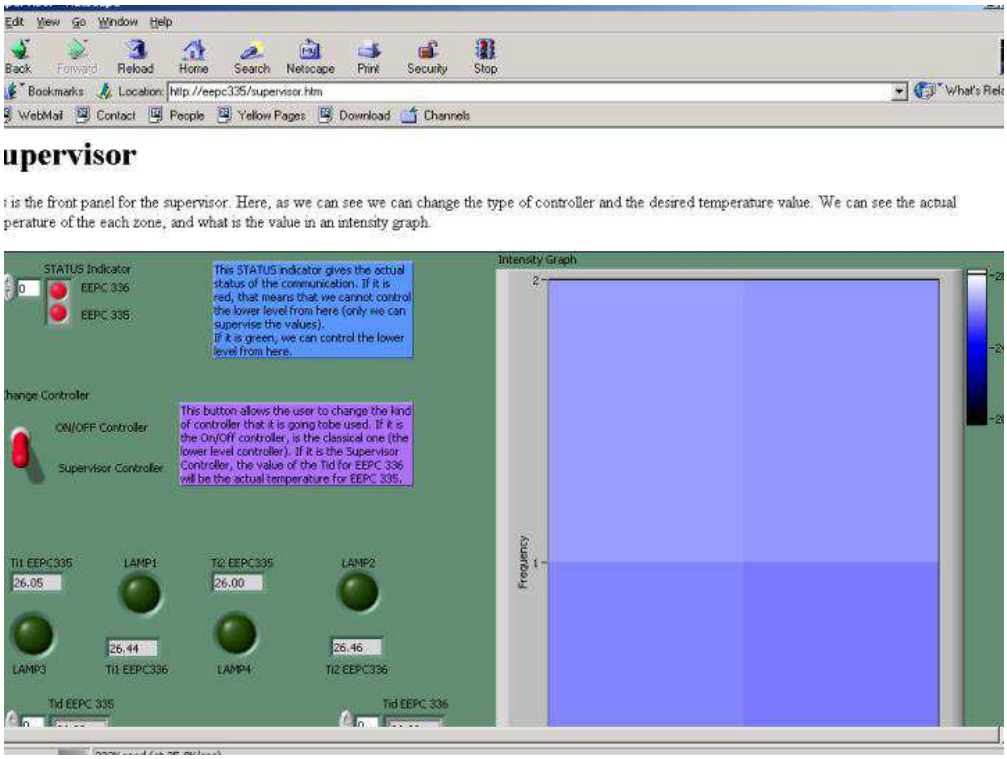

Figure 2.32: The supervisor running under Netscape.

CHAPTER 3

dSPACE

Centralized and decentralized controllers are developed here for a temperature control grid consisting of 16 zones, each of which has a temperature sensor and heater. The physical layout of the temperature control grid experiment results in the heater from each zone affecting the temperature in some neighboring zones. Also, disturbances from ambient temperatures and wind currents have a significant impact. The zones are divided into two groups, (each with eight zones) and the eight analog inputs on each of two DS1104 dSPACE cards on separate computers to interface to the analog temperature sensors. Eight digital outputs are used on each DS1104 dSPACE card in the corresponding zones to turn the heaters (lights) on and off. The sensed signals from one computer are transmitted over a RS-232 link to the other "main" one and the main computer can transmit commanded digital heater inputs to the other computer's DS1104 dSPACE card. In this manner we can implement centralized or decentralized controllers for the multizone temperature control problem.

In this case, we implemented a centralized controller, where we divided the grid into four big zones (called "superzones"), each consisting of 4 sensors and heaters, and we implemented two different strategies to track different desired temperatures.

3.1 Temperature Control Experiment Hardware and Challenges

We developed our multizone temperature control experiment to be as simple as possible but still rich enough to study the ability of a variety of centralized and decentralized controllers to meet different control objectives. As shown in Figure 3.1, the experiment consists of sixteen different zones numbered 1 through 8 for the ones that are controlled by the DS1104 dSPACE card # 1 which is the main computer, and the others called TX1 through TX8 for those controlled by the DS1104 dSPACE card on the other computer. We add an extra notation above each lamp and sensor for the mathematical development that we will do in Sections 4.1 and 4.2. The control input is labeled u_i, $i = 1, 2, \ldots, 16$, and the temperature sensed will be called T_i. The odd numbers are those associated with the dSPACE DS1104 card # 1, and the even numbers belong to the DS1104 dSPACE card # 2. The two computers can be viewed as being in a "master/slave" configuration. The one that is the "master" is the one that has the signals that belong to the DS1104 dSPACE card # 1, and the "slave" is the one that handles signals named TXp, where $p = 1, 2, 3, 4, 5, 6, 7, 8$. The analog inputs take the signals that come from the sensors, and the digital I/O (DIO in the figure), are connected to Darlington devices (National Semiconductors DS2003) to handle the current that we need for each lamp (each lamp needs 100 mA). The sensors are National Semiconductors LM35CAZ temperature sensors, and these convert from degrees Celsius to volts. The data sheets are given at the web site mentioned in Chapter 3.2.3. Both computers are connected using a serial interface, RS-232. This configuration does not allow communication between more than two computers at the same time.

There are two types of control objectives for this experiment, both of which will be studied:

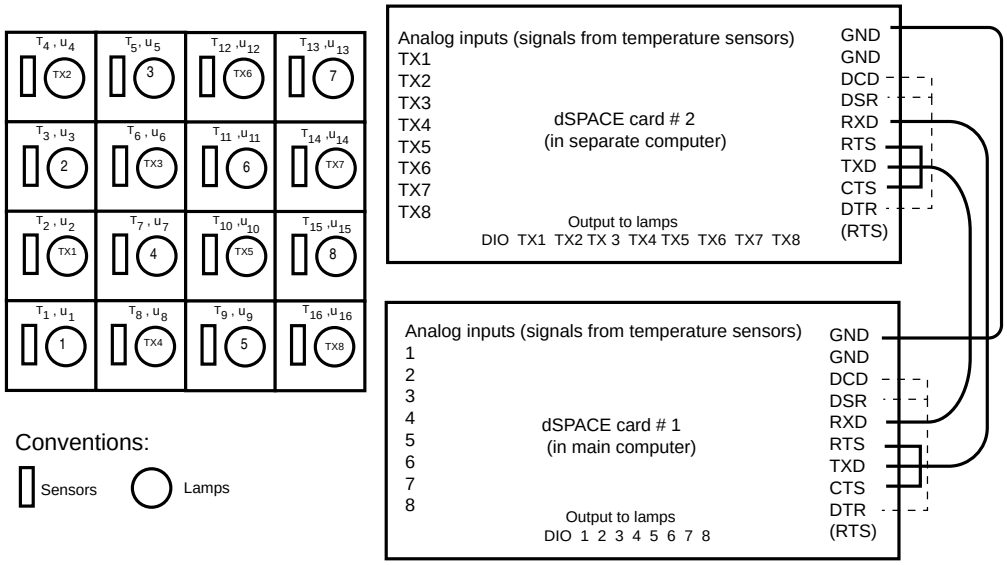

Figure 3.1: Hardware for the temperature control experiment.

1. Temperature tracking: For instance suppose that we group the 16 zones into four contiguous sets of four zones. Then we can try to make the temperatures within each of the four "superzones" track one temperature, or we could try to make them track the temperature of another superzone.

2. Maximum uniform temperature: In this case the objective is to make the temperature in each zone as high as possible, but to also achieve a uniform temperature across all 16 zones.

Both these objectives are difficult to meet due to ambient temperature and wind current influences, and communication constraints that lead to decentralization of the controller implementation (e.g. communication delays and communication topology constraints on what local controllers can sense in order to determine whether or not to turn on the heater in their zone).

Another problem is the calibration. When we put the sensors in the board, we picked 16 that seemed to generally have the same characteristics (i.e., the temperature measured was practically the same). In this way we tried to minimize the impact of fabrication differences. But, once we start soldering we could change some of these characteristics that we cannot measure. Calibration after construction of the experiments is a possibility but extremely tedious, time consuming, and error prone due to the need to do so for 16 sensors that are in close proximity, and due to the ever-present ambient influences. Hence, in effect our calibration approach is based on calibrating one sensor and then picking 15 others that provide temperatures close to it, *before* construction. Next, to help with the interpretation of our results we use an external temperature probe which is as accurate as our sensors and properly calibrated to provide the ambient temperature at the start of each experiment. This temperature varies according with the hour of the day. Most of the experiments were done during the summer (July-September) when the temperature in the room was between 21 and 23 degrees Celsius.

3.2 Superzone Temperature Tracking

3.2.1 RCS Methodology

To develop the block diagrams in Simulink for the experiment we followed the RCS methodology described in [1]. The idea is to have a pre-process, a decision process, and a post-process. In the pre-process we will acquire the data, do some conversions, apply some scaling factors for the signals that are acquired, and store this information in some global variables that will be used in the decision process. All of these computations must be done each cycle, and for all of the experiments we choose 100 ms as the sampling time. In the decision process, we develop a couple of subsystems (i.e., the name used in Simulink to designate each of the elements that a complex block diagram has to make everything more

compact and easy to read) that will be in charge of making decisions concerning the control (if you are in the main computer), or other tasks such as update variables to be used in the post-process. The post-process will send the data needed by the other computer, and will update the digital outputs.

RCS has design and diagnostic tools that are used to develop the control algorithm for the experiment that we are performing, and to change some variables in real time. In dSPACE-Matlab these two tools can be viewed as Simulink and the GUI that is provided in dSPACE. In Simulink we develop the controller and all the necessary functions to run the experiment. Once we have the code, that we call the "model," we compile it and following some steps that are transparent to the user, we obtain a file that will run the code in real time, and provide the ability to set up a user interface. This GUI in dSPACE can be viewed as the diagnostic tool, since we can change some variables, and we can see in real time some of the variables defined by the user in the model.

3.2.2 Control Strategy

This first experiment is called "superzone" temperature tracking because we group the 16 zones into 4 "superzones" as we can see in Figure 3.2 and study tracking desired temperatures. The objective is to implement two different tracking strategies. First of all, we develop one strategy where we set a desired temperature T_i^{ds} $(i = 1, 2, 3, 4)$ and each of the superzones will track this temperature. The next strategy is quite similar, but superzones 2 and 3 will not have a fixed desired temperature. For this case, the user can change the desired temperature for the superzones 1 and 4, but the desired temperature T_i^{ds} $(i = 2, 3)$ will be the average temperature of superzones 1 and 4 respectively.

The Simulink code that was developed is shown in Figure 3.3. As we can see in that figure, we followed RCS methodology described in Section 3.2.1, so we have a pre-process, a

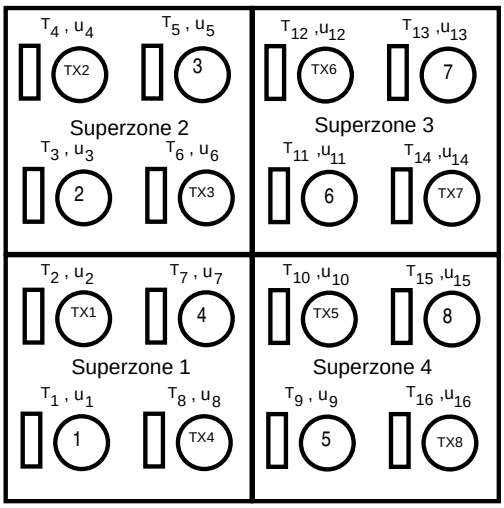

Figure 3.2: Zones division for the experiment.

decision process, and a post-process. As we explained in Section 3.1 we use two computers, one of them is going to be the "master," and the other one the "slave." Both computers have the same structure, the only difference is that the decision process is different. The "slave" computer does not make any control decisions, since the "master" is the one that makes all the decisions. The idea is that the "slave" computer transmits the data that it acquires from all the sensors, and it receives the values that it has to put in each of the lamps in the zones TXp, $p = 1, 2, \ldots, 8$. Next, we are going to explain the block diagram that was designed in the "master" computer, since it has all the information that we need to explain the experiment. The "slave" computer has a large difference in the decision process, since it does only a variable conversion such that the post-process can put the correct data in the digital port. The pre-process and the post-process are practically the same as the "master" computer.

