ABSTRACT

Development and Implementation of a Multi-Agent Control System
for the Rankine Cycler Laboratory Teaching System

Jason Gomes, M.S.E.C.E.

Mentor: Ian A. Gravagne, Ph.D.

One challenge faced by the engineering discipline in the near future is increased electric power demand coupled with greater efficiency requirements for power generation stations. Development of more robust control systems capable of supporting larger power plants with more adaptable control schemes is one solution to the problem. A Multi-Agent Control System is a type of coordinated control scheme capable of multiple parallel operations using several elements, or agents, communicating over a network. This thesis discusses the development and implementation of a Multi-Agent Control System on a small laboratory power generation plant.

Development and Implementation of a Multi-Agent Control System
for the RankineCycler Laboratory Teaching System

by

Jason Gomes, B.S.E.C.E.

A Thesis

Approved by the Department of Electrical and Computer Engineering

_____
Kwang Y. Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science in Electrical and Computer Engineering

Approved by the Thesis Committee

_____
Ian A. Gravagne, Ph.D., Chairperson

_____
Kwang Y. Lee, Ph.D.

_____
Kenneth W. Van Treuren, Ph.D.

Accepted by the Graduate School
August 2012

_____
J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

Copyright © 2012 by Jason Gomes

All rights reserved

TABLE OF CONTENTS

LIST OF FIGURES v

LIST OF TABLES vii

LIST OF ABBREVIATIONS viii

ACKNOWLEDGMENTS x

CHAPTER ONE Introduction
1.1 Background 1
1.2 Power Plant Control Basics 1
1.3 Research History and Objectives 5
1.4 System Overview 6
1.5 Chapter Descriptions 6

CHAPTER TWO System Hardware 7
2.1 Rankine Cycler 7
2.2 Modifications 12
2.2.1 Automation 12
2.2.2 Sensors 17
2.2.3 Water Level Sensor 19
2.2.4 Water Pump 19
2.2.5 Switches 23
2.2.6 Power Supply 23

CHAPTER THREE Project Software 25
3.1 Control Structure 25
3.2 Multi-Agent System Overview 27
3.3 MAS Communication 31
3.4 MATLAB Algorithms 33
3.4.1 Free Agent 34
3.4.2 Delegation Agent 34
3.4.3 Feedforward Agent 36
3.4.4 Feedback Agent 41
3.4.5 Interface Agent 44
3.5 LABviewAlgorithms 46
3.5.1 Acquisition and Control 47
3.5.2 Gauges 49
3.5.3 Motor Control 50

CHAPTER FOUR Results 53
4.1 Objectives 53
4.2 Load Demand Curves 53
4.3 Results 55
4.4 Neural Network Training Process 59
4.4.1 Neural Network Configuration 59
4.4.2 Data Hunter 62

CHAPTER FIVE Conclusion 69
5.1 *RankineCycler* Size 69
5.2 Feedwater System and Water Level Sensor 70
5.3 Points of Interest for Further Research 71
5.3.1 Control System Frequency 71
5.3.2 Gain Selection 73
5.3.3 JADE 74

APPENDIX A Motor Control Wiring 76

APPENDIX B LABView Software 77

APPENDIX C MATLAB Software 107

APPENDIX D Operating Instructions 191

LIST OF FIGURES

Figure 1. System Overview 6

Figure 2. *RankineCycler* 8

Figure 3. Rankine Cycle Diagram 9

Figure 4. *RankineCycler* Boiler 10

Figure 5. *RankineCycler* Turbine and Turbine Cutaway 11

Figure 6. *RankineCycler* Generator 12

Figure 7. *RankineCycler* Condenser Tower 13

Figure 8. Motor Control 14

Figure 9. Steam Valve motor (top). Fuel Valve Motor (bottom) 16

Figure 10. Load Potentiometer motor(top). Airflow knob motor (bottom) 17

Figure 11. Driver/Controller Mounting (top). Controller board closeup (bottom) 19

Figure 12. IOtech DAQ/56 21

Figure 13. Original Sight Glass (left). Sight Glass with Water level sensor (right). 23

Figure 14. Water lever sensor bar (top). Unmounted Water level sensor board(bottom). 24

Figure 15. Pump and flow sensor 25

Figure 16. Power Supply 26

Figure 17. Control Structure. 28

Figure 18. Reference Governor Diagram 29

Figure 19. Agent Architecture. 31

Figure 20. Agent Task Hierarchy 31

Figure 21. MAS message field structure 34

Figure 22. Free agent code flowchart 38

Figure 23. FeedForward Agent Code Flowchart 40

Figure 24. Starting point of the Particle Swarm (top). After 100 iterations (bottom) 43

Figure 25. Feedback Agent code Flowchart 45

Figure 26. Interface agent GUI 48

Figure 27. Acquisition and Control Front panel 50

Figure 28. Gauges VI front panel 53

Figure 29. Motor Control front panel 54

Figure 30. Target Demand Curve 58

Figure 31. Power vs. Power demand Results 60

Figure 32. Voltage and load resistance during test 61

Figure 33. Fuel and Steam Valve positions. 62

Figure 34. Pressure and Pressure Demand. 62

Figure 35. Layer Recurrent Network 64

Figure 36. Feedforward network structure 65

Figure 37. Simulated Steady State Evaluation 66

Figure 38. DataHunter Flowchart 68

Figure 39. Data Hunter GUI 70

Figure 40. DataHunter Sample Output 72

LIST OF TABLES

Table 1. Rankine Cycler Sensors 20

Table 2. Sensor Accuracy and Averaging 22

LIST OF ABBREVIATIONS

ACL	Agent Communication Language
AC	Alternating Current
β	Preferences vector
c_1	Particle swarm personal weight
c_2	Particle swarm swarm weight
DAQ	Data Acquisition module
DC	Direct Current
E	Output Power
E_d	Desired Power
E_{uld}	Desired Power-over-Time Vector
g_{best}	Swarm best position
HPSO	Hybrid Particle Swarm Optimization
J	Objective vector
K_d	Derivative gain matrix
K_i	Integral gain matrix
K_p	Proportional gain matrix
LED	Light Emitting Diode
MAS	Multi-Agent System
Ω_1	Fuel valve operating window
Ω_2	Steam valve operating window
P	Boiler Pressure
p_{best}	Particle best position

P_d	Desired Pressure
PSO	Particle Swarm Optimization
RCMAS	*RankineCycler* MAS implementation
SimMAS	Simulated MAS implementation
u_1	Fuel valve position
u_2	Steam valve position
u_1*	Fuel valve optimal position
u_2*	Steamvalve optimal position
v	Particle velocity vector
UDP	User Datagram Protocal
USB	Universal Serial Bus
VI	LABview Virtual Instrument
w	Particle swarm scaling value

ACKNOWLEDGEMENTS

There are several people who helped contribute to this thesis that deserve thanks. First is Dr. Ian Gravagne for being patient and helpful, along with aiding some of the modifications to the *RankineCycler*. Ryan Schroeder and Aaron Provost also helped with the modifications and the LABView software.

On the MAS side, Dr. Kwang Y. Lee deserves credit for managing the implementation team. Jason Head and Craig Williams also contributed significantly to the project.

CHAPTER ONE

Introduction

1.1 Background

Electric power demand has been increasing steadily as society becomes more reliant on technology. In the United States alone, electricity consumption per year has increased by over 300,000 GWh in the past 10 years [1]. This increasing demand has caused a need for larger, more complex power generation stations, along with more efficient power generation. Research on power plant control is important to meeting increased demand on the power grid.

Advances in computing and communication have allowed the use of groups of computers to work together on one task. A Multi-Agent System (MAS) is a method of coordinating computers and devices while maintaining individual element flexibility. This configuration is one solution to the problem of increased power plant complexity. A Multi-Agent control system would allow for more robust and efficient control of power plants along with giving the controller capabilities beyond traditional systems.

1.2 Power Plant Control Basics

The vast majority of power generation is done by turbo alternators; a device where a turbine provides a mechanical input to a synchronous electric generator. Synchronous generators are the primary means of regulating the electric grid voltage to its nominal value. Thousands of these generators are interconnected to the grid and synchronized to match the standard voltage and frequency defined by the regulatory

agencies of the local grid [2]. Most generators are equipped with a voltage regulator that senses the bus voltage to be regulated and compares it with a desired value. This difference is used to control the turbine-generator system.

A power plant controller has two goals: maintain steady turbine rotation speed and maintain a target voltage across the load bus terminals. This is a challenge due to changing load characteristics on the output bus. Electric current flowing through the stator windings of the generator produces electromechanical torque that opposes the torque supplied by the turbine. If the goal of the electric grid is to maintain power at a specific voltage level, increased load translates to increases in the current flowing through the stator windings. This means that an increased load on the bus will make the turbine harder to rotate while maintaining nominal voltage levels. A power plant control system has three methods to compensate for changing currents in the generator; increasing steam flow into the turbine, increasing the pressure of the steam flowing into the turbine, or changing the strength of the electromagnetic field in the generator.

Power plant control schemes in use by the industry or being researched can be categorized in three groups: boiler-following, turbine-following, and coordinated control schemes. The differences between the types are apparent in the different configurations and emphasis of their control loops.

Historically, boiler-following control schemes were the first used in industry. Boiler-following control schemes consist of two coupled single input, single output control loops. The first operates around the turbine-generator system and is designed to throttle the steam input valve into the turbine and ensure the generator is producing the target power provided by the demand curve. It performs this function without regard to

pressure or temperature conditions in the boiler. A second control loop changes fuel and water input to the boiler to maintain a constant pressure. In this configuration the boiler "follows" the turbine, responding to conditions caused by changes in the turbine state. The main advantage of this configuration is rapid response time to load changes and inaccuracies in the predicted load demand. The main drawback is that the system is less stable because the boiler has a tendency to overshoot its targeted pressure due to the longer time scales required for the boiler system to respond to changes in inputs.

Turbine-following control schemes are similar to boiler-following methods, but differ in the order of dependency. With turbine-following methods, the load demand is converted to a desired pressure in the boiler and the control loop around the boiler subsystem is the primary method of control. It modulates the fuel and water inputs and the steam throttle valve output to achieve a desired pressure. The pressure may be constant or variable depending on the specific control system design. The turbine subsystem "follows" the boiler, because the steam throttle valve changes position based on conditions in the boiler subsystem. The advantages of this scheme are increased stability and minimal steam pressure and temperature fluctuations. Limited changes in steam conditions are desirable because operating in this manner extends the lifetime of the plant and reduces downtime. The disadvantages of turbine-following control schemes are much slower response times and the system does not take advantage of the energy storage capabilities of the boiler.

A third type of control scheme is coordinated control, which is any advanced control scheme that tries to maintain the strengths of both boiler-following and turbine-following schemes while minimizing the disadvantages. Specifically, most coordinated

control schemes operate similar to turbine-following methods but free the turbine subsystem in some manner to take advantage of energy storage in the boiler. Actual implementations of advanced coordinated control schemes are scarce. This is because they typically do not perform well under real-world requirements, particularly real-time performance and wide-range operating capacity. They also assume conditions rarely found in practice, like constant process parameters, correct and exact measurements, and precise mathematical models. Development of a precise model is difficult because of dimensionality and non-linearity of power plant dynamics. Despite these limitations on coordinated control schemes, they represent pinnacle of current power plant control research. The Multi-Agent System is an example of a coordinated control scheme.

Another important consideration in power plant control systems is the economic market for electricity production. Over time the market for electrical power generation has become more and more competitive due to several factors including newer power generation techniques like wind power and solar energy. The regulatory status of the industry has also changed, with more requirements on emissions and an emphasis on efficiency. With increasing competitive and regulatory pressures, there is demand for more comprehensive and flexible control methods.

The MAS is a control system that can meet the technical and economic challenges of the industry by utilizing:

- Artificial intelligence to allow for advanced coordinated control,
- A decentralized control scheme that allows for control elements to be located over a wide physical area, which is becoming more common with the prevalence of larger power plants,

- Structured system architecture that allows for a modular control scheme,
- Open design that allows for diverse functional and operational requirements to counter a highly variable market,
- A simple end-user realization.

1.3 Research History and Objectives

This project is the latest in a series of research projects originally started by Dr. Kwang Y. Lee at Pennsylvania State University. At Penn State the control algorithms were originally developed by doctoral students Raul Garduno-Ramirez [3] and Jin Heo [4]. The result of their research was a simulation of a MAS system, along with performance analysis of different variations on the algorithms. For example, the choice to use a hybrid particle swarm optimization scheme instead of other search algorithms was due to the research done at Penn State. However, the Penn State researchers did not develop a communications scheme between agents in the system, and their simulations do not take communication between agents into account.