Let us start by explaining the pre-process in Figure 3.3. In Figures 3.4, 3.5, and 3.6 we can see the elements that belong to the pre-process block. The idea of the pre-process

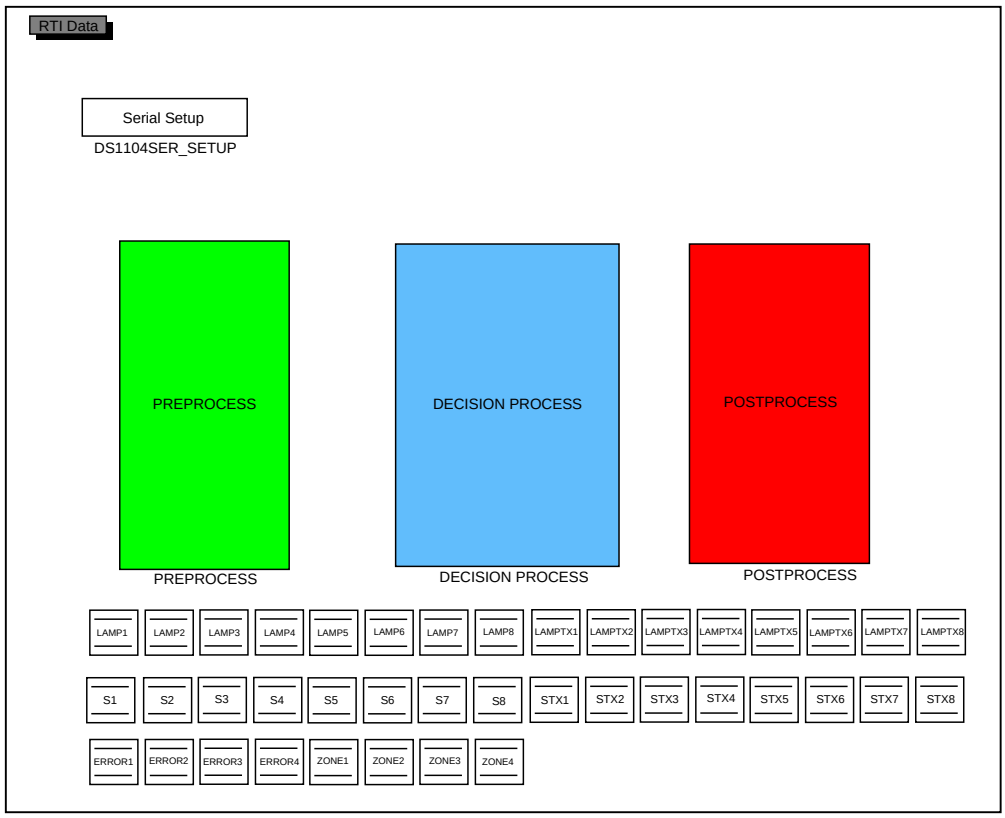

Figure 3.3: Block diagram for the superzone temperature tracking experiments.

is to acquire the data that comes from the sensors that belong to the "master" computer and at the same time it should receive the data that comes from the "slave." As we can see in Figure 3.4, we have one block called "Pre-Process" where we acquire the data from the sensors that belong to DS1104 dSPACE card # 1 (as we can see in Figure 3.5), and some other blocks that receives the data via RS-232, and converts the value that originally should be scaled between 0 and 255, into a double number. The variables max_temp and min_temp that are in Figure 3.6 are defined by the user in a program that runs at the same time as we compile the program (for the results that are going to be presented in Section 3.2.3 $min\_temp = 20$ and $max\_temp = 38$). These variables give the resolution that we have in the sensed values. We needed to fix a minimum and a maximum temperature value, because

when we send the data via RS-232, we need to convert the value into an 8 bit resolution (i.e., an integer value between 0 and 255). With the temperature that we have in the room, and a couple of experiments, we determined that those values were small enough to give a good resolution (the values change in increments of 0.07 degrees Celsius). Also we have the blocks called "Average Superzone i" ($i = 1, 2, 3, 4$) where we are going to compute the average temperature T_i^{as} that is going to be stored in a variable called "ZONEi" where i is the number of the zone that we are analyzing.

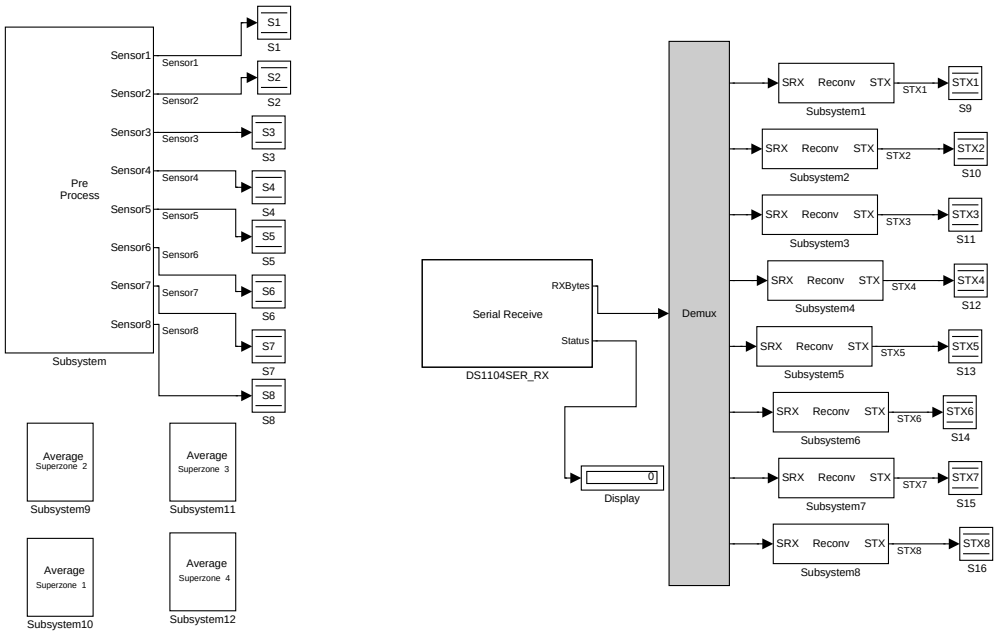

Figure 3.4: Pre-process for the superzones temperature tracking problem.

The decision process, shown in Figures 3.7 and 3.8 holds the control strategy that we are going to use. Since we divided the zones into superzones as we can see in Figure 3.2, we will have four superzones. These four superzones can be controlled to their own desired temperature T_i^{ds}, $i = 1, 2, 3, 4$, that the user changes in a dSPACE GUI, or the desired temperature T_j^{ds} for $j = 2, 3$ will be the average temperature T_k^{as} of $k = 1, 4$. The selection

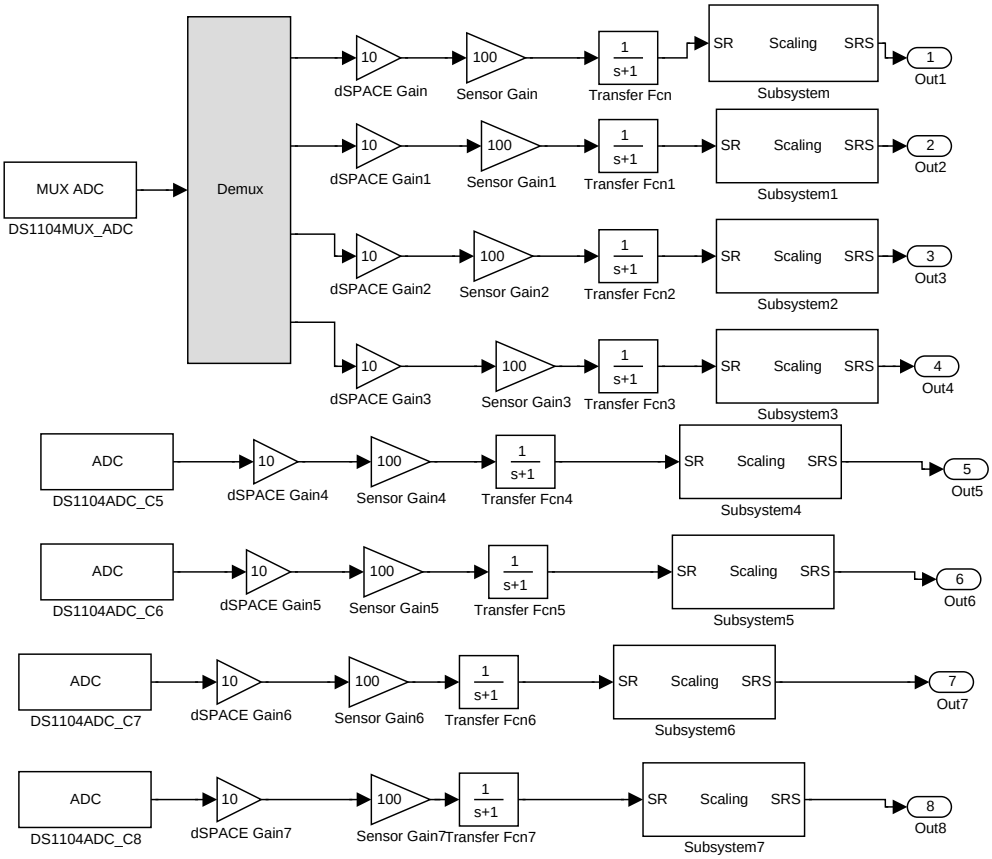

Figure 3.5: Analog data acquisition.

of any of these strategies is done via dSPACE GUI, as we can see in Figure 3.9. In other words, the control strategy developed can be explained as follows. Each of the superzones has its own desired temperature T_i^{ds}, $i = 1, 2, 3, 4$, to be reached. During the pre-process, each of the zones will compute the average temperature T_i^{as} that is going to be stored in a variable called "ZONEi" where i is the number of the superzone that we are analyzing. Then, we compute the error $E_i = T_i^{ds} - T_i^{as}$, and if this error is negative, all the lamps in the zone are off. But, if E_i is positive, then we have to see how negative it is with respect to T_i^{as} and make the decision. For that, we compute another error called $e_i = T_i^{as} - T_i$ and if the value of $E_i - e_i$ is positive, then the i^{th} lamp should be on, and off otherwise. With that

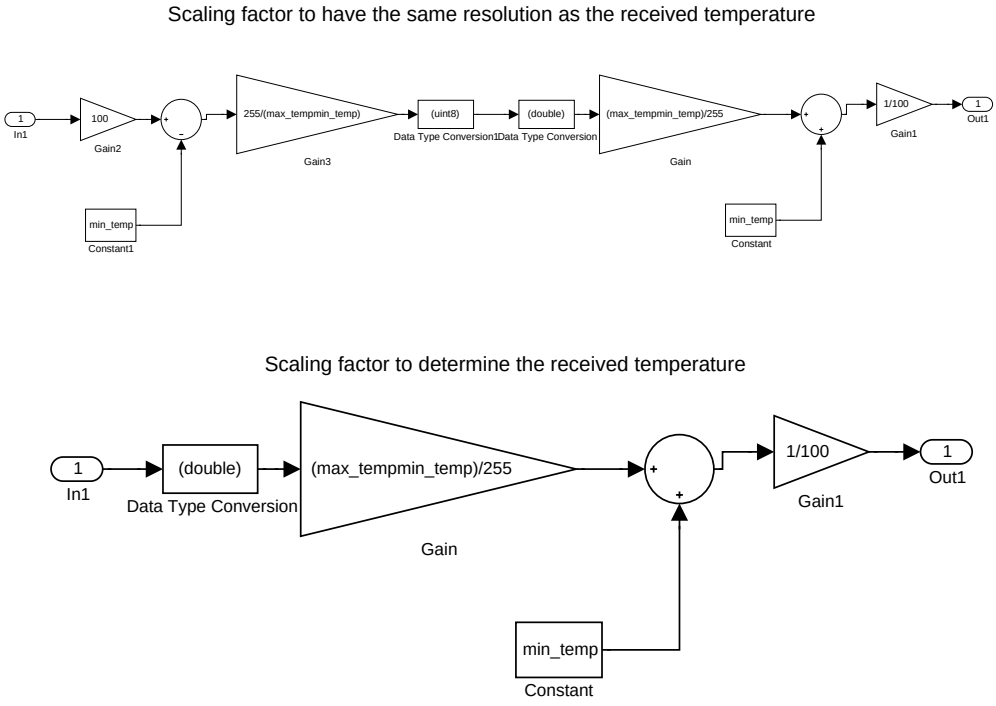

Figure 3.6: Scaling factors.

we guarantee that not all the lamps will be on if E_i is negative, because the influence that some of the lamps have in the other superzones is high, and it makes a large disturbance. Also we notice that with this strategy, some of the zones could have a high temperature, but the superzone will maintain a stable temperature.

The post-process for this case is shown in Figure 3.10. Here, we update the values in the digital outputs for the lamps that belong to the main computer, and for those that belong to the other one, we send the data via the RS-232 link.