The objective of this part of the research is to implement a Multi-Agent system on a small laboratory scale power plant. This is the first step in an effort to implement a Multi-Agent system on a larger power plant. The work reveals difficulties in the project that may be hidden in a computer simulation.

1.4 System Overview

The power plant used in this research is a modified version of the *RankineCycler* built by Turbine Technologies, Ltd., which is controlled by a MATLAB based Multi-Agent System. A laptop running LABview provides the interface between the MAS and the power plant. Communications between the laptop and the MAS are performed via

User Datagram Protocol (UDP) over the Baylor University local area network. The laptop is connected to the data acquisition (DAQ) and control boards of the *RankineCycler* via Universal Serial Bus (USB). A diagram of these connections is shown in Figure 1.

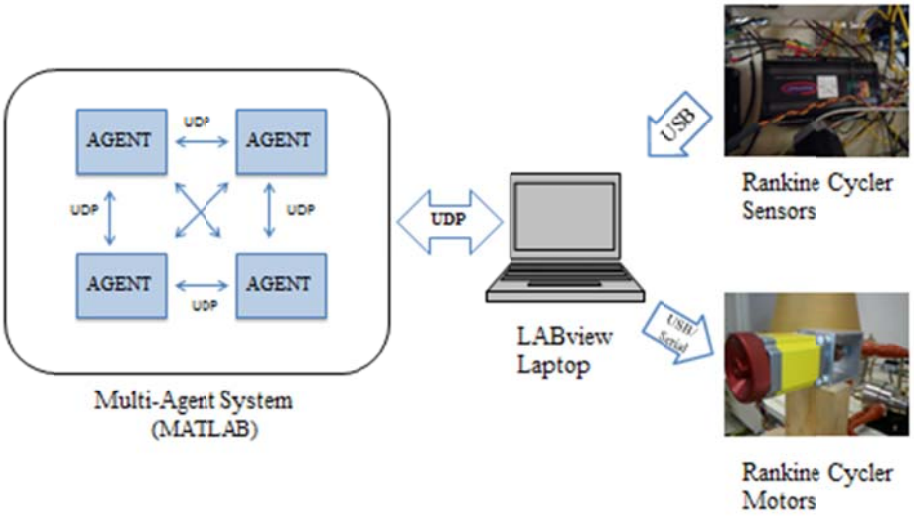


Figure 1. System Overview.

1.5 Chapter Descriptions

Chapter One is an introduction to the research and this document. It provides the goals of the research along with some background and chapter descriptions. Chapter Two elaborates on all the hardware changes made to the Rankine Cycler. Chapter Three describes the operation of the MATLAB based Multi-Agent System. Chapter Four provides results and data on the operation of the system. Chapter Five concludes the thesis.

CHAPTER TWO

The RankineCycler

2.1 Rankine Cycler

In order to measure the performance of a power plant control system, a power plant or power plant simulator is needed. In this case, a modified version of a Turbine Technologies, LTD *RankineCycler* is the test system. The *RankineCycler* is a small laboratory sized steam-electric power plant fueled by liquid propane gas. It includes a multi-pass boiler, axial-flow steam turbine, AC/DC generator and a condenser tower. A picture of the *RankineCycler* is shown in Figure 2.

Figure 2. *RankineCycler* [5].

The *RankineCycler* is a laboratory tool intended to teach students about the Rankine cycle, a type of heat engine found in almost all power generation plants. The Rankine cycle is a 4 step process used to convert heat into work. The working fluid in the cycle is typically water.

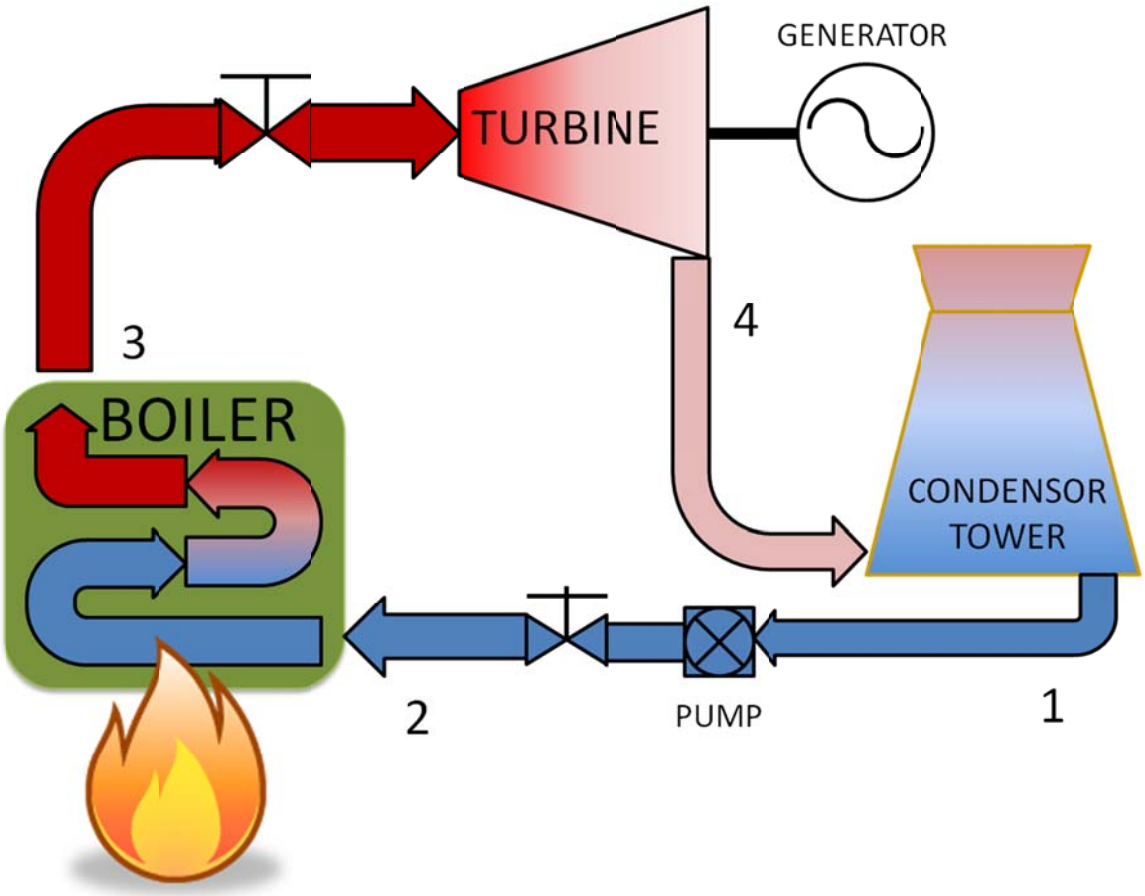


Figure 3. Rankine Cycle Diagram.

The first step in the Rankine cycle is pumping cool low-pressure water into a high-pressure boiler. This is shown as the transition from 1 to 2 in Figure 3. In the boiler, heat is added to the water until it becomes high-pressure steam. Common heat sources for power plants are coal, natural gas, and nuclear fission. In the case of the *RankineCycler*, the heat source is liquid-propane gas. The *RankineCycler*'s boiler is shown in Figure 4.

Figure 4. *RankineCycler* Boiler [6].

The steam then expands through a turbine in order convert the thermal energy into work, shown at point number 3 in Figure 3. Converting this work into electric power is done by connecting the shaft of the turbine to the rotor of an AC generator. Figure 5 shows the *RankineCycler's* turbine. The silver casing is the turbine housing; the black box to the rear of the assembly is the generator. The blue arrows in the cutaway diagram represent steam flow direction through the assembly.

AC Generators typically have the magnetic field component attached to the rotor. In large commercial power plants the magnetic field is generated by field coils excited from an external source. In smaller systems, like the *RankineCycler*, the field component is just a simple magnet. The rotation of the magnetic field creates current in the armature windings, and thus electric power.

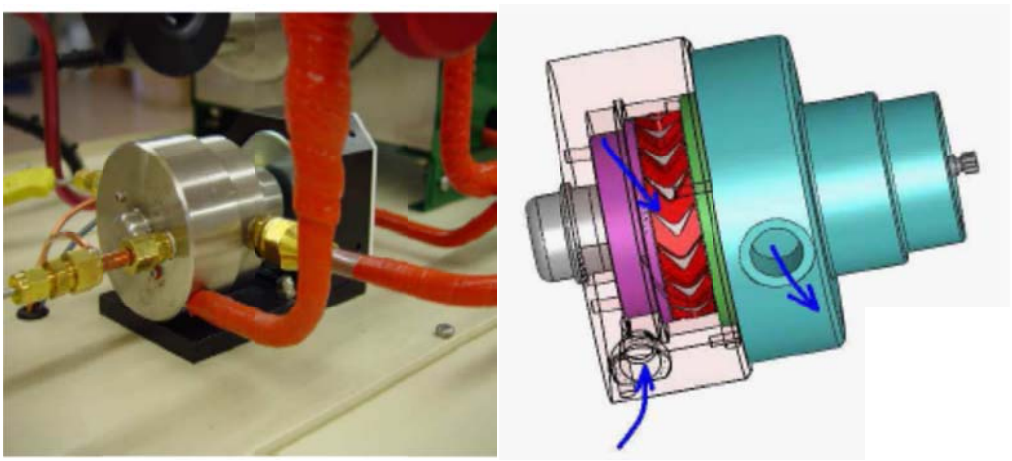


Figure 5. *RankineCycler* Turbine and Turbine Cutaway [7].

Being a laboratory tool, the *RankineCycler* is not set up to power external devices. The power generated is dissipated by an adjustable resistive load controlled on the front panel. A voltmeter and an ammeter are connected to the circuit to display the effects of loading the system to the students. Sensors connected to the data acquisition system also record this data electronically. Figure 6 shows the generator side of the turbine-generator assembly.

The last step in the rankine cycle is when the now low-pressure steam enters the condenser tower. Here the steam becomes water again and it is ready to be pumped back into the boiler. This is the transition from point 4 to point 1 in Figure 3. The *RankineCycler* deviates slightly from the cycle in this step; the condenser tower, shown in Figure 7, is not large enough to convert enough of the steam back into liquid form, so the remaining water is not pumped back into the boiler. This means there is a finite time limit on operation on the laboratory setup. In standard Rankine cycle power plants this is not the case, the working fluid can continue the cycle indefinately, or until the fuel is depleted.

Figure 6. *RankineCycler* Generator.

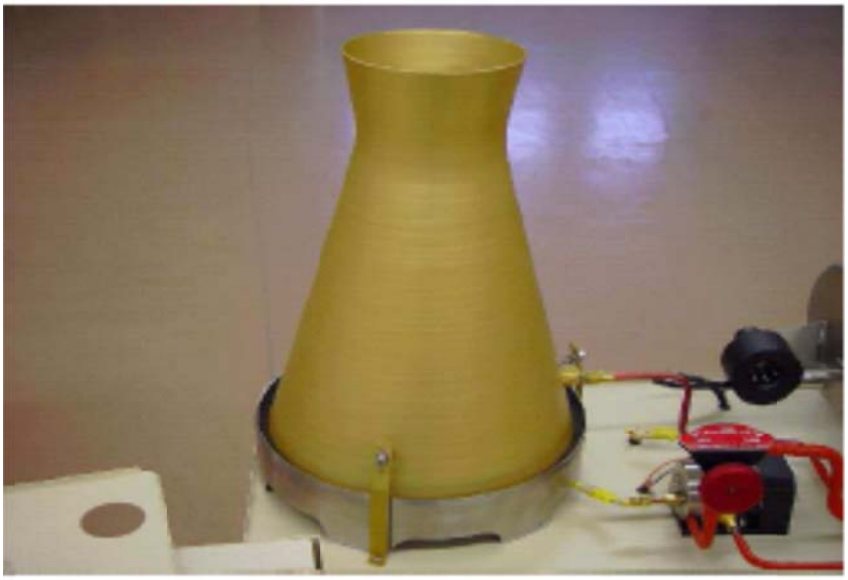


Figure 7. *RankineCycler* Condenser Tower.

2.2 Modifications

The *RankineCycler* was designed as a teaching tool, so it was originally configured for more hands-on laboratory experiments. For the purposes of this thesis several modifications were made to allow for automated control along with expanding the system's capabilities. The following section provides details of the modifications made to the *RankineCycler*. This includes the automation elements added to the control valves, new sensors, additions to the water supply and propane subsystems, and additional changes to the *RankineCycler's* control panel.