3.2.3 Results

To show how the experiment operates, we run the experiment using the two strategies. First, we fixed the desired temperatures as follows: $T_1^{ds} = 24$, $T_2^{ds} = 25$, $T_3^{ds} = 26$, and $T_4^{ds} = 26$. Figure 3.11 shows the behavior for this case. As we can see, T_i for $i = 2, 3, 4$

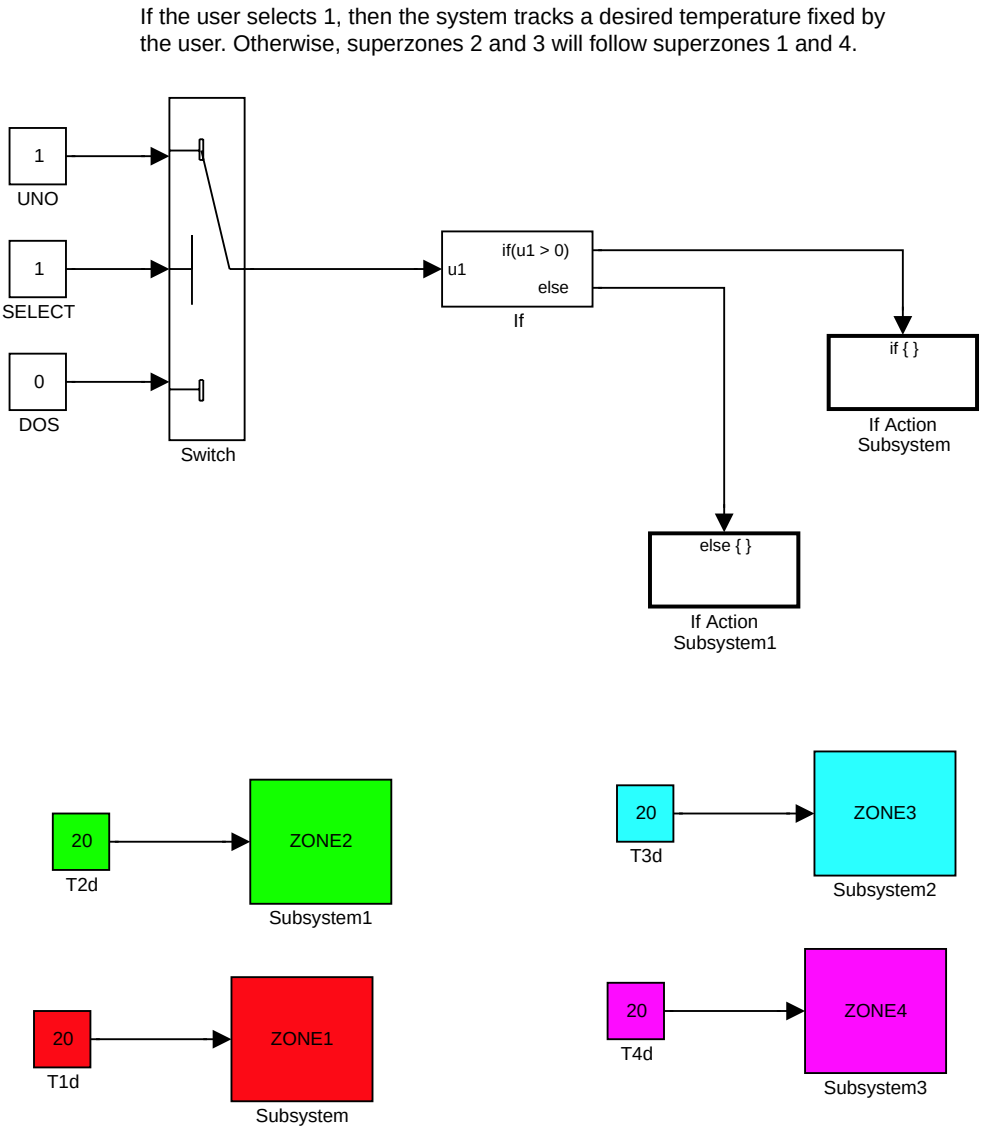

Figure 3.7: Selection of the strategy and desired temperature for each zone.

arrives to the desired temperature, meanwhile T_1 remains high. The problem in this case is that since we fixed T_2^{ds} and T_4^{ds} at high values the temperature T_1^{as} will be affected by the influence of the other superzones. Therefore the final values that we got after running this experiment for 200 seconds where the following:

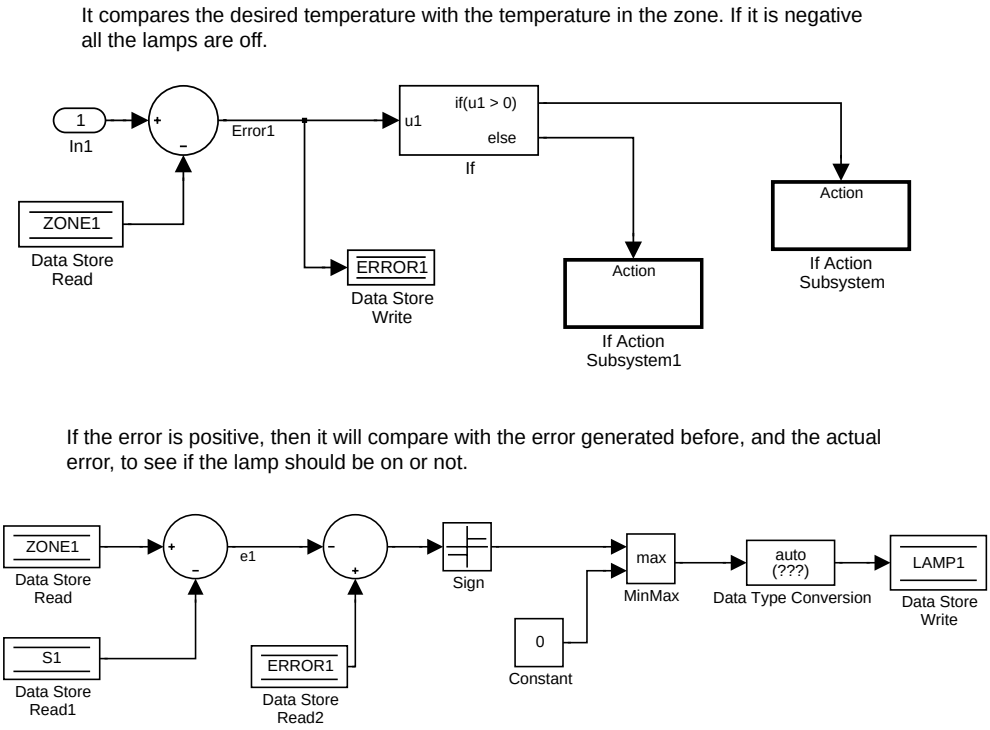

Figure 3.8: Control strategy for the superzones temperature tracking problem.

- For superzone 1, T_1^{as} will be around 24.9941 and 25.1529 degrees Celsius that is considerably high, because of the huge interaction that we have between different superzones.

- For superzone 2, T_2^{as} will be around 24.9412 and 25.1353 degrees Celsius that is a value really close to the desired temperature T_2^{ds}. The steady state error is small, and since the controller is totally digital, then we will never reach the exact value of T_2^{ds} because of the multiples disturbances described before.

- For superzone 3, T_3^{as} will be around 25.9647 and 26.0353 degrees Celsius. We will have the same disturbances as before.

- For superzone 4, T_4^{as} will be around 27.0059 and 27.1294 degrees Celsius. We will have the same disturbances as before.

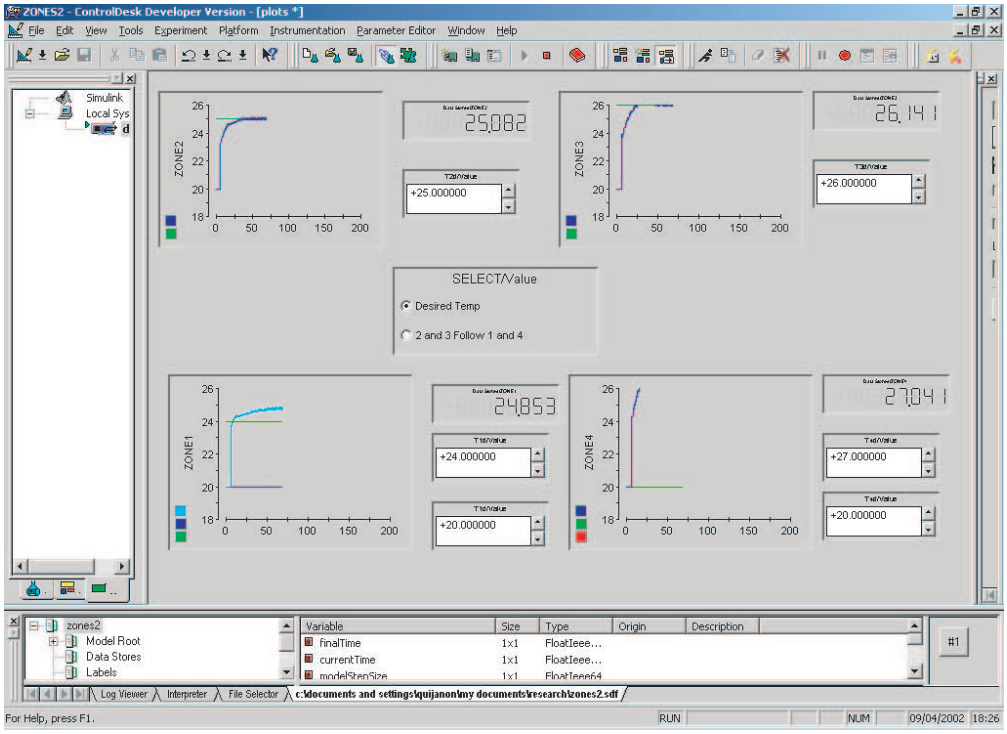

Figure 3.9: GUI for the superzones temperature tracking problem.

The desired temperatures T_i^{ds} were chosen according to the temperature in the room, such that most of the zones will arrive to the desired temperature. To do that we used a Fluke meter 179, and we measured this ambient temperature. For the case described in Figure 3.11 the temperature in the room was equal to 22.8 degrees Celsius at the start of the experiment (with a couple of degrees of variations during the execution of the experiment due to external factors). Then practically all the superzones arrived to the desired temperature, and only the one that has the smallest desired temperature did not arrive to the final value.

In Figure 3.12 we change the strategy (with the same ambient temperature as before), and now instead of having a desired temperature for all of the superzones, we define $T_1^{ds} = 25$ and $T_4^{ds} = 27$. The desired temperature for superzones 2 and 3 is the actual average temperature of zones 1 and 4 respectively. As you can see in Figure 3.12, all the superzones track correctly

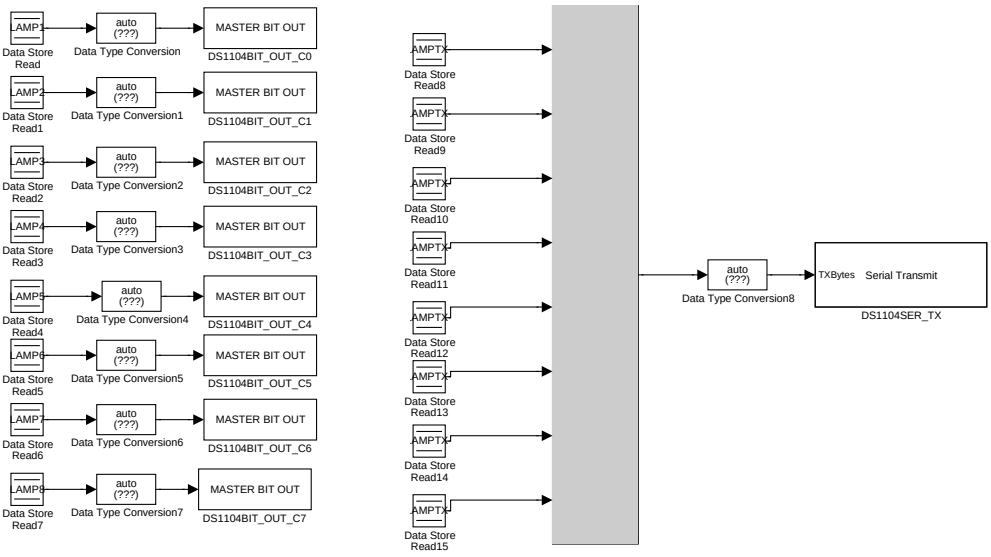

Figure 3.10: Post-process for the superzones temperature tracking problem.

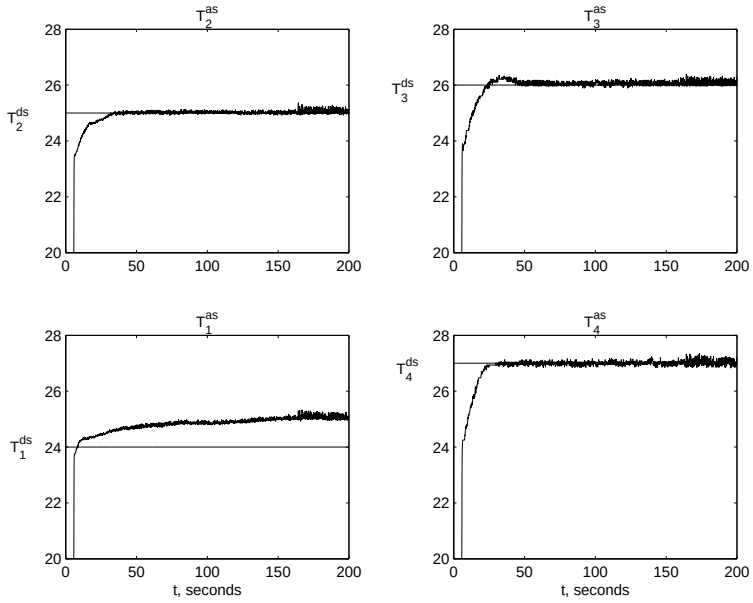

Figure 3.11: Response of each of the superzones, when the desired temperature is fixed by the user and the temperature in the room is 22.8 degrees Celsius.

the desired temperature. Taking some data, we found that the final values for each superzone were:

- For superzones 1 and 2 we have final values near 25.15289 and 25.1176 degrees Celsius respectively.
- For superzones 3 and 4 we have final values near 27.0941 and 27.200 degrees Celsius respectively.

Each superzone tracks almost perfectly the other one and, as always, there is a large interaction between different superzones. Keep in mind also that we do not really know what is the calibration value for each sensor, and clearly this can also affect the tracking behavior.

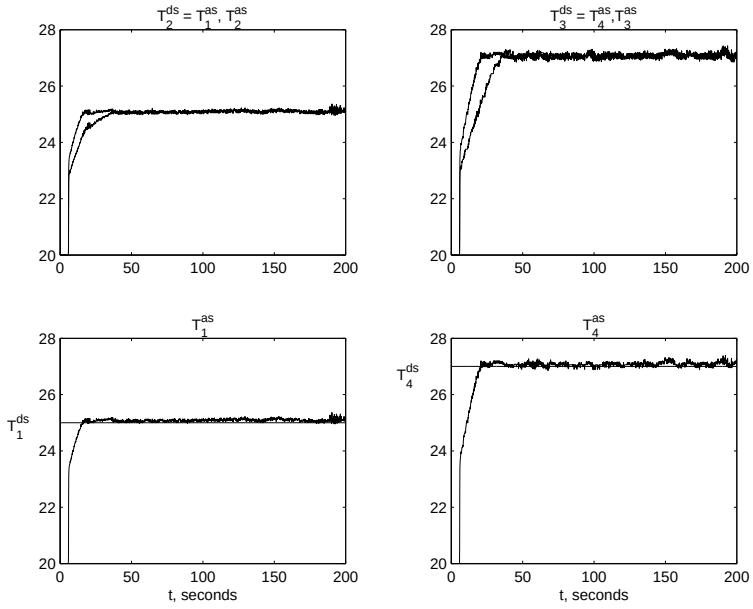

Figure 3.12: Response of each of the superzones, when the desired temperature for superzones 1 and 4 is fixed by the user, and the desired temperature for superzones 2 and 3 is the actual temperature of 1 and 4 respectively. The temperature in the room is 22.8 degrees Celsius

CHAPTER 4

RESOURCE ALLOCATION EXPERIMENTS

Using the same hardware configuration as in Chapter 3, we implemented two types of "resource allocation strategies." For the first one we allow one heater to be on at a time and view it as the "resource" that must be allocated to try to get a uniform but highest possible temperature across the grid. To do this, at each time instant it applies a pulse on the heater corresponding to the zone that is sensing the lowest temperature. We study the impact of delays between applying these pulses as this represents one way to study the effect of limiting availability of the resource. Moreover we study the impact of delays in sensing temperatures. For our second resource allocation strategy, at each time instant, the average of all temperatures is computed and considered at that time instant, to be the desired temperature for every zone. Then, the controller turns the heater on in any zone whose temperature is below the average. Hence, at each time instant there will in general be more than one heater on and there is a pattern of distribution of resources (heat) to try to eliminate lack of uniformity of temperature across the grid and to raise the temperature as high as possible.

We also implemented a decentralized controller that basically corresponds to a decentralized version of the first resource allocation strategy described above. To achieve decentralization we set up a "communication topology" that allows a "local controller" for each zone to sense the temperatures in neighboring zones (middle zones have more neighbors than edge

zones). The goal is as above, to achieve the highest possible uniform temperature. To do this, each local controller turns on its heater if the temperature in its zone is the lowest of all the zones it can sense. We study the impact of disturbances and delays in the local controllers sensing the temperatures in neighboring zones. Both of these, and decentralization, have significant impacts on the ability to maintain a uniform temperature across the grid.

4.1 Centralized Resource Allocation

4.1.1 Control Strategy for a Centralized Resource Allocation Strategy: Heat Minimum Temperature Zone

For the following experiments, we will use the 16 zones numbered T_i, $i = 1, 2, \ldots, 16$. The control input (i.e., the value that we apply to the lamp) is u_i for $i = 1, 2, \ldots, 16$. The first control strategy simply applies a pulse at each time instant of 100 ms duration to the zone that has the lowest temperature. We are trying to allocate a "resource" (the time that the light is on) and the objective is to try to reach the maximum temperature with some kind of uniformity across the temperatures of all the zones. In other words, at each time k, for each i

$$u_i(k) = \begin{cases} 1 & \text{if} \quad T_i(k) \leq T_j(k) \quad j \in \{1, 2, \ldots, 16\} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Figures 4.1, 4.2, and 4.3 show how the experiment behaved. First of all, Figure 4.1 shows how many lamps were on during the 200 seconds that we acquired data. As we can see six of the sixteen lamps never turned on. That can be explained by the large influence that each of the zones have in each other, and the fact that the algorithm turns on the lamp that is coming first in order. In other words, if more than two lamps are tied in the minimum value (that could be possible because of the scaling factors that we put to have consistency with the values that are sending from the other computer) then we give priority to the lamp that is in the main computer, and has the lowest value (T_i for $i \in \{1, 3, 5, \ldots, 15\}$) . But,

even with some of those lamps off, the whole grid reaches a practically uniform final value. For this case, we started with a room temperature of 22.8 degrees Celsius, and the average final value that we obtain is around 24.1 degrees Celsius. Some of the zones are more warm than this value (the maximum value was 24.8 degrees Celsius), but we have to take into account the disturbances associated with the experiment. Another problem is that the room temperature is variable, thus the initial conditions will change each time that we want to run an experiment, so we cannot predict a final steady state value for this experiment.

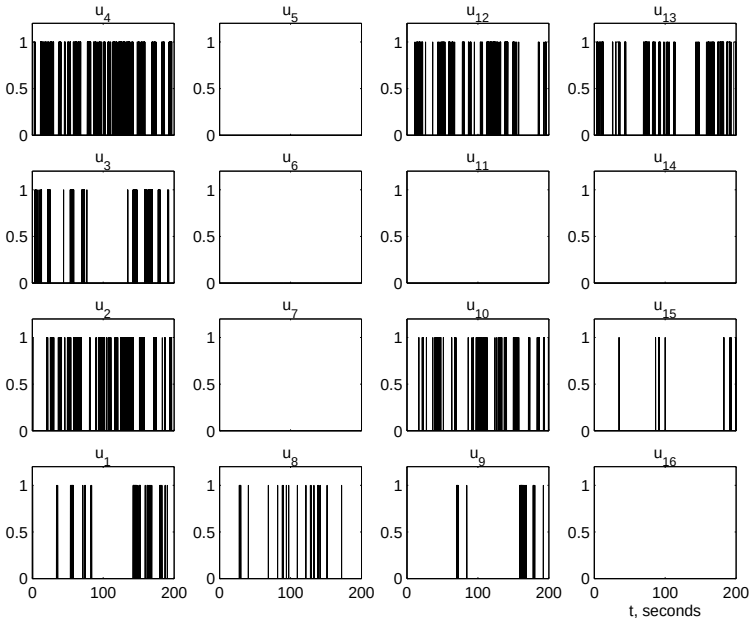

Figure 4.1: Lamps on in a centralized resource allocation experiment. This experiment was done with a 22.8 degrees Celsius of temperature in the room

In Figure 4.3 we have embedded a movie. For those who cannot run the movie (that is available in the web site defined in Chapter 3.2.3) we are going to explain the meaning of it. We have 16 big bars that describe the temperature for each zone, i.e., the T_i, $i = 1, 2, \ldots, 16$. In the lowest left corner we will have T_1 and we have the same structure as the one that we

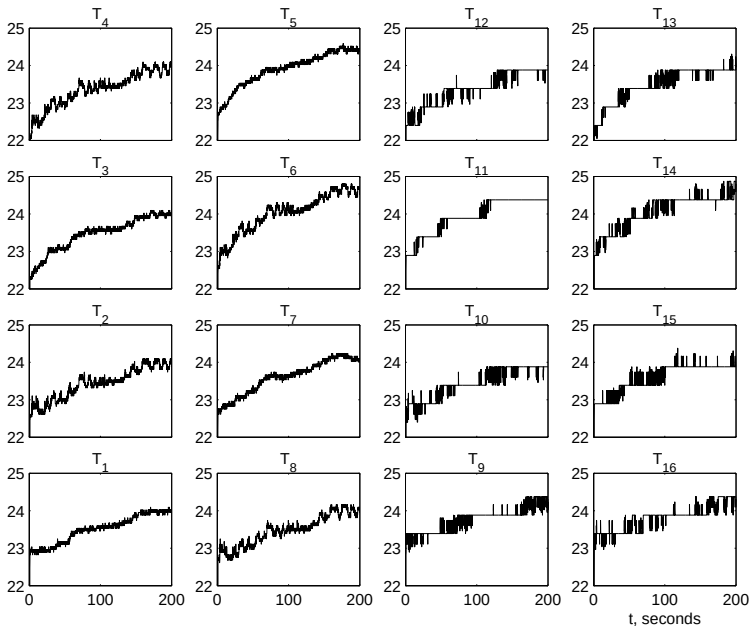

Figure 4.2: Sensor behavior for a centralized resource allocation experiment. This experiment was done with a 22.8 degrees Celsius temperature in the room.

described in Figure 3.1, so the upper right corner will be T_{13}. We have a small block bar in between the zones that represent the lamps (because physically that is how the lamps are in the experiment). When we click in the figure, we see how the lamp is on only in the zone that has the minimum temperature value. As the experiment continues to run, we see how the temperature starts to increase, and after a couple of minutes it starts to remain constant. The only axis that should be taken into account is the *z-axis* since it has the temperature in degrees Celsius and the lamp input. The y and *x-axes* do not mean anything in particular as they just provide the index grid for Matlab to draw the figure. Since the minimum temperature for this experiment was around the temperature in the room (i.e., 22.8 degrees Celsius), we pick a scale between 22 and 25 degrees Celsius (the final value as we said before was around 24.8 degrees Celsius). In the movie (that runs only for a couple of seconds), we can see at the beginning that the experiment starts struggling to arrive to

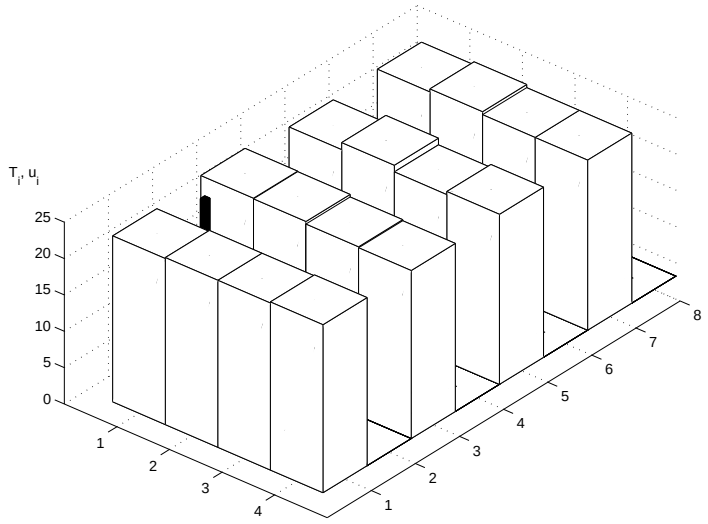

Figure 4.3: Lamps and sensor temperatures. This experiment was done with a 22.8 degrees Celsius temperature in the room.

a final value. But if we run the movie for more than 50 seconds, we could see how around 200 seconds, it starts having the same temperature, and the grid of temperatures is more uniform at the end. This can be seen in Figure 4.2, since at the end the temperatures arrive to a similar value. Another thing that we can see here, is the significant interactions between different zones. This is evident when we see how if one lamp is on, it affects some of the neighbors. We have to take into account also that we have some calibration issues, plus the disturbances that are random, and we cannot predict how these will affect all the lamps.

4.1.2 Effect of Delays for a Centralized Resource Allocation Strategy: Heat Minimum Temperature Zone

The control strategy developed in this case is practically the same as the one that we developed in Section 4.1.1. The main difference is that instead of having a pulse of 100 ms on the lamp whose sensor has the lowest temperature, it has to wait some time. Hence, after

a pulse is applied we have a delay till $u_j(k)$ values are chosen according to Equation 4.1. This time is fixed by the user in the dSPACE GUI. In this case, we can go from delays of 0.01 seconds to whatever the user wants. To show the differences with the results obtained in Section 4.1.1, we decided to run the experiment with a delay of 5 seconds, and with 0.5 seconds. The problem will always be the room temperature, that is varying due to a variety of factors. Figure 4.4 shows the temperatures sensed by all the sensors in the grid with a delay of 5 seconds between two consecutive pulses (as we can see in Figure 4.5), and Figure 4.6 does the same, but with a delay of 0.5 seconds (the delay can be seen in the control input u_i for the lamps in Figure 4.7). Comparing the figures with the delays, we can see that the temperature for the case where the delay is small is higher than the other one. Besides we can see that in the case where the delay was 5 seconds, the quality of temperature uniformity on the grid is not as good as in the case where the delay is small. This is consistent with resource allocation ideas. If we start increasing the value of the delay, we will obtain a result practically similar to the one in Figure 4.4 since with the conditions that we had in the ambient, it is practically impossible that the temperature increases or decreases more.