2.2.1 Automation

The most important modifications made to the *RankineCycler* were to allow for automated operation of the device. The Multi-Agent Control system runs processes in MATLAB and transmits control values via a network connection. Additional hardware is required to translate these control values into valve positions. The system does this in four steps. First, a LABView VI running on a laptop interprets the commands from the MAS. The LABView code then sends the command to a stepper motor control board connected through USB. The control board then triggers one of four stepper motor driver boards to change the motor position. A diagram of this process is shown in Figure 8.

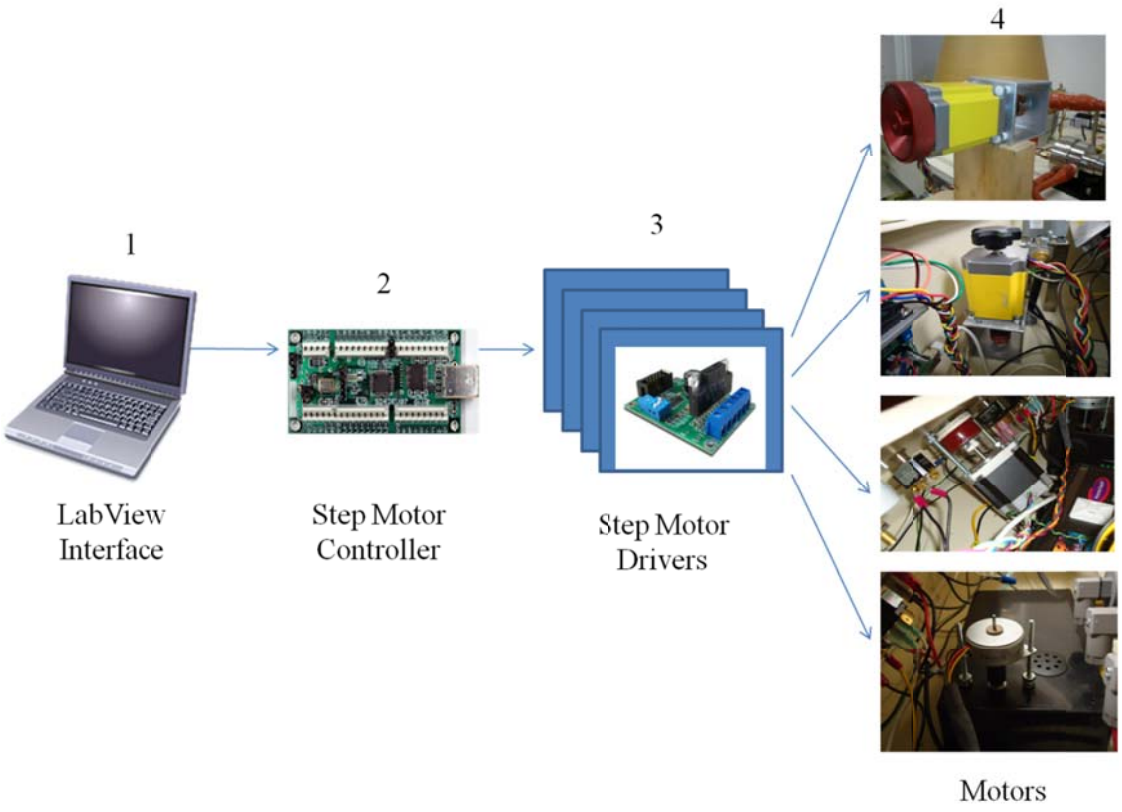


Figure 8. Motor Control.

The four motors installed on the *RankineCycler* are in place to manipulate the steam emission valve, the propane fuel valve, the load potentiometer, and the airflow control potentiometer. The step motors on the steam valve and fuel valve are Probotix HT23-280-8 motors [8]. This motor was chosen because of its high torque capabilities and a double shaft configuration. The torque capabilities of the motors are important because the system as implemented does not have feedback between motor rotation and valve position. Higher torque reduces the chance of motor slip. Additionally, the valves tend to stick near their closed positions, so the higher torque motors ensure the motor is capable of opening valves under all conditions. The double shaft configuration allows for

manual operation of the *RankineCycler* when the step motors are not energized. Pictures of the steam and fuel valve step motors are in Figure 9.

The other two motors are attached to the rotation component of potentiometers. One is attached to the load potentiometer mounted on the underside of the *RankineCycler* control panel. This allows for automated control of the loading of the generator. The fourth motor is mounted on the *RankineCycler*'s original power supply. It rotates the knob that controls the blower fan speed. The LABView control program can be toggled to control the airflow motor based on the fuel valve motor's position. Control over airflow is essential to ensuring the proper fuel-air mixture in the burning chamber. Pictures of the load and airflow motors are in Figure 10.

The four step motor driver boards and the step motor controller are mounted on a single frame behind the front panel of the *RankineCycler*. Mounting the motor control boards together made installation more convenient and allowed for a simple shared heatsink to be used by all four motor drivers. Along with the driver and control boards is a relay to switch AC power to the water pump on command. This relay can be triggered by a switch on the front of the *RankineCycler* or the motor controller.

There is also a small circuit designed to put the driver boards and motors to sleep based on the position of a switch mounted on the front of the *RankineCycler*. The driver boards and control board are shown in Figure 11.

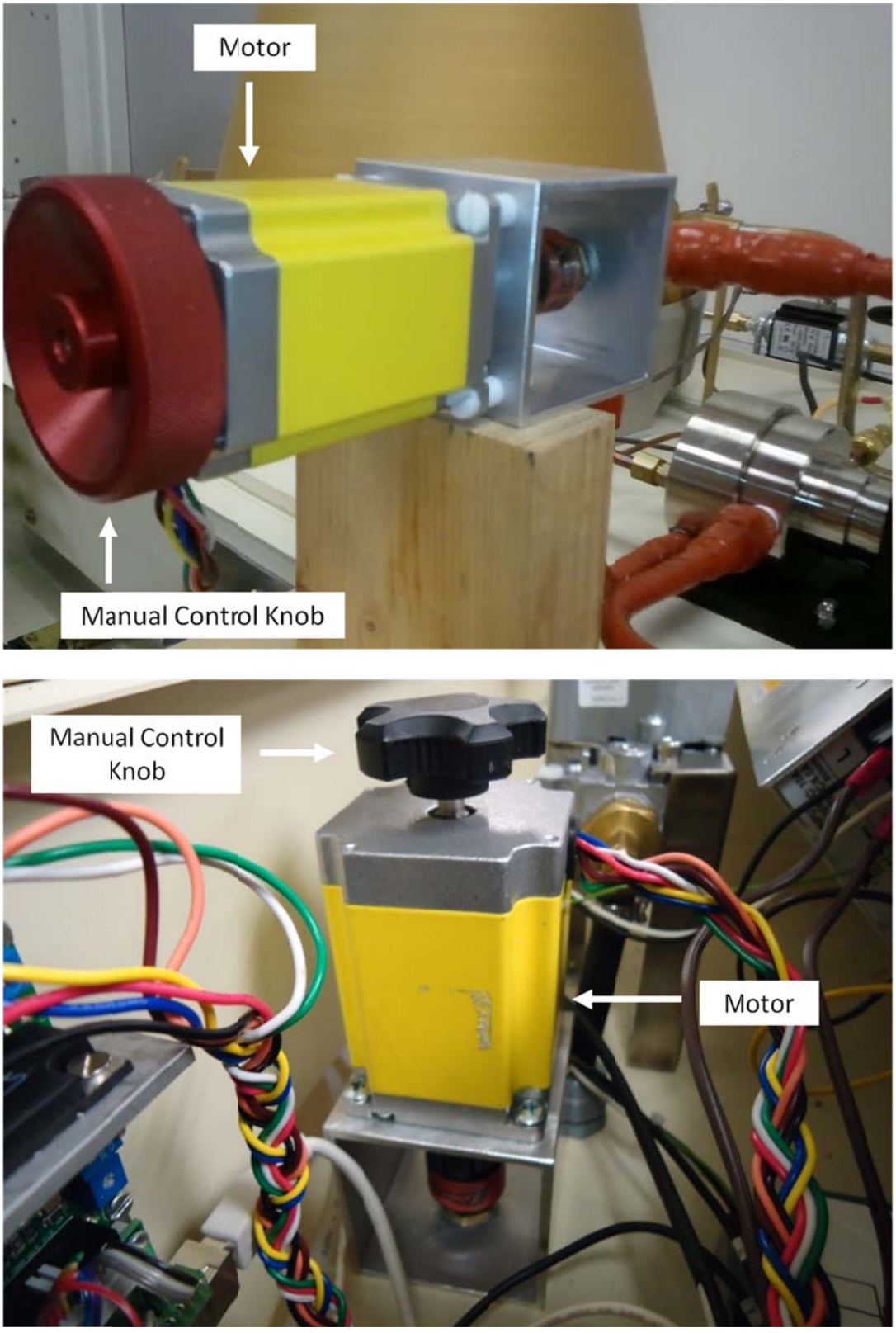


Figure 9. Steam Valve motor (top). Fuel Valve Motor (bottom).

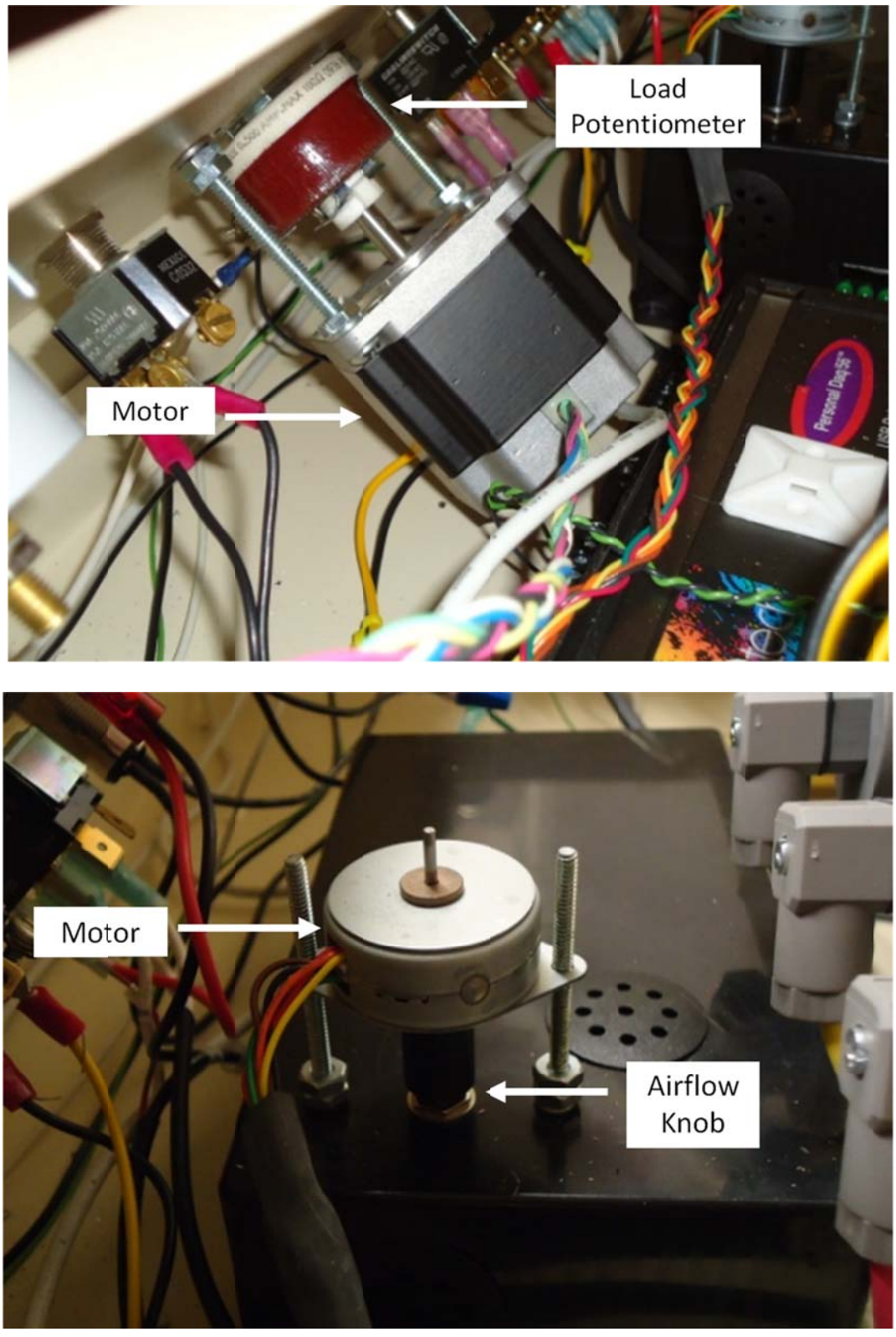


Figure 10. Load Potentiometer motor(top). Airflow knob motor (bottom).

2.2.2 Sensors

The *RankineCycler* came with an extensive sensor system already installed. The system uses an IOtech Personal DAQ/56 USB Data Acquisition module to collect data at several different points in the system. The device has 10 differential input lines, along with 16 digital I/O pins and 4 frequency counters. The DAQ is set up to measure the properties listed in Table 1.

In addition to this default configuration, two of the frequency counters were set up to record water flow into the boiler and generator AC frequency. The switch that toggles the system between manual and automatic mode uses one of the digital input lines. Four additional digital pins are used to relay data from a water level sensor added to the system.

The *RankineCycler* came with software that included LABView Virtual Instruments (VIs) to interface with the DAQ/56. This is the main reason a LABView interface was chosen for the laptop controlling the *RankineCycler*. These VIs use a USB connection to communicate with the data acquisition module. These Vis were modified and expanded for compatibility with the project. More details on the LABView VIs can be found in Chapter Three. The DAQ module is shown in Figure 12.

Sensor Error

Estimates of sensor error are important to establish the accuracy of the data collection process. Table 2 lists each sensor, their accuracy, and the averaging period of the DAQ over their output. The averaging time of the DAQ has significant influence over the control system design by limiting the frequency at which the controller modifies the system.

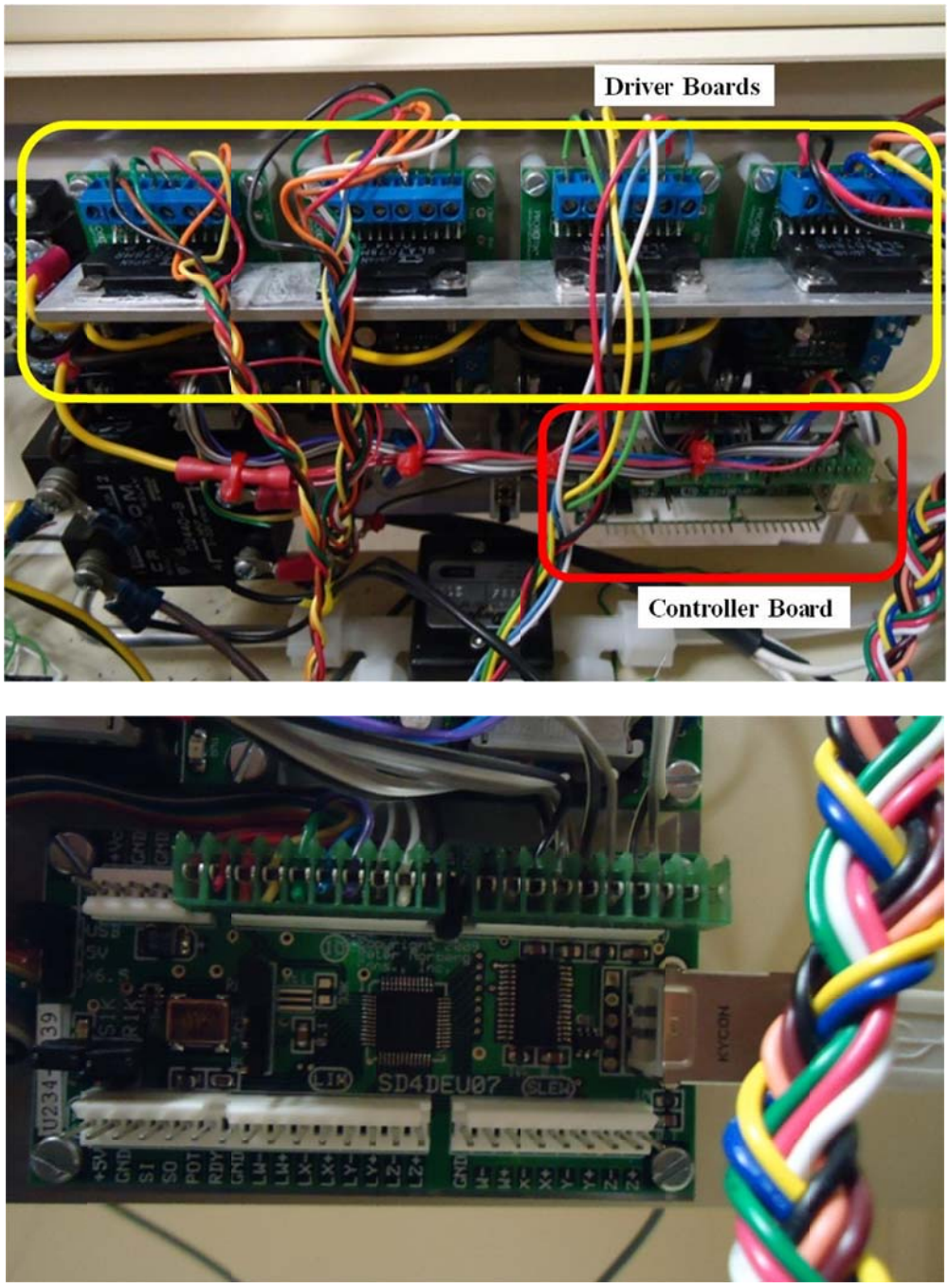


Figure 11. Driver/Controller Mounting (top). Controller board closeup (bottom).

Table 1. *RankineCycler* Sensors.

Boiler Pressure	Steam Turbine Outlet Temperature
Boiler Temperature	Generator Current
Steam Turbine Inlet Pressure	Generator Voltage
Steam turbine Inlet Temperature	Fuel Flow
Steam Turbine Outlet Pressure	Turbine RPM

The most important sensors used by the control system are the boiler pressure and the generator current and voltage. These measurements are used in the feedback loop of the control system.

2.2.3 Water level Sensor

A water level sensor was added to the *RankineCycler* in order to allow for automatic control. This is because one important variable to monitor in a power plant is the water level in the boiler. The *RankineCycler* has a sight glass in front of the boiler to allow for a user to visually monitor the water level. The addition of a sensor to relay this information to the control system is necessary. This was achieved using infrared LEDs and photodiodes alongside the edge of the sight glass. The positioning is shown in Figure 13.

The sensor consists of 16 infrared LEDs, 16 matched IR detectors, two encoder chips, one NAND chip, a power regulator, and an assortment of resistors and potentiometers to get ideal voltage levels.

Figure 12. IOtech DAQ/56.

Table 2. Sensor Accuracy and Averaging.

Sensor	*Sensor Type*	*Accuracy*	*Averaging Time*
Boiler Pressure	Setra Model 209	±0.5psig	40ms
Boiler Temperature	K-type Thermocouple	±3.96º F	110ms
Steam Turbine Inlet Pressure	Setra Model 209	±0.06psig	40ms
Steam Turbine Inlet Temperature	K-type Thermocouple	±3.96º F	110ms
Steam Turbine Outlet Pressure	Setra Model 209	±0.06psig	40ms
Steam Turbine Outlet Temperature	K-type Thermocouple	±3.96º F	110ms
Generator Current	PDAQ/56	21bits	110ms
Generator Voltage	PDAQ/56	21bits	110ms
Fuel Flow	Dwyer TF2110	±0.3 Ltrs/min	40ms
Turbine RPM	PDAQ/56	19bits	40ms

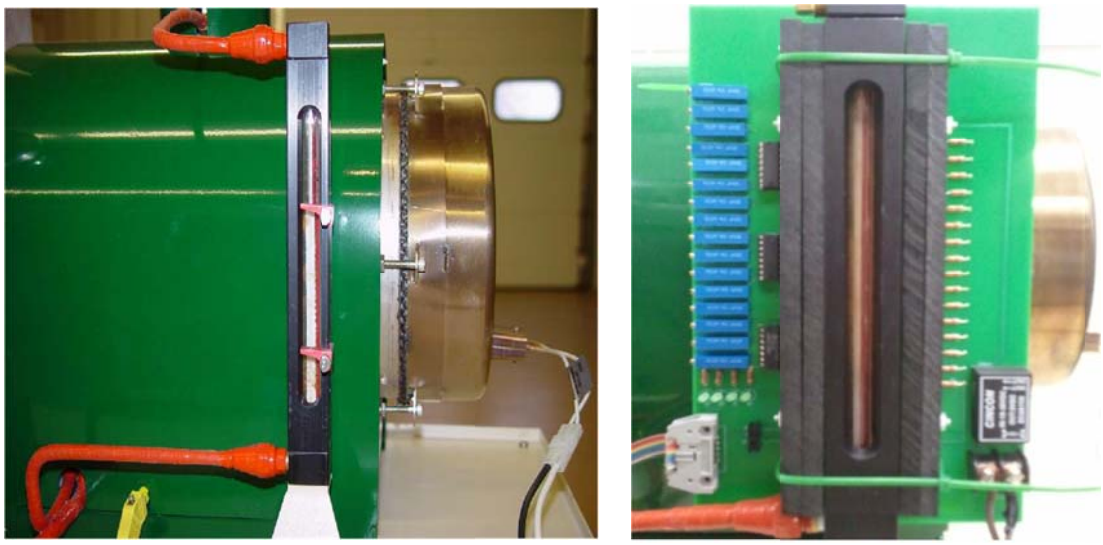


Figure 13. Original Sight Glass (left). Sight Glass with Water level sensor (right).

The IR LED detectors are paired together in an assembly that spreads out along the length of the sight glass, separating the detectors by about 7 mm. The separation between LED detectors is limited by the physical size of the LEDs and phototransistors. They are matched Fairchild Semiconductor QEE113 LED's and QSE113 phototransistors. When the sensor is in operation, each phototransistor detects the LED across the sight glass and presents approximately 5V to the decoder pins. The sensor detects the water level by having one of the IR beams blocked by a floating wood cylinder that was inserted into the sight glass. The board then relays which sensor is blocked through the DAQ to the LABview sensor VI. A close-up of the sensor bars and the water level board is shown in Figure 14.

Figure 14. Water lever sensor bar (top). Unmounted Water level sensor board(bottom).

2.2.4 Water Pump

The most obvious difference between the *RankineCycler* and commercial power plant is the lack of feedwater system into the boiler. This prevents the *RankineCycler* from achieving true steady-state operating conditions. In an effort to correct this flaw, a high-pressure piston pump was added to the back of the *RankineCycler.* The pump is a Fluid-o-Tech 70 W "Mono" oscillating piston pump. The pump was chosen due to the specifications listing of 15 l/h at 120 psi. This allows the pump to fill the boiler even

when the boiler is at its maximum pressure. The pump is controlled by the same control board that controls the motor driver boards. One of the digital pins triggers an ac relay to activate the pump. The flow meter is a FLR1010ST manufactured by Omega. The flow meter is connected to the DAQ through one of the frequency counter pins. The mount on the back of the *RankineCycler* holding the pump and flow sensor is shown in Figure 15.

Figure 15. Pump and flow sensor.

2.2.5 Switches

Two switches were added to the front side of the *RankineCycler*. One switches the system between manual and automatic modes by toggling the sleep function of the motor driver boards. While in manual mode, the *RankineCycler* is operated exactly like its original configuration. The second is for manual operation of the pump while the LABView controls are disabled.

2.2.6 Power Supply

A Meanwell SE-450 DC power supply is installed underneath the front panel of the *RankineCycler* in order to power the modifications. It powers all of the step motors, the water level sensor, and the water pump. Specifications on the power supply are

included in the appendix. The power supply position next to the propane regulator is shown in Figure 16.