Again, to show the difference in the way that the system behaves, we add a movie. These movies, that can be accessed if we click on Figures 4.8 and 4.9, have the same axis as the one that we used in Section 4.1.1. Here, we start with a temperature in the room of 22.8 degrees Celsius, that varied during the experiment. We can notice in these two movies how the pulses are applied to the zone that has the minimum temperature not always as before, but every 5 or 0.5 seconds, depending on how long we defined the delay. Also, even though we did not run the movie for too long, we notice that we have more uniformity with an experiment that does not have delays, compared with these two experiments. Another important aspect that we can see in Figures 4.8 and 4.9 is that the temperature is higher for the case when the

delay is 0.5 seconds. That is what we expected since we do not allow a lot of time without a pulse in a zone that decreases its temperature. Besides, if we start increasing the delay, we will not see a significant difference with a delay higher than 7 or even 8 seconds if the temperature conditions are the same. That is due to the fact that the experiment never reaches a high temperature, and so the T_i will remain near the ambient temperature of the room.

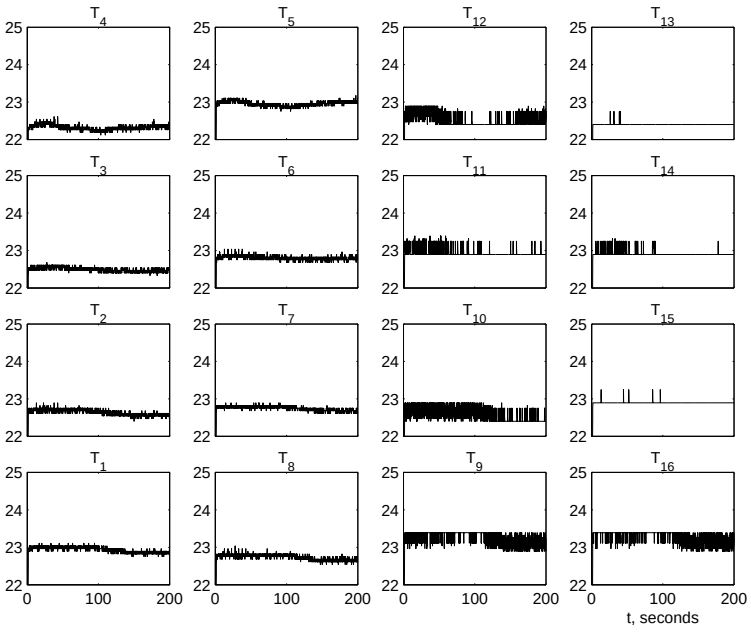

Figure 4.4: Temperature in all the zones when a delay of 5 seconds is applied between pulses.

We can use another type of delay. Before, we stored the data all the time, but the pulse appeared only every 5 or 0.5 seconds. This variation consists on store the data in the variables using a transport delay that has the same delay as the pulse that we want to apply. The result is practically the same as before, i.e., we have less uniformity, and the final temperature is practically the same ambient temperature. For example, we can use a delay of 3 seconds, an we obtain the results that we can see in Figures 4.10 and 4.11.

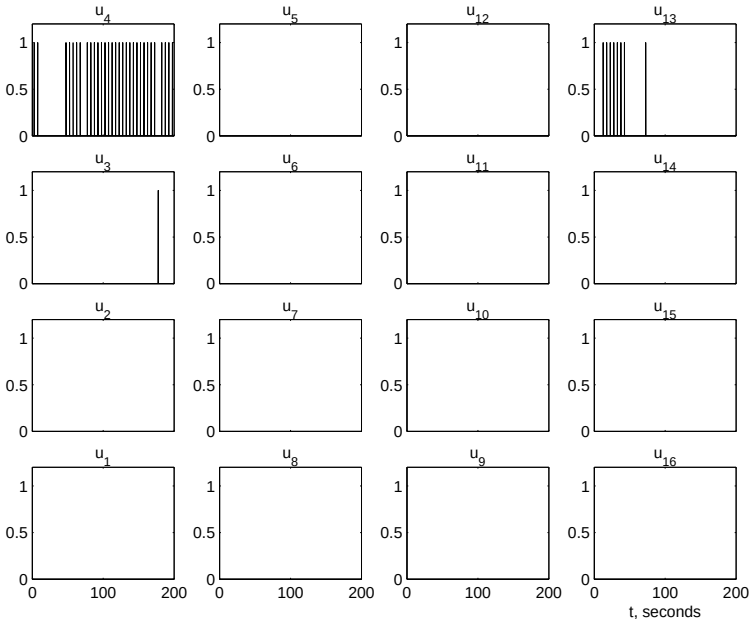

Figure 4.5: Control output for a delay of 5 seconds.

Here, as we can see after 500 seconds of simulation, the final temperature value oscillates in practically all the zones between 22 and 22.8 degrees Celsius. This value is practically the same temperature in the ambient (22.2 degrees Celsius measured with the Fluke). Then as we can see, the behavior of the experiment with some delays is not that good, since it has less uniformity in the grid. Figure 4.12 has an embedded movie that can be accessed by clicking in the figure, and has the same axis as the one that we used in Section 4.1.1. The movie shows how the pulses appear every 3 seconds, and in this case we can see that at the pulse is not necessarily applied to the zone that has the lowest temperature. That was what we expected, since as we could see before, the results with putting a pulse every x seconds (for $x > 1$ in almost all the cases) gave us something similar to the ambient temperature. In this case we even though the pulse could be applied to the wrong zone, the results are similar to those obtained before.

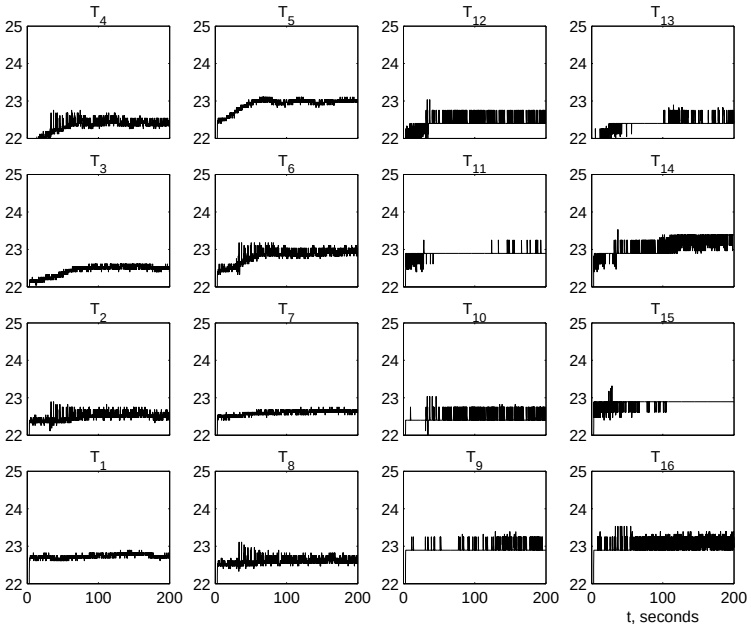

Figure 4.6: Temperature in all the zones when a delay of 0.5 seconds is applied between pulses.

4.1.3 Centralized Resource Allocation Strategy: Heat Any Zone Less Than Average Temperature Zone

Now, instead of heating based on the minimum temperature of all the zones, of each time instant we measure the temperature in all the zones, compute the average of these temperatures which we denote by T_{avg}, and those lamps associated with the sensors that have a temperature below T_{avg} are turned on. The lamp will be on if the error between the average temperature and T_i is positive. Otherwise, the lamp is off. In other words of each time k the lamp in zone i has

$$u_i(k) = \begin{cases} 1 & T_{avg} - T_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where T_{avg} is defined as

$$T_{avg} = \frac{1}{16} \sum_{i=1}^{16} T_i$$

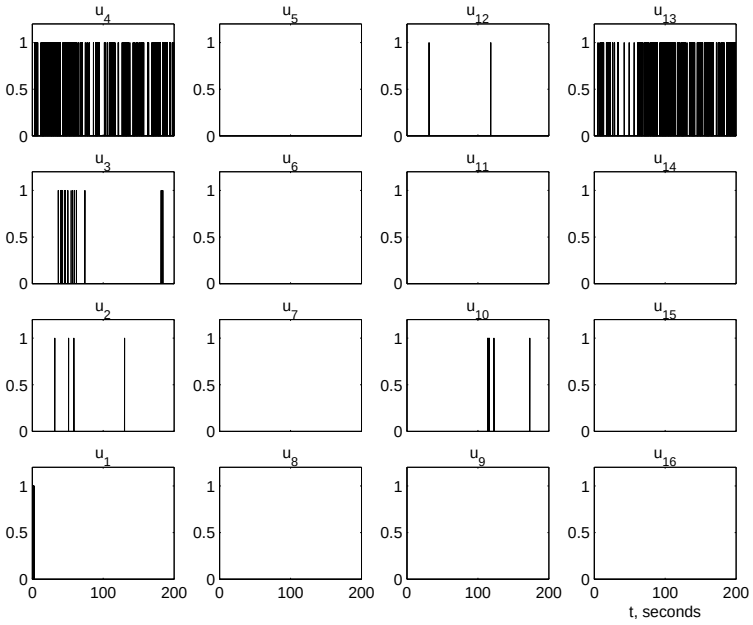

Figure 4.7: Control output for a delay of 0.5 seconds.

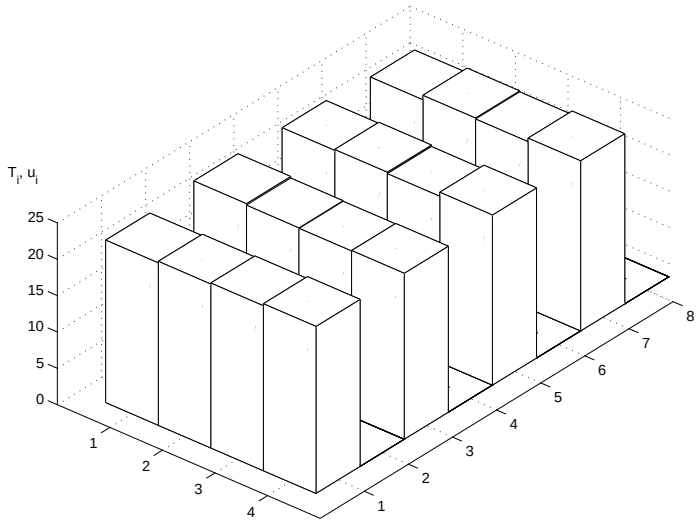

Figure 4.8: Lamps on in a centralized resource allocation experiment with a delay of 5 seconds. This experiment was done with a 22.8 degrees Celsius temperature in the room.

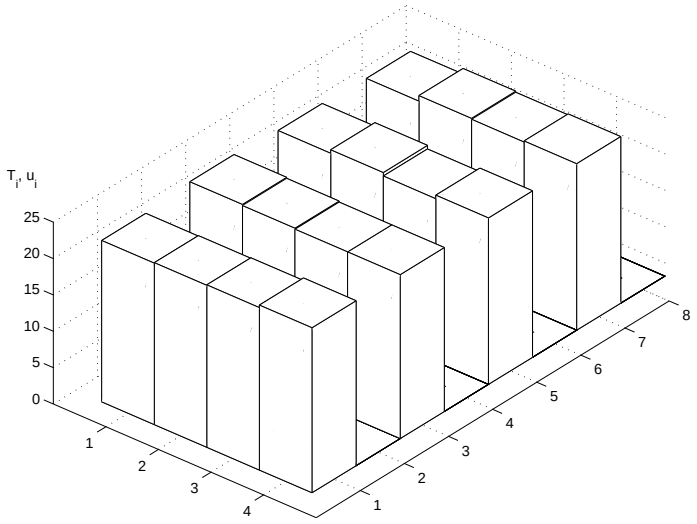

Figure 4.9: Lamps on in a centralized resource allocation experiment with a delay of 0.5 seconds. This experiment was done with a 22.8 degrees Celsius temperature in the room.

Using this strategy we expect to reach a higher final temperature value compared to the strategy of the last section since now we will typically have more than one lamp on. Intuitively T_{avg} will change over time and at the end we will have around eight of the lamps on. Since the influence between zones is significant, the temperature is going to be quite high in the middle since this is where we have more influence from all the zones. This can be seen in Figure 4.13, where the final temperature is around 38 degrees Celsius (which was the *max_temp* defined at the beginning of all the experiments). Some of the lamps are on practically all the time as we can see in Figure 4.14. These lamps are mainly physically located in the external edges of the experiment. That makes sense since the temperature of the room varied during the experiment a couple of degrees. In Figure 4.15 we have embedded a movie that shows for a couple of seconds what happened during the experiment, and can be accessed by clicking in the figure with the same axis as the one that we used in Section 4.1.1 If we click on the figure, we see the temperatures start to increase their values,

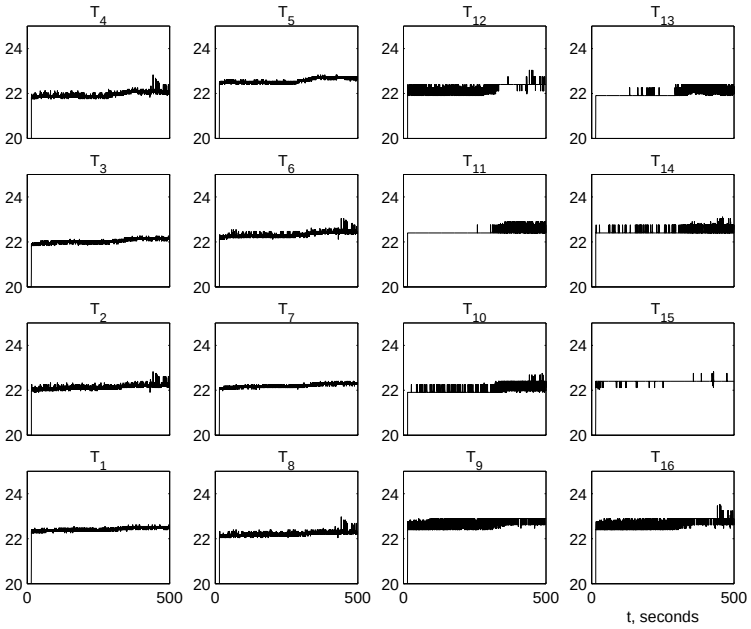

Figure 4.10: Temperature in all the zones when a delay of 3 seconds is applied between pulses using a transport delay of 3 seconds to store the data.

and each zone tries to follow the T_{avg} defined each 100 ms. With that, most of the zones reach a high value, meanwhile others try to increase the temperature value in their zone, but the disturbances are quite high, so their values remain below the average and hence the lamps try to compensate for the disturbance.