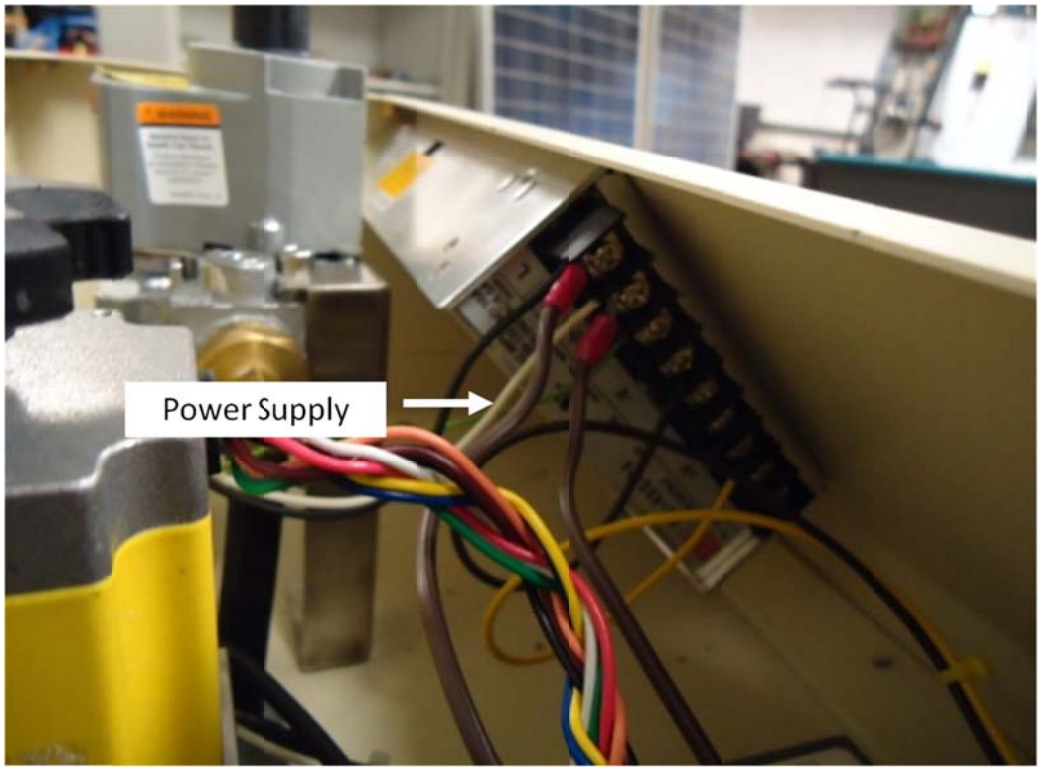


Figure 16. Power Supply.

CHAPTER THREE

Project Software

The system software for this project can be categorized based on development environment. The bulk of the software is the implementation of a Multi-Agent Control System (MACS), which is written in MATLAB. Several LABView VIs were also developed to link the MAS to the *RankineCycler's* automation and sensor hardware. The following chapter does three things: describe the overall the control structure of the system, define the Multi-Agent System and how it operates, and provide code overviews for both the MATLAB algorithms used in the MAS and the LABView VIs that allow for automation of the *RankineCycler*.

3.1 Control Structure

The overall control scheme consists of four main elements: the reference governor, feedforward controller, feedback controller, and the gain optimizer. Figure 17 shows a diagram of the interaction between the components.

The reference governor generates power and steam pressure setpoints from a load demand curve. It takes a desired power output E_d and converts it to valve positions u_1* and u_2* and a desired operation pressure P_d to be applied to the *RankineCycler*. However, the process of converting desired power to valve positions is not simple.

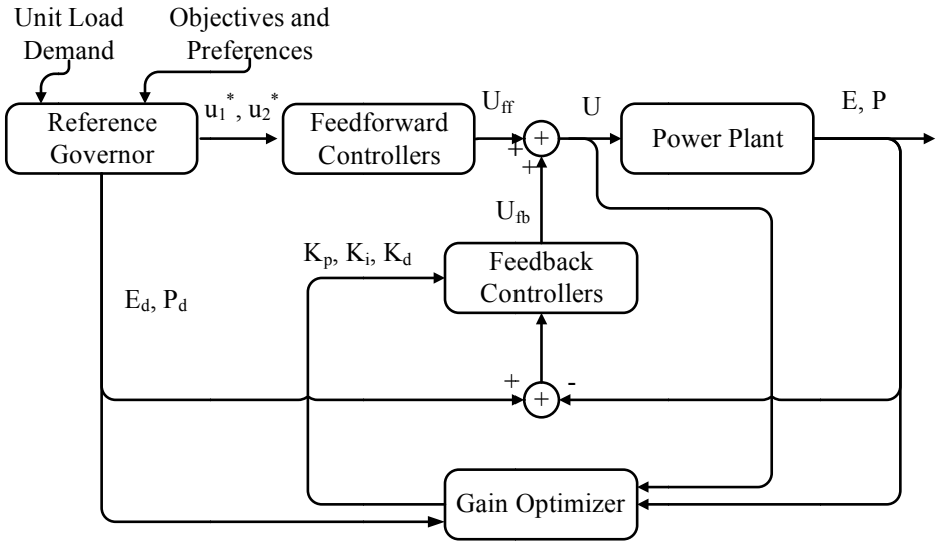


Figure 17. Control Structure.

The reference governor converts desired power to valve positions in three steps. First, it finds a range of possible valve settings at which the system can operate and provide the desired amount of power. These are called the operating windows, Ω_1 and Ω_2 . Then the reference governor runs a particle-swarm optimization algorithm based on the objectives and preferences of the control system. For example, the optimization algorithm can be configured to prioritize fuel efficiency over set-point accuracy or any other combination of optimization settings. Finally the optimal valve settings are sent to the setpoint scheduler. A diagram of the reference governor is shown in Figure 18.

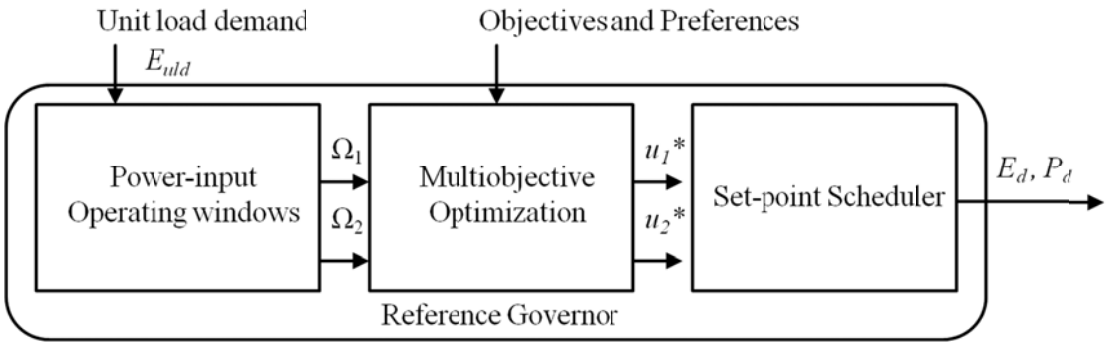


Figure 18. Reference Governor Diagram.

The output from the reference governor is sent to the feedback controller and the gain optimizer. The optimal valve positions and the desired power and pressure calculated by the reference governor are used by the feedback controller to produce valve positions, *U,* to be sent to the power plant. The gain optimizer also uses this information to assign the proper gains to the feedback controller.

The MAS self-organizes into a network of agents that divide up the tasks of this control scheme.

3.2 Multi-Agent System Overview

A Multi-Agent System(MAS) is a software program that is distributed amongst multiple different computing environments in order to accomplish a coordinated goal. It consists of several entities called "Agents" that are by definition reactive, proactive, social, flexible and robust [9]. These agents communicate with each other in order to form the control system. Some advantages of a Multi-Agent Control System come from the distributed computing power and improved durability. A MACS can also run more computationally intensive algorithms than standard control systems. For instance, feedforward control is typically accomplished by using a neural network or lookup table to predict desired steady-state setpoints. In a standard control system these elements could not adapt to changes in the plant during operation. The MACS design, however, can re-train and implement new neural networks based on readings from the plant during operation. This means the MACS has increasingly accurate feedforward control values over time. The MACS is also more durable because it is designed in such a way that in the case of failure, the MACS reconfigures remaining agents to accomplish its tasks.

An Agent in this implementation of MAS is a block of continually executing MATLAB code. Each agent requires two processing threads, so an agent can exist on any multithreaded computing platform connected to the network. Multiple agents can also exist on a single machine, given sufficiently capable hardware. This allows for flexibility in distributing the MAS between hardware platforms. One thread accomplishes the task the MAS requires of that particular agent; the other thread handles communication with the other agents in the system. These two threads communicate with each other via functions in the MATLAB Parallel computing toolbox. A diagram of the agent architecture is shown in Figure 19.

Once the system is running, each agent can be described by the task it is currently running. There are three levels of tasks an agent can perform in the system hierarchy, and one separate startup state. The tasks are shown in Figure 20.

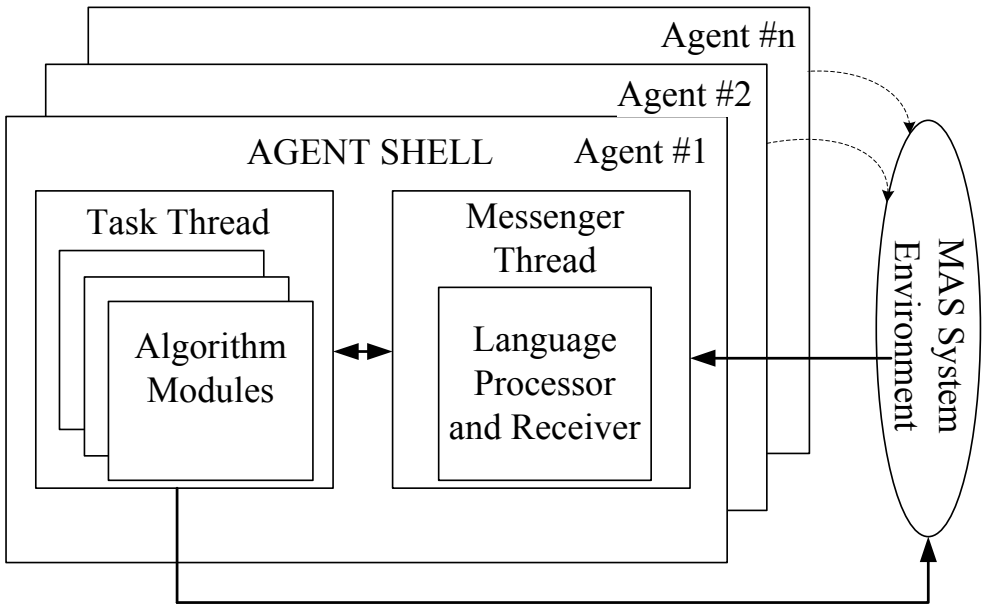


Figure 19. Agent Architecture[10].

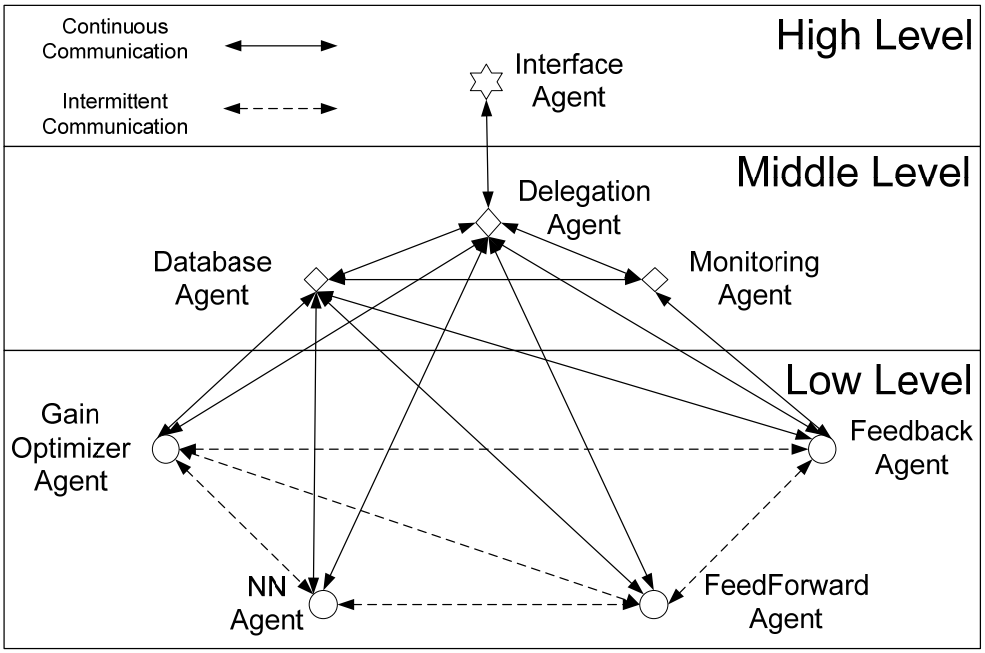


Figure 20. Agent Task Hierarchy[10].

The *Interface* Agent is considered a high level task because it is responsible for interacting with the human operators of the system. The Interface Agent can be used to change any of the operational settings of the MAS. This agent also displays the location and tasks of all the active agents on the network, along with power plant status information. The Interface Agent can be used to switch a specific piece of hardware to perform a specific task, to change the optimization preferences of the reference governor, change the feedback gains, and most importantly it can be used to control the desired power output of the system. Interaction with this agent can be performed from any computer on the same network as the MAS.