4.2 Decentralized Resource Allocation

4.2.1 Control Strategies

First, before we start developing any control strategy, we need to define a "sensing topology." This topology will be used to define how we will put a logical 1 into the control input u_i for $i = 1, 2, \ldots, 16$. To define the topology that we used in this experiment, we first analyzed the influences of each of the lamps on all of the sensors. The experiment that we used was to apply $u_i = 1$ to only one control input at a time, and see how the other sensors behave. In

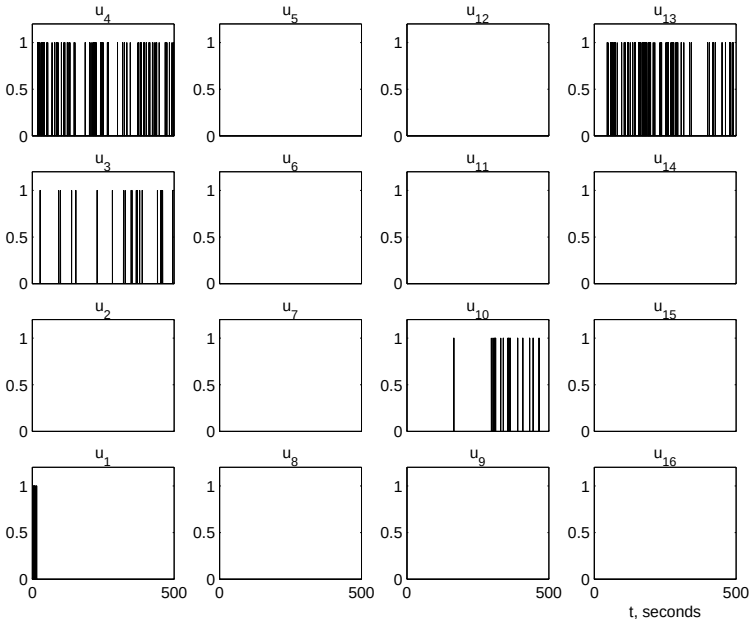

Figure 4.11: Control output for a delay of 3 seconds using a transport delay of 3 seconds to store the data.

Figures 4.16, 4.17 and 4.18 we did this for 200 seconds to u_1, u_2, and u_{11}, respectively. Here we can see the significant influence that one lamp has on its neighbors. We show these three figures to show the three cases that we have on the grid. First we have a corner zone case. We find this case for T_1, T_4, T_{13}, and T_{16}. Here we will have the influence of three neighbors plus the influence that the zone has on its own sensor. The next case, that we call the edge zone case, is repeated eight times. The zones are T_2, T_3, T_5, T_8, T_9, T_{12}, T_{15}, and T_{16}. Here, instead of having three neighbors that have a significant influence on the zone, we have 5. Finally, the last four zones are the middle ones, T_7, T_6, T_{10}, and T_{11} and these are affected by 8 neighbors. A diagram that illustrates how the zones are affected by their neighbors can be found in Figure 4.19. It is these arrows that we will use to define which inputs the controller for each zone can use. In this way we have used a "disturbance analysis" to define

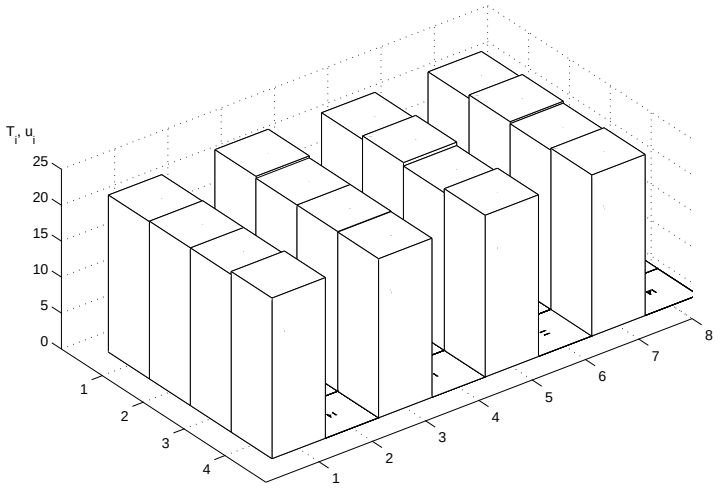

Figure 4.12: Lamps on in a centralized resource allocation experiment with a delay of 3 seconds, but storing the data also after 3 seconds. This experiment was done with a 22.2 degrees Celsius temperature in the room.

the topology. This approach makes sense since we will then have the local controllers make decisions based on the zones whose temperature they affect the most.

Let $N(i)$ denote the "neighbors" of zone i that the controller for zone i can sense temperatures in, including zone i. It is defined via Figure 4.19 (e.g. $N(13) = \{11, 12, 13, 14\}$ and $N(5) = \{3, 4, 5, 6, 11, 12\}$). Let for all i, at each time k

$$T_i^{min}(k) = min\ \{T_j(k) \mid j \in N(i)\} \tag{4.3}$$

and

$$u_i(k) = \begin{cases} 1 & T_i(k) = T_i^{min}(k) \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

If we run the experiment with an ambient temperature of 21.8 degrees Celsius, we obtain the results that are shown in Figures 4.20, 4.21, and 4.22. Figures 4.20 and 4.21 show how the control input works for each of the zones, and how the temperature changes in each of

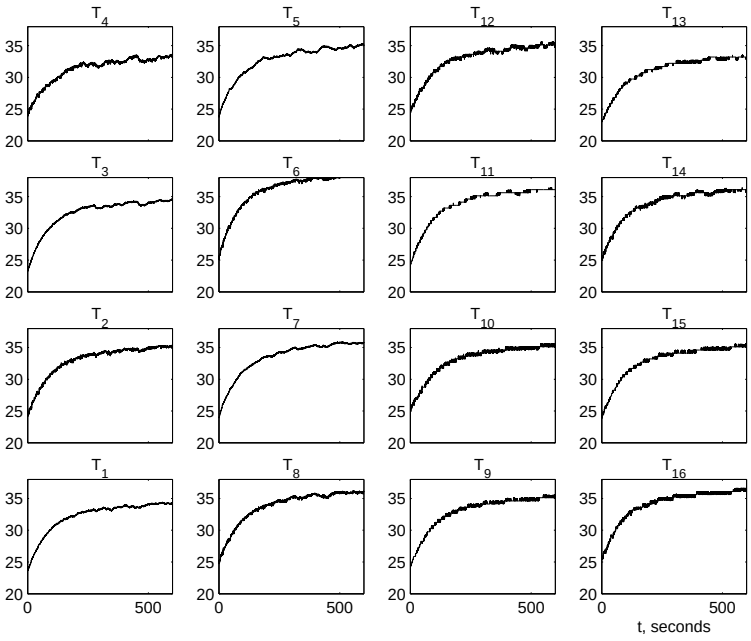

Figure 4.13: Temperature in all the zones when we heat any zone less than the average, with a temperature in the room of 23.0 degrees Celsius.

the zones respectively. After 200 seconds, the temperature in some of the zones is practically the same (between 26 and 27 degrees Celsius). There are other zones that are on the edges that cannot reach this value because of the large disturbances that we have discussed before. On the other hand, there are a couple of zones in the middle of the grid that have a large value of temperature (e.g., T_6) because the lamps that are in the edges are on practically all the time. Figure 4.22 has a movie embedded. If we click in the figure (that has the same axis as the one that we used in Section 4.1.1), we will see how more than one lamp could be on at a time, and how the temperature starts increasing in all the zones. However, there are some zones that do not increase the temperature as much as other zones, due to the large influence of the disturbances of the room that prevent the temperature from increasing as much as in other zones. One important characteristic for this case is that any lamp associated with a zone is not all the time on. That allows the experiment to reach a

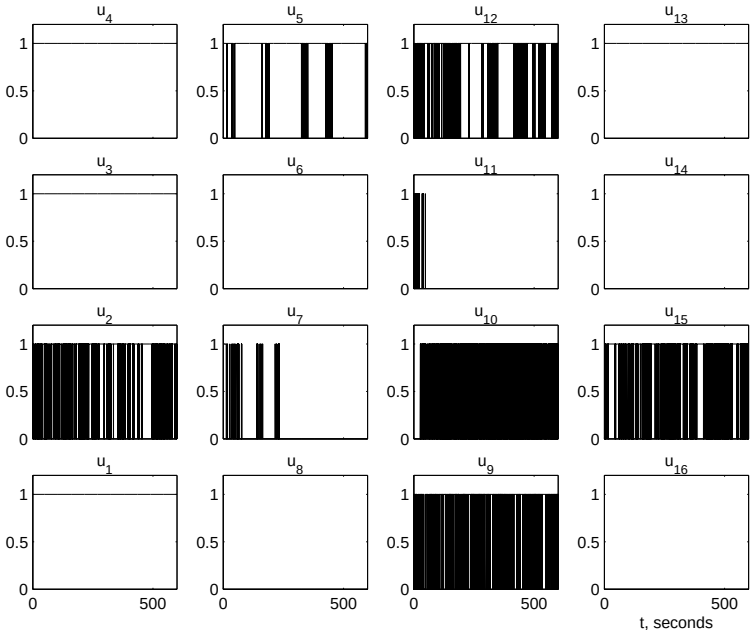

Figure 4.14: Control output when we heat any zone less than the average, with a temperature in the room of 23.0 degrees Celsius.

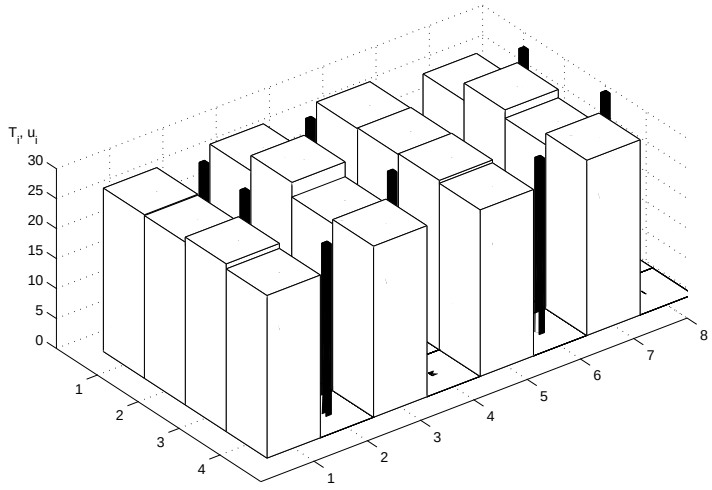

Figure 4.15: Control output and temperature value when we heat any zone less than the average, with a temperature in the room of 23.0 degrees Celsius.

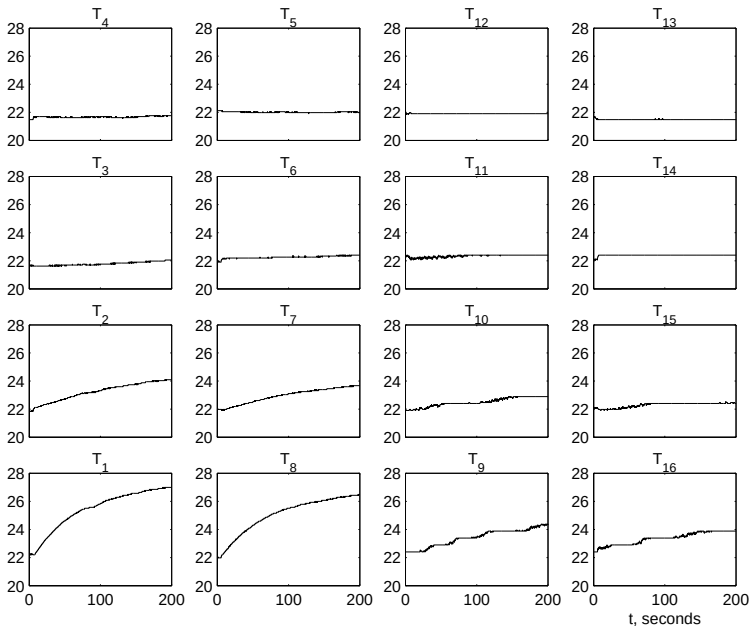

Figure 4.16: Behavior of all the sensors when $u_1 = 1$ if the temperature in the room is 21.4 degrees Celsius.

high temperature in the zone, but makes the experiment more uniform than the experiment that used the policy in Equation 4.2, showed before for the centralized case. In this case we cannot forecast what is going to be the maximum temperature reached by the grid, since we have some external factors that affect considerably the performance of the experiment. However, we knew originally that the maximum temperature must be higher that the final temperature in the centralized case, since we had more than one lamp on at a time.