The three tasks in the middle level are responsible for monitoring and maintaining the integrity of the Multi-Agent System. The *Delegation* Agent is the most important; it delegates tasks to all the other agents on the network. It has a prioritized list of tasks based on importance to the control loop which it uses to ensure the minimum required

tasks to control the power plant are always being performed by the MAS. In the case of one agent dropping from the network, the delegation agent reassigns the tasks to make sure the system is still running. The *Monitoring* Agent is responsible for collecting data from both the power plant and the MAS. It then coordinates with the *Database* Agent for more permanent data storage and insuring that any agent that needs the data can access it.

The lower level tasks run the algorithms that perform the actual control of the power plant. The *Feedforward* Agent generates optimal steady-state setpoints for the power plant. Essentially, it performs most of the Reference Governor functions. The *Feedforward* Agent uses neural networks generated by the *Neural Network* Agent in order to calculate the setpoints. The Neural Network Agent is responsible for retraining the neural networks while the system is running. It also makes decisions on when to retrain by comparing plant outputs with the setpoints being sent to the plant by the MAS. The *Feedback* Agent is the feedback element of the control system. Due to speed requirements, this is the agent that sends the control signals directly to the power plant. It generates these control signals based on the steady-state setpoints sent from the Feedforward agent and feedback from the power plant using a PID control algorithm. The gain values used in the PID controllers are generated by the *Gain Optimizer* agent, which provides the optimal gains based on the current and desired power output levels.

The final task an agent could be performing only occurs on startup and in the event of a delegation agent switch. A *Free* Agent is an agent whose task is to poll the network for other agents and become a delegation agent if the system does not currently have one.

A more detailed description of each Agent's specific algorithmic structure can be found in section 3.4.

3.3 MAS Communication

There are two types of communication necessary within the multi-agent system. The first is the intra-agent communication that takes place between the task and messaging threads. The second is the communication over the network between agents. Intra-agent communication is carried out by synchronizing the memory of the task thread with the messaging thread as new data is received by the messaging thread from other agents. Inter-agent communication is done by sending UDP messages over the network. At the lowest level, network communications use built-in MATLAB functions to send the UDP messages.

When an agent sends a message to another agent, it does so using an Agent Communication Language (ACL). The ACL defines the structure of messages being sent. The ACL used here is derived from a protocol proposed by the Foundation for Intelligent Physical Agents (FIPA) [11]. The ACL structure is modified slightly to fit the specific needs of this system. An advantage of using such an ACL is that additional messaging capabilities can be added or changed by adding or modifying the necessary fields in the common message structure used by all agents. Messages in the MAS system contain at least four fields, with an optional fifth field, shown in Figure 21.

Figure 21. MAS message field structure[10].

The first field is the *performative* field, which contains a keyword that informs the receiving agent for what purpose the message is being sent. Possible performatives are request, inform, subscribe, agree, refuse, and notUnderstood. The *request-performative* informs the messaging thread that its agent will need to either send data or consider becoming another agent. For example, using this performative, the delegation agent can request a free agent become the feedback agent. The *inform-performative* informs the messaging thread that it is being sent data for the task thread. For instance, when the load demand vector E_{uld} changes the feedforward agent sends an inform message to the feedback agent with the new data. The *subscribe-performative* informs the messaging thread that an agent wants to be sent the most recent copy of particular data anytime that data is changed. The *agree-performative* informs the messaging thread that a task requested of another agent is going to be performed. The *refuse-performative* informs the messaging thread that a task requested of another agent cannot be performed. The *notUnderstood-performative* informs the messaging thread that a message previously sent to another agent was not readable by that agent, meaning either that the agent was not programmed to respond to that type of message or the message was corrupted.

The *sender* field of the MAS message structure simply contains the identification of the agent sending the message. For example, this field will contain data such as task, network address and the port the sending agent is sending and receiving messages on.

The *receiver* field contains the identification of the intended recipient of the message. This field will contain data such as the network address, operating port, and task of the recipient agent.

The *content* field, depending on the type of message, will contain things such as data, the name of a data file, an instruction, or possibly nothing at all. When a request is sent for an agent to take on a certain task, the content field will contain the name of the task. If an inform message is being sent, the content field will contain data or the location of data on the network. If a refuse or notUnderstood message is being sent, the content field may contain a reason for refusal or other error message. If a simple agree message is sent, the content field does not need to contain anything, because the content is already in the performative.

The *replyby* field is an optional field that is used when a message of importance is being sent and needs to meet a deadline. This field is meant to imply to the recipient agent that this message is of critical importance and needs to take priority over what it is currently doing.

The message is converted to a string for communication purposes. The conversion format uses a '$$' character to delineate between fields and a '$' character to distinguish separate items within a field. For example, a message from the delegation agent requesting a free agent to change tasks to the feedback agent is translated to:

"request$$192.168.0.1$10001$delegationAgent$$192.168.0.2$10002$freeAgent$
$become$feedbackAgent"

3.4 MATLAB Algorithms

The algorithms that control the power plant are executed by the task thread of each agent. The system is designed so that any number of agents can perform any number of tasks, but there are three main tasks used to control the *RankineCycler* specifically. Three agents are sufficient because of the relative simplicity and size of the

RankineCycler; in a larger system more would be used. *RankineCycler* limitations also restrict the usefulness of agents with more long term goals, like the gain optimizer and the neural network training agent. The three agents required are the Delegation, Feedforward, and Feedback agents. MAS agents also have the default task of a Free Agent when the system comes online. The following sections describe these algorithms in detail. Due to the nature of the system, all of these algorithms are operating in parallel.

3.4.1 Free Agent

The purpose of the free agent algorithm is to initialize the Multi-Agent System. The first step is polling the network for other agents. This can be done using either the TCP or UDP communication protocols depending on configuration. The Free agent then uses the responses to build a list of existing agents connected to the system. Delegation Agent detection is the next part of the algorithm. The delegation agent is the most important agent in the system because it assigns tasks to the other free agents. If a free agent determines there is no Delegation Agent and it has the lowest IP and port combination of all the reporting agents on the network, it becomes a Delegation Agent. A flowchart of the Free Agent algorithm is shown in Figure 22.

3.4.2 Delegation Agent

The delegation agent is responsible for making sure the MAS is populated with the required agents. It also queries the other agents periodically to make sure they are still connected to the system.

The algorithm is relatively simple. It has a prioritized list of tasks starting with the feedback and feedforward agents and an agent table of all the agents reporting into the

MAS. It goes down the list of agents assigning tasks. While the other agents are performing the control tasks, the delegation agent periodically sends acknowledgement requests to the other agents in order to maintain the agent table. The period is every 10 seconds.

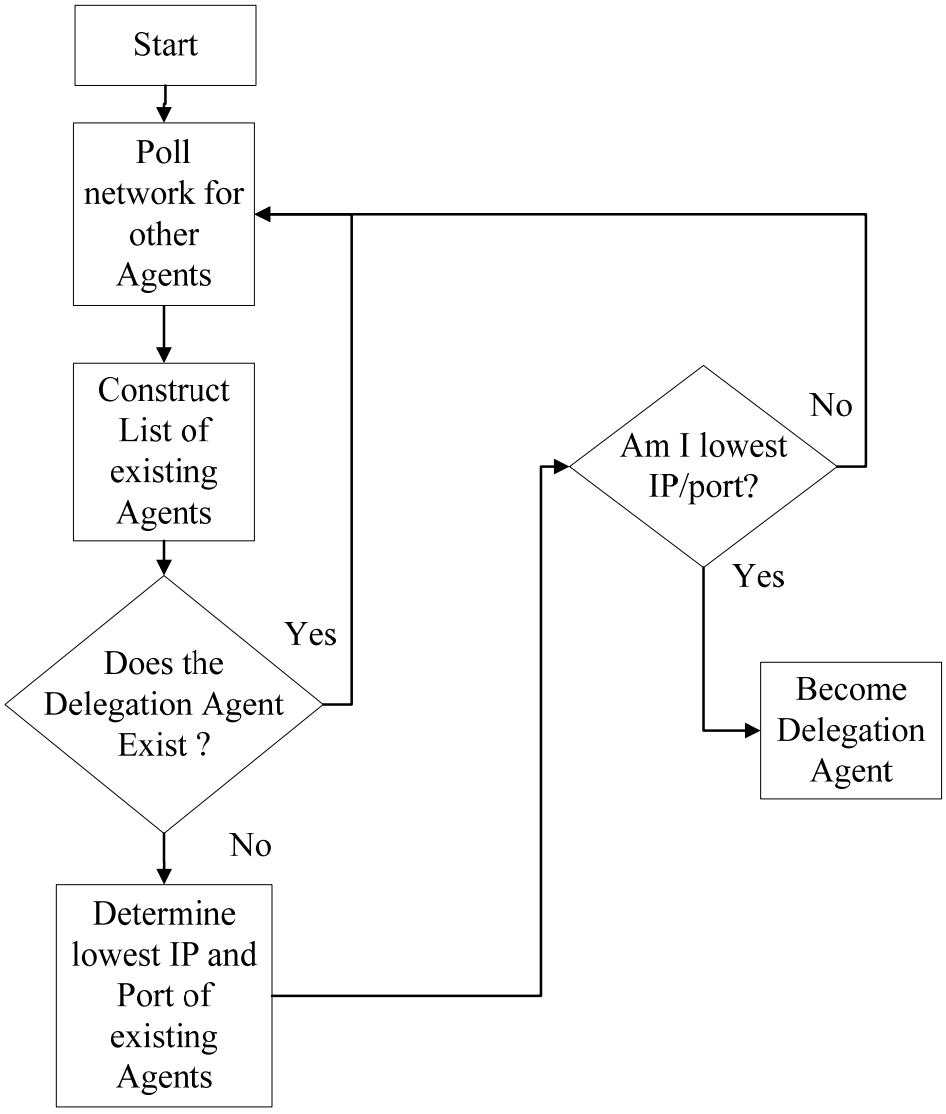


Figure 22. Free agent code flowchart.

3.4.3 FeedForward Agent

The feedforward algorithm accomplishes the reference governor functions. The overall goal of the feedforward agent is to generate optimal steady-state setpoints for the power plant. The feedforward agent also controls the timing of the system and attempts to follow programmed demand curves. The algorithm waits until it receives a demand power or E_{uld}. The E_{uld} can be either a single point or a series of points representing a demand curve. See section 4.2 for an example of a demand curve. Once the feedforward agent gets a E_{uld}, it calculates the optimal control values using a Hybrid Particle Swarm Optimization (HPSO) process. These control values include both output and input values for the power plant. Output values E_d and P_d represent the optimum power and pressure operational points to achieve the desired power output. The vector of input values u^* contains the valve positions that should achieve the desired output values. The feedforward agent sends these results to the Feedback Agent. A flowchart of the feedforward agent operation is shown in Figure 23.

Hybrid Particle Swarm Optimization.

The core algorithm of the feedforward agent is a hybrid particle swarm optimization (HPSO). This is a variant of the particle swarm search algorithm originally attributed to Kenndegy, Eberhart and Shi [12].

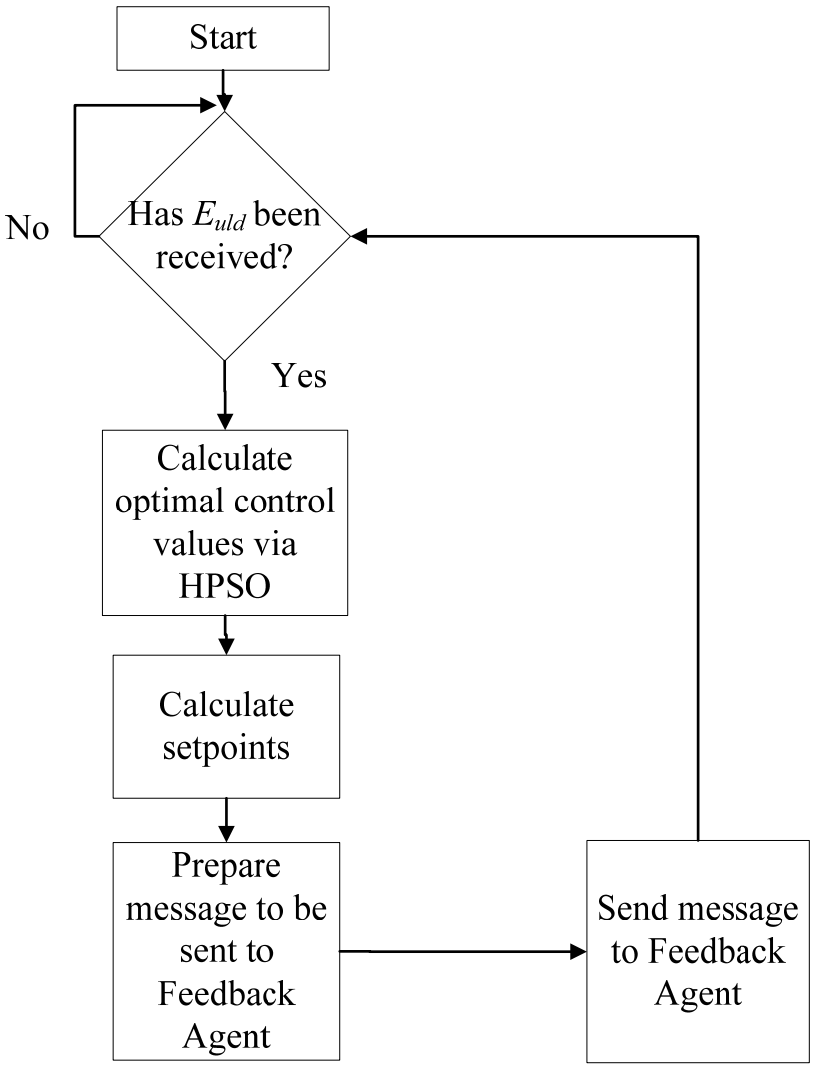


Figure 23. FeedForward Agent Code Flowchart.

PSO optimizes a problem by evaluating a population of candidate solutions, called particles, and then moving these particles around the search space according to a mathematical formula involving the particles position and velocity. Each particle moves towards its own personal best solution and the global best solution of the swarm in each iteration. The equations governing their movement are shown below.

$$pos(n) = pos(n-1) + v$$

$$v = w * v(n-1) + c_1 * \left(p_{best} - pos(n-1)\right) + c_2 * \left(g_{best} - pos(n-1)\right)$$

Where w is a scaling value depending on the iteration. c_1 and c_2 are weight values randomly selected from a range of values between 0 and 0.5. P_{best} and g_{best} represent individual particle personal best position and the swarm global best position, respectively. In this swarm implementation, a particle position is a vector consisting of valve stepper motor positions for the *RankineCycler* fuel and steam valves.

Each particle position is used as the input to a neural network approximating the *RankineCycler*. The outputs are then weighted by the following objective functions to determine the best position.

$$cost = sum(J * \beta)$$

$$J_1 = abs(E_{uld} - E)$$

$$J_2 = u_1 - u_{1min}$$

$$J_3 = u_2 - u_{2min}$$

$$J_4 = P - 80$$

The position with the lowest cost is the best position based on the objective functions. β is the preferences and is used to weight the objective functions to achieve the optimization goals. The default setting is β = [1 0.1 0.1 0.4]. This weights the first objective function, J_1, as the most important. E is the power output at that particle position, P the pressure. E_{uld} is the power demand. J_2 and J_3 reference u_1 and u_2, which are the fuel and steam valve positions. J_4 is important to note because it compares the output pressure to 80 psi. This means the system tries to keep the boiler pressure close to 80 psi. This objective function was added because of limitations on the *RankineCycler*.

The HPSO is set to run 100 iterations with 140 particles. This was chosen due to timing constraints; in some situations the feedforward agent has to produce output in under a second. Ideally the system would run the optimization as often as possible during a ramp, however the *RankineCycler* motor control system cannot issue commands to the motors faster than once a second. The HPSO could be tuned to produce optimal setpoints less frequently, but it would result in delayed responses to demand changes. Due to the way MATLAB handles neural network evaluations, evaluating many particles over a few iterations is faster than evaluating few particles over more iterations.

Figure 24 shows the progress of the particle swarm. Each circle represents a pair of valve positions being evaluated by the neural network and objective functions. Over the course of several iterations the swarm moves toward optimal solutions. The top figure is the randomized starting point of the swarm. The bottom figure is the end result, where most of the particles have converged to the optimal solution.

The "Hybrid" term in HPSO describes a modification of the search technique used to improve performance. This means at the end of each iteration, the lowest performing particles are moved to the position of the highest performing particles while still retaining their velocity vectors. This modification was chosen due to the research by Lee and Garduno-Ramirez [13] that compared standard particle swarm, hybrid particle swarm, evolutionary particle swarm, constriction factor approach, and genetic algorithms as applied to power plant feedforward systems.

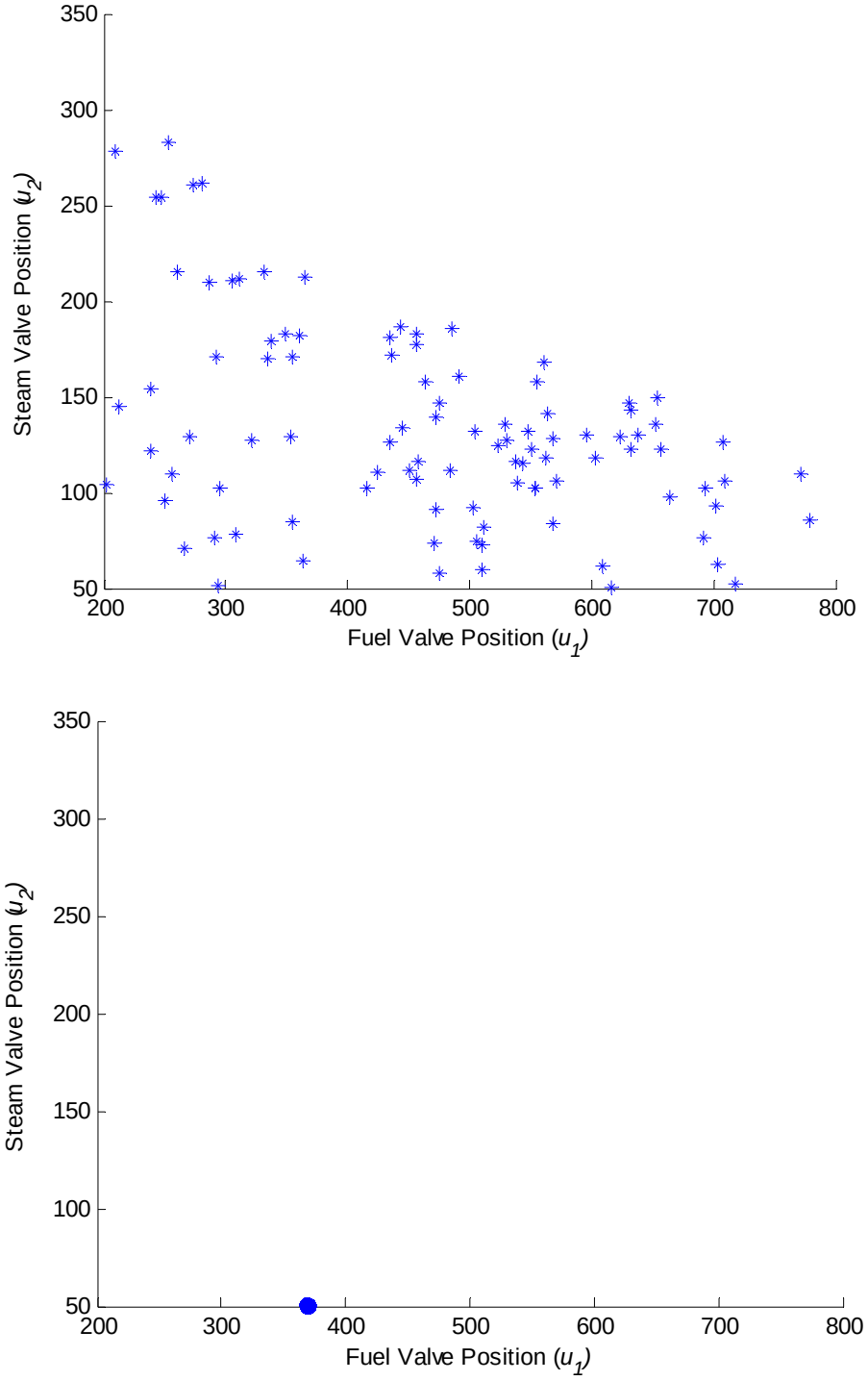


Figure 24. Starting point of the Particle Swarm (top). After 100 iterations (bottom).

The conclusion of that research is hybrid particle swarm is the superior algorithm in most cases. The comparison of each type of algorithm was done using system equations of an existing power plant, the same model used in the original development of the MAS. This plant is much larger than the *RankineCycler*; it is a180MW natural gas fired plant. An assumption was made that the search space of the two plants would be similar in shape, so the HPSO would still be a good choice. However, the last two objective functions were changed to compensate for the differences between the two.

3.4.4 Feedback Agent

The algorithm run by the Feedback agent contains the feedback loop for the control system. The core of the algorithm is a multiple input multiple output PID controller. The Feedback agent also has code for saving data histories and water pump control.

The feedback algorithm is timer driven, and executes the PID calculation once every second while continuously checking for new power plant, feedforward, and gain optimizer data. A flowchart of the algorithm is shown in Figure 25.

The PID calculation is a two-input, two-output equation that uses desired power and pressure from the feedforward data as the setpoints. The integral window is 100 seconds long. The PID equation is as follows:

$$Pn(n) = Pn(n-1) + K_p * Er(n) + K_i * \int_{n-100}^{n} Er(\tau)d\tau + K_d * \frac{dEr(t)}{dt}$$

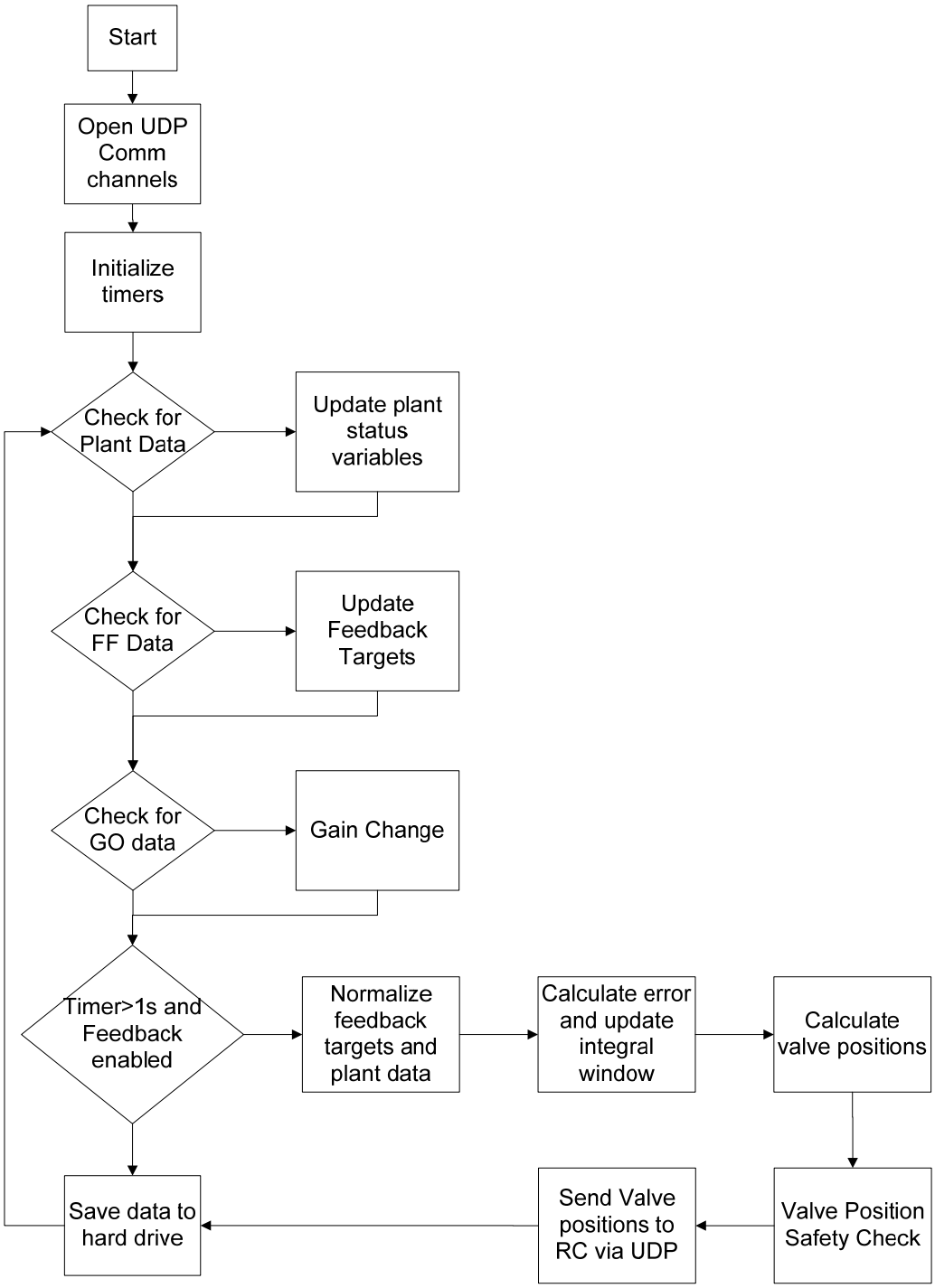


Figure 25. Feedback Agent code Flowchart.

$$\text{Where: } Pn(n) = \begin{bmatrix} Pn_{Fuel}(n) \\ Pn_{Steam}(n) \end{bmatrix}$$

$$Er(n) = \begin{bmatrix} E_d - E \\ P_d - P \end{bmatrix}$$

$$K_p = \begin{bmatrix} 0 & 0.1 \\ 0.1 & 0 \end{bmatrix}$$

$$K_i = \begin{bmatrix} 0.005 & 0.001 \\ 0.0001 & 0 \end{bmatrix}$$

$$K_d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$Pn(n)$ represents the valve positions and $Er(n)$ the power and pressure error. K_p, K_i, and K_d are the gain matrices. The feedback agent has the ability to use a derivative term in the feedback equation, but in the current implementation of the system this ability is unused and the gains are set to zero. Feedback gains were obtained through a combination of the Ziegler-Nichols method and manual tuning. It is important to note the feedback agent has the ability to change gains in real time while the system is operating. This is essential to the function of a gain optimizer agent.