Another modification to the same decentralized experiment is to find an average per zone T_i^a, and based on the topology defined before, we make control decisions. For instance, we find

$$T_i^a(k) = \frac{1}{|N(i)|} \sum_{j \in N(i)} T_j(k) \tag{4.5}$$

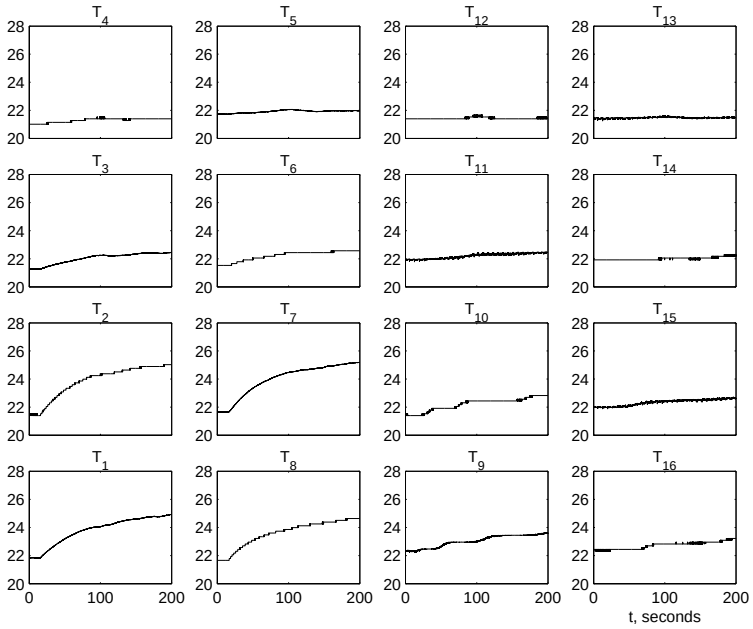

Figure 4.17: Behavior of all the sensors when $u_2 = 1$ if the temperature in the room is 21.4 degrees Celsius.

and let

$$u_i(k) = \begin{cases} 1 & T_i(k) \leq T_i^a(k) \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

If we run the experiment, we will see that the final temperature is quite high, and some of the lamps are on practically all the time. For this case we find the same problem that we had before, when in the centralized case we tried to compute the average and turn on the lamp associated with the zone that was below the average. The temperature in the middle is quite high because of the influence of the lamps on the edges, and those zones are on practically all the time, since the disturbance affects those more.

4.2.2 Effect of Delays

To study the effects of delays in the decentralized experiment, we modify the first control strategy studied in the last subsection. Now, instead of storing the data directly from the

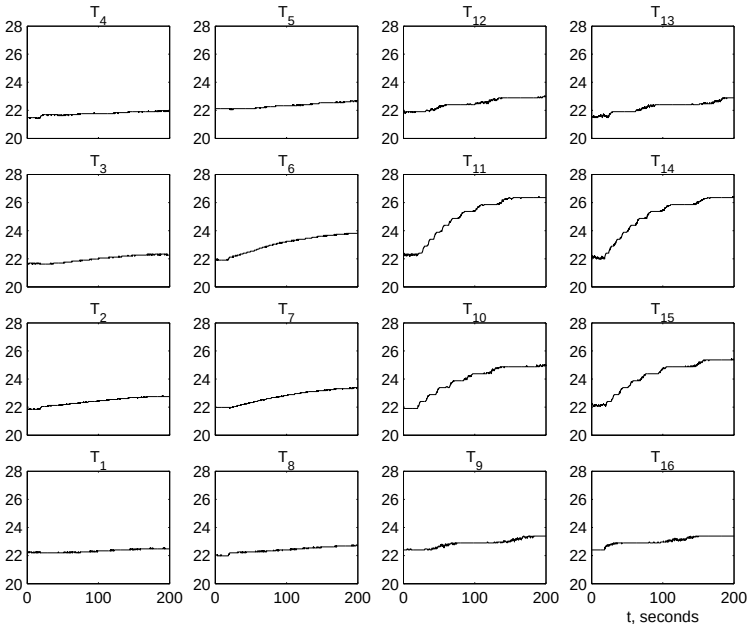

Figure 4.18: Behavior of all the sensors when $u_{11} = 1$ if the temperature in the room is 21.4 degrees Celsius.

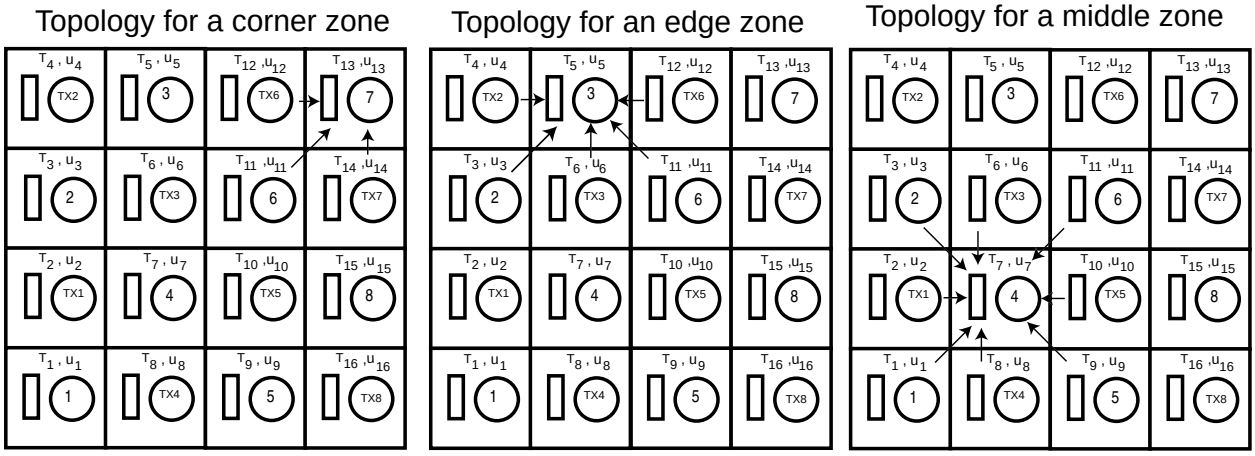

Figure 4.19: Topology for the decentralized control experiment.

data acquisition, we delay the value acquired using a transport delay. For this case we fixed the value of the delay, and we use the *min_temp* value as the initial condition for the delay

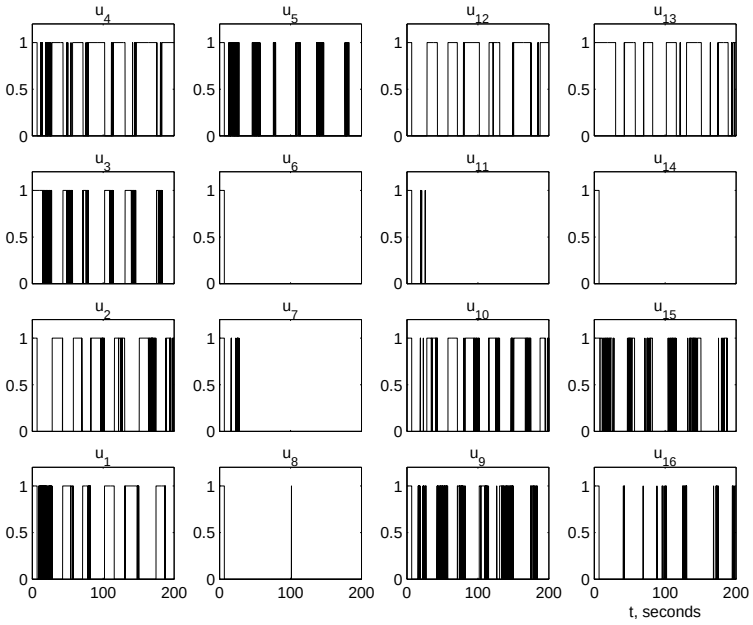

Figure 4.20: Control output when we use the topology defined for a decentralized experiment, with a temperature in the room of 21.8 degrees Celsius.

values. To show the differences between what we got for the normal case, we run a couple of experiments, and we took the one that has 40 seconds of delay. Here, as we can see in Figure 4.23, even though the temperature values are not reaching steady state, it has a better performance than the average strategy. The final value is not very high, and the delay results in the controllers turning on some lights (as we can see in Figure 4.24) even though they should not. But, the performance in this case is not as good as the one that we got for the case where we have not delays. In that case, the final temperature value was practically the same in all the zones as the one that we got here (taking into account the ambient temperature), but the difference is that the temperature in the zones did not have so many oscillations. In Figure 4.25 we have an embedded movie (with the same axis as the one that we used in Section 4.1.1), where we can see at the beginning all the lamps on, since it assumes that the minimum is already in that zone. The movie, that runs for a couple

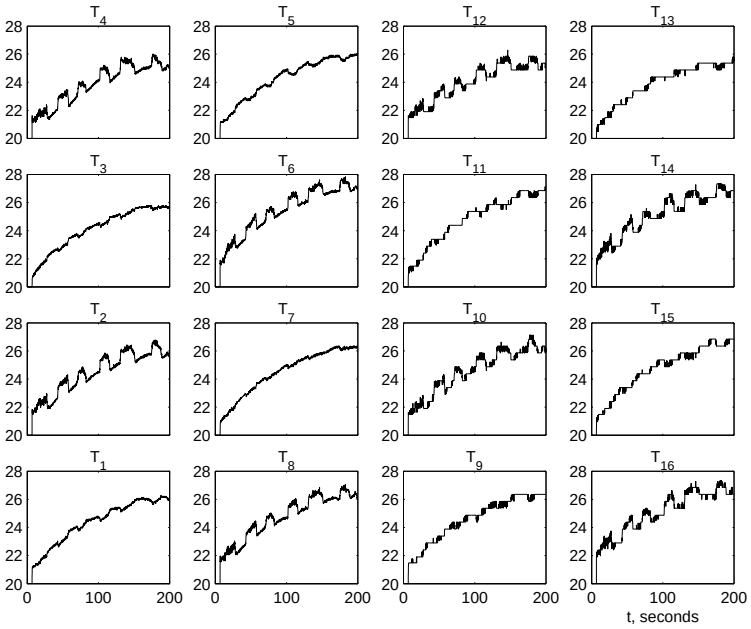

Figure 4.21: Temperature in all the zones when we use the topology defined for a decentralized experiment, with a temperature in the room of 21.8 degrees Celsius.

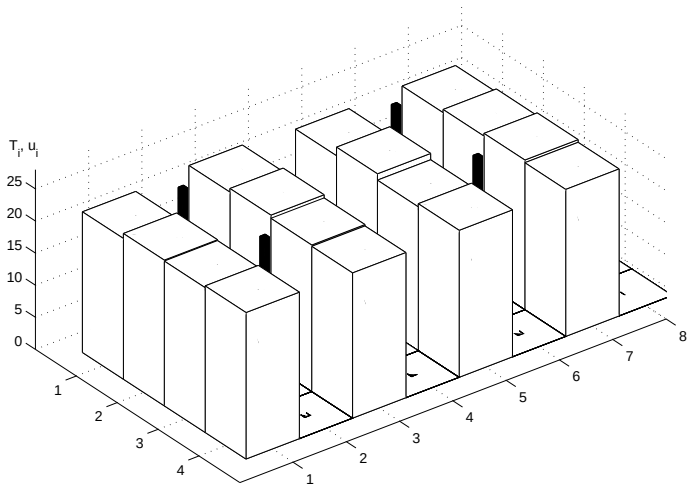

Figure 4.22: Control output and temperature value when we use the topology defined for a decentralized experiment, with a temperature in the room of 21.8 degrees Celsius.

of seconds, shows how even when the temperature starts increasing its value in each of the zones, the control input u_i, $i = 1, 2, \ldots, 16$ does not change its status from 1 to 0. Only after 40 seconds does that happen, but as we can see, the control inputs that have a 1 might not be the lamps that should be on, because the data used in decision making were acquired 40 seconds before the decision. In Figure 4.26 we can see how the controller takes a wrong decision, because the value that it uses to make a decision is an old value. For instance, the temperature that is making in the decision process at time 80 seconds for T_9 is 31 degrees, instead of the real value that is close to 28 degrees Celsius. So the controller makes a wrong decision, and so the lamps could be on, instead of being off since the value is 40 seconds delayed.

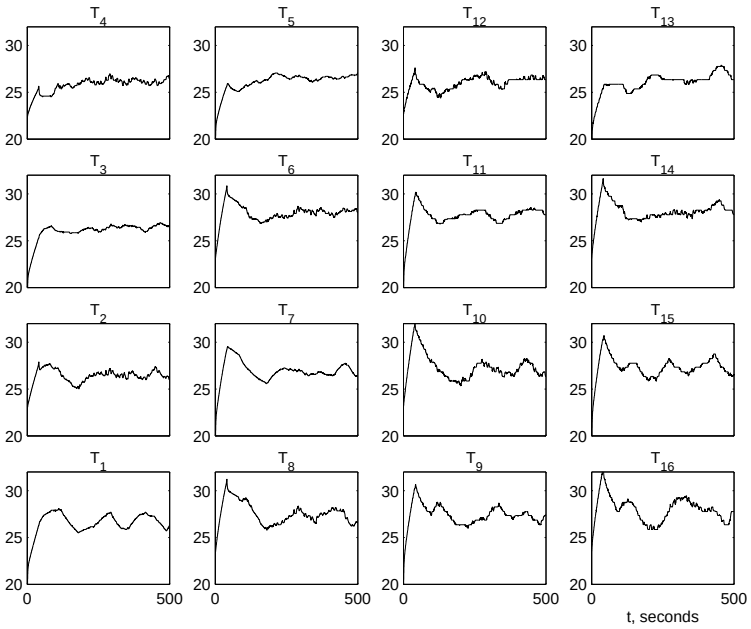

Figure 4.23: Temperature in all the zones when we use the topology defined for a decentralized experiment, with a temperature in the room of 21.8 degrees Celsius and a delay of 40 seconds.

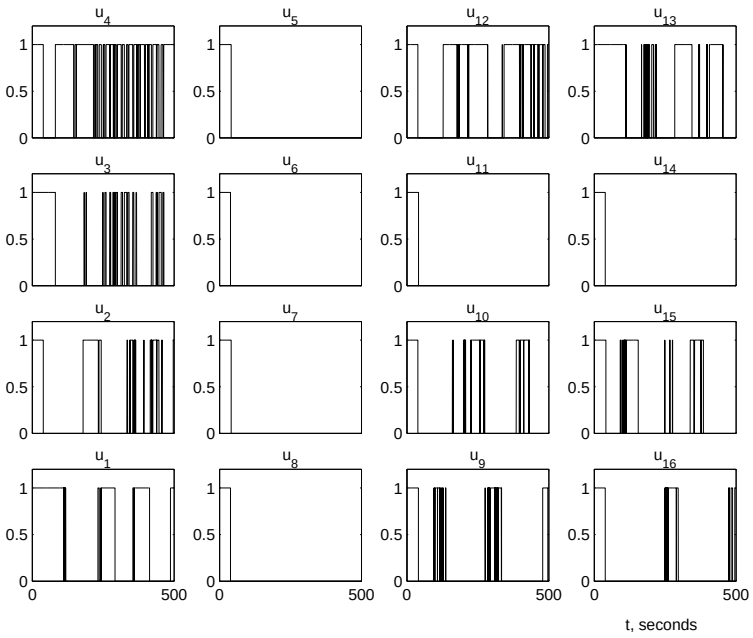

Figure 4.24: Control output when we use the topology defined for a decentralized experiment, with a temperature in the room of 21.8 degrees Celsius and a delay of 40 seconds.

4.3 Conclusions

If we compare some of these experiments, we can say that in terms of achieving good temperature uniformity the centralized one, where we heat the minimum temperature zone is the best. There, even though the final temperature was not very high, the uniformity of the grid was high, and essentially after a couple of minutes we had the zones near a steady state value (24 degrees Celsius for the experiment that was presented). The other centralized and decentralized experiments had a lot of troubles with the disturbances and the influences between zones. When we added a delay in the centralized experiment, we found that the grid lost a lot of its temperature uniformity when we increased the value of the delay. In the decentralized case, we reached a higher temperature since we allowed more than one lamp on at a time (the same thing happens for the centralized case when we heat a zone less than

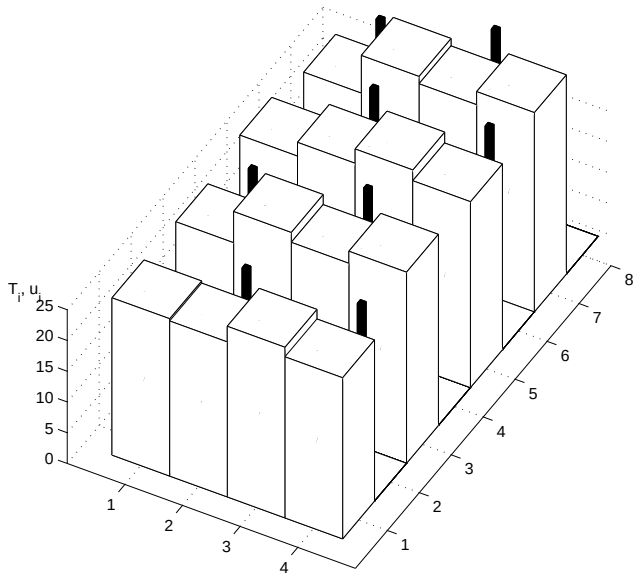

Figure 4.25: Control output and temperature value when we use the topology defined for a decentralized experiment, with a temperature in the room of 21.8 degrees Celsius and a delay of 40 seconds.

the average). The problem for these cases was the lack of temperature uniformity of the grid. Besides, we had the problem of the *max_temp* that limited the maximum value that we can reach in the experiment (due to the number of bytes that we can transmit using an RS-232 connection).

For the superzone temperature tracking, we can make some comparisons with LabVIEW because the experiment that was done before had most of the characteristics of this one. The first thing that we can say is that in LabVIEW there are some parts of the program that are easy to code. The problem is that since Simulink was not developed originally to work as a data acquisition system, some of the blocks have characteristics that we cannot change, and we would like to. Besides, the dSPACE GUI is not as well developed (e.g. it has fewer interface options) as the front panel in LabVIEW. On the other hand, the real-time

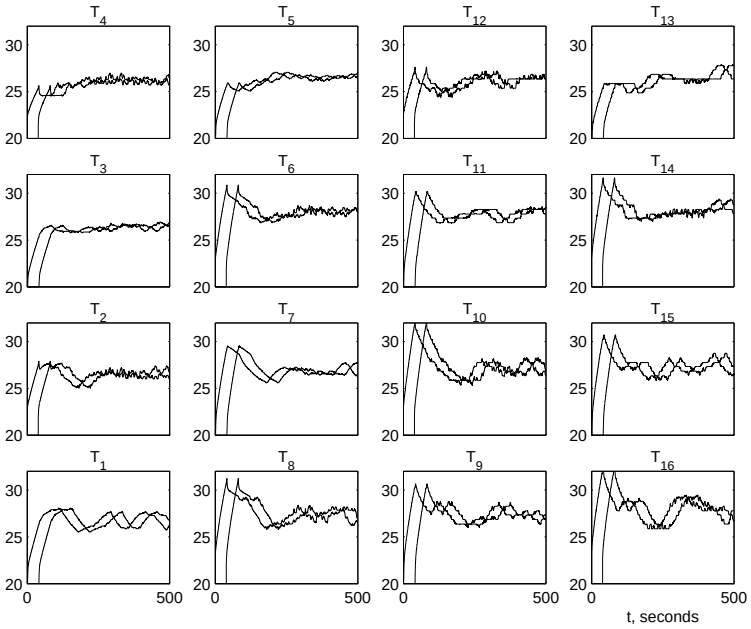

Figure 4.26: Temperature in all the zones (delayed and non delayed by 40 seconds) when we use the topology defined for a decentralized experiment, with a temperature in the room of 21.8 degrees Celsius.

card that we used in LabVIEW had a lot of troubles that we did not have with the DS1104 dSPACE card. Even though we had practically the same analog inputs (but more analog outputs and DIO), we knew the exact sampling time (100 ms).

CHAPTER 5

CONCLUSION

5.1 Summary

The thesis consisted of three main experiments. First of all, we developed some of the NIST RCS methodology in LabVIEW. Then, using the same concepts, but without having a hierarchical controller (only a master/slave configuration), we developed a dSPACE based experiment. This experiment used four "superzones," each of them consisting of four heaters and four sensors. The experiment had practically the same characteristics as the LabVIEW one, but it was more complex due to the number of elements that were in one loop, and the large amount of disturbances that we had. Finally, using the same hardware as the second experiment, we developed different resource allocation algorithms. First, we used the centralized case using 16 different zones. Two main experiments were developed, one where you turn on only the heater which zone has the minimum temperature value, and another one, based on the average of all zones. For the first case we also studied the influences of different types of delays. The next resource allocation strategy was developed for the decentralized case. Here, we used a "network topology" to control each zone, depending on the neighbor's information. We also studied the impact of delays. For these dSPACE based experiments, we found large amount of disturbances from the wind and the ambient temperature to the calibration of each sensors.

5.2 Contributions

During the thesis process, most of the contributions were

1. We developed a new distributed experiment, where we can study different kind of algorithms, with a large amount of disturbances. The great advantage that this experiment has is the price (the experiment is not very expensive), and that we can increase the amount of zones. The problem in this case will be the number of DS1104 dSPACE interface cards to acquire more data.

2. Most of the RCS methodology could be implemented in LabVIEW and some of it in dSPACE. That will help to make everything standard, and using the great capabilities that LabVIEW has, we can develop more sophisticated experiments, without losing any generality.

3. Some of the resource allocations algorithms could be implemented using the distributed experiment. The study of the algorithms and their significant influence on performance contributed to our general understanding of dynamics of complex control systems.

4. Finally note that this thesis work was on the development of the dSPACE based "Control Systems Implementation Laboratory" (i.e., EE758 at The Ohio State University), in Spring Quarter 2002 (see `http://eewww.eng.ohio-state.edu/~passino/ee758.html`).In the future it is expected that the experiments described here will become part of the regular laboratory experiments conducted in this educational laboratory.

5.3 Future Directions

In the future, our work should solve the following issues:

1. Generalization of the RCS-LabVIEW control module and communications. Specifically, it needs to be extended to handle flexible type of messages. Now we have a cluster that has a size that should be defined by the user at the beginning. But the user cannot send a message that has less elements than the ones that he/she defined.

2. Assess the capabilities of the TCP/IP or the DataSocket approach to communications relative to NML communications.

3. Investigative the feasibility of message-based communication to more faithfully implement RCS-style communications and to ensure scalability up to large applications.

4. Design and implementation of additional decentralized control experiments to develop and exercise RCS-LabVIEW. Some candidates include balancing balls in tubes, that is an experiment where we try to allocate air pressure optimally to keep all balls at a uniform height but maximally elevated. Can study influence of delays/network effects.

   We are also evaluating the benefits of implementing a variation on the temperature grid experiment studied here that would use an analog demultiplexer chip to provide an inexpensive way to use many analog inputs to provide for continuous inputs to each of the 16 zones.

5. Development of remote internet-based interfaces for both monitoring and controlling an experiment.

6. For dSPACE, we have to be able to solve the TCP/IP communication to have something similar to the LabVIEW experiment. Right now for dSPACE we cannot implement a hierarchy like the one developed in LabVIEW, but we can communicate (with constraints in the number of bytes) some data via the RS-232 between two computers (the RS-486 communication link on the DS1104 dSPACE card would allow more than

two cards but not remote connection). In this aspect, LabVIEW has a more reliable communication and easier to use communications, and even though it is really difficult to program in LabVIEW, we know that the data is going to arrive as we expected. Perhaps the TCP/IP communication software we recently received from dSPACE will help it achieve some of the nice remote communication capabilities, and to implement a true hierarchical and decentralized controller.

BIBLIOGRAPHY

[1] V. Gazi, M. L. Moore, K. M. Passino, W. P. Shackleford, F. M. Proctor, and J. S. Albus, *The RCS Handbook: Tools for Real Time Control Systems Software Development*. NY: John Wiley and Sons, 2001.

[2] R. H. Bishop, *Learning with LabVIEW 6i*. NJ: Prentice Hall, 2001.

[3] J. Travis, *Internet Applications in LabVIEW*. NJ: Prentice Hall, 2000.

[4] C. D.Schaper, K. El-Awady, and A. Tay, "Spatially-programmable temperature control and measurement for chemically amplified photoresist processing," *SPIE Conference on Process, Equipment, and Materials Control in Integrated Circuit Manufacturing V*, vol. 3882, pp. 74–79, September 1999.

[5] C. D. Schaper, T. Kailath, and Y. J. Lee, "Decentralized control of wafer temperature for multizone rapid thermal processing systems.," *IEEE Transactions on Semiconductor Manufacturing*, vol. 12, pp. 193–199, May 1999.

[6] M. Zaheer-uddin, R. V. Patel, and S. A. K. Al-Assadi, "Design of decentralized robust controllers for multizone space heating systems.," *IEEE Transactions on Control Systems Technology*, vol. 1, pp. 246–261, December 1993.

[7] D. Bertsekas and R. Gallager, *Data Networks*. NJ: Prentice Hall, 2nd ed., 1992.

[8] D. E. Comer, *Computer Networks And Internets*. NJ: Prentice Hall, 2nd ed., 1996.

[9] A. S. Tanenbaum, *Computer Networks*. NJ: Prentice Hall, 3rd ed., 1996.