The feedback agent also has safety mechanisms in place to prevent rapid changes in the valve positions and to make sure the equation does not give valve positions outside of acceptable ranges. The fuel valve is limited to moving less than 100 steps per iteration with the range [150 1000] steps. The main reason for the speed limitation is to allow air intake blower fan to appropriately adjust to fuel flow changes. If this limit is not in place the airflow extinguishes the flame inside the burning chamber, effectively shutting off the boiler. The lower limit on the range is there for similar reasons: below 150 steps there is not enough propane in the burn chamber to maintain the burn. Step positions above 1000

do not change the propane flow rate, this is the effective maximum position of the valve. Similarly, the steam valve is limited to 50 step changes with a range of [50 400] steps. The lower limit is in place to ensure the turbine is always spinning; motor positions below 50 do not allow enough steam into the turbine to spin the generator. The upper limit is the effective maximum position of the steam valve. Limiting the steam valve movement per iteration to 50 steps was done to prevent the pressure in the turbine intake chamber from spiking too rapidly. Tests without the speed limitation resulted in minor damage to the turbine bearings and turbine rotation speeds beyond recommended operational limits. It is important to note that typically the feedback gains are too low to impose a steam valve position change greater than 50 steps per iteration. The safety limit is still in place to prevent software errors or user mistakes that assign abnormally high gains to the feedback loop.

3.4.5 Interface Agent

The interface agent is a higher level agent used to interact with the MAS while it is running. It is a GUI created using the MATLAB GUI development tool. While still called an agent, the interface agent does not have the same algorithmic structure as the other agents in the MAS; it is a single-threaded algorithm and can be run from any workstation or platform connected to the same network as the MAS. It can also run on a computer simultaneously with other agents.

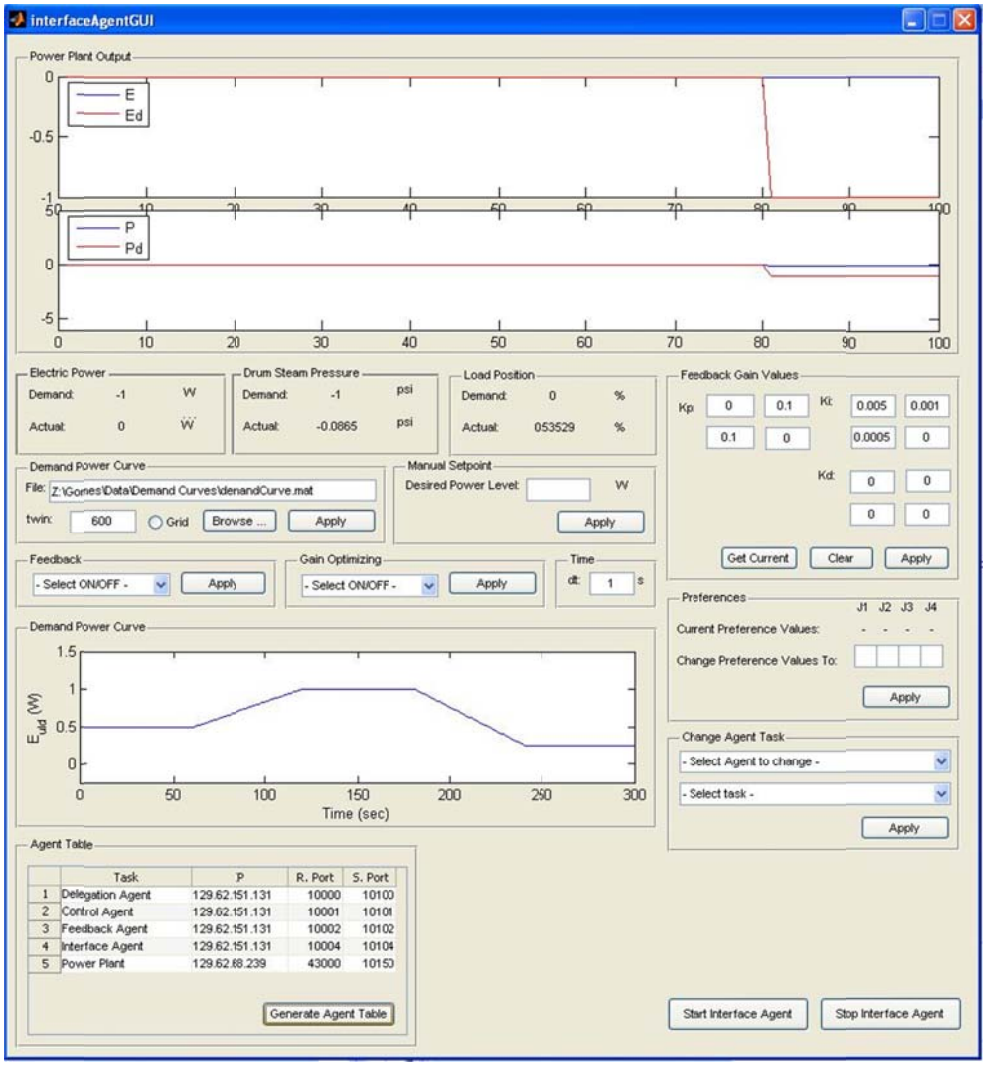


Figure 26. Interface agent GUI.

The interface agent, like most GUIs, is largely interrupt driven. It does have a main operating loop that continuously checks the output file of the MAS and updates its displays and figures appropriately. Most of the *RankineCycler* and MAS data are displayed in the top two sections of the GUI, shown in Figure 26. The two axes in the top section of the GUI display the power and pressure output and targets over time. The real-time state of the *RankineCycler* is displayed in the section below and to the left of the figures.

The other sections of the GUI are controls for the MAS. The interface agent can assign demand curves for the system to follow, set specific power setpoints for the system, enable and modify the performance of the feedback agent, change the preferences of the feedforward agent, and even change the tasks of each agent. All of this can be done in real time while the MAS is running. The interface agent achieves these tasks by sending MAS format messages to each agent responsible for the change.

3.5 LABView Algorithms

The software that directly interfaces with the data acquisition system and the motor control boards is run in National Instruments' LABview environment. Software programs in LABview are typically referred to as Virtual Instruments or VIs. These VIs are run on a laptop connected to the data acquisition and motor control boards of the *RankineCycler*. The laptop uses its wireless networking card to establish communication between the VIs and the Multi-Agent System. These VIs have five tasks: DAQ configuration, DAQ sensor read, motor system control, motor system interface, and network communication.

VIs do not function in the same manner as C++ or MATLAB code. Each VI has a front panel and a block diagram. The front panel is a graphical user interface with interactive switches, input boxes, and displays. The block diagram represents the logic in the program. It is similar in appearance to a flowchart, but more detailed. Programming in LABview is comprised of manipulating this block diagram. The following sections describe in detail the front panels and block diagrams of the system VIs. Full printouts of the VIs are included in the appendix.

3.5.1 Acquisition and Control

The top level VI is the Acquisition and Control VI. The control system is initialized by running this VI in LABview, and this VI calls the other LABview functions as subVIs. The VI front panel is shown in Figure 27.

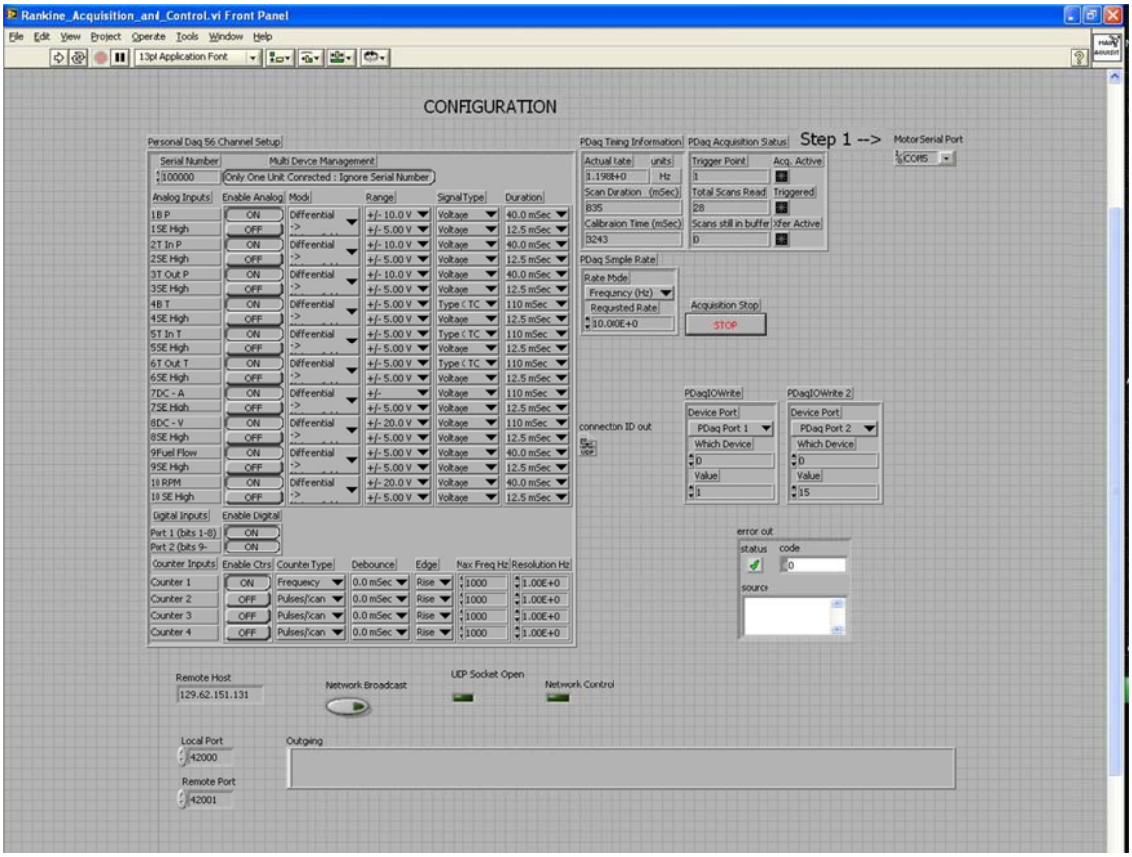


Figure 27. Acquisition and Control Front panel.

The upper left part of the VI's front panel is a GUI that allows for the configuration of each individual sensor. The default settings can be found in the appendix. The top right portion displays DAQ timing information after the VI has been synchronized with the DAQ. The lower portion of the VI is dedicated to outgoing communications. When the "network broadcast" button is toggled the VI broadcasts the

RankineCycler sensor information to the IP and port designated in the lower left fields. The large box displays the text format of the message for bug testing purposes.

The Acquisition and Control VI starts by initializing the DAQ and calibrating the signals for their respective sensors, along with setting timing constants. It then enters its main logic loop. Each iteration of the VI includes a DAQ read, a Gauges VI call, a Motor Control VI call, and if the network broadcast is toggled, a UDP broadcast. The DAQ read is performed by a subVI which was included in the software package of the IOtech DAQ. The block diagram can be found in the appendix. The upper portion of the block diagram translates and compresses the resulting data so the Gauges VI and the UDP logic block can read it. The center portion of the block diagram is the Motor Control subVI call. This subVI's operation is described in detail in section 3.5.3. It performs its tasks then returns the motor positions to be combined with the sensor data. The lower portion of the block diagram is the UDP logic. When the network broadcast button is toggled, the UDP socket is created and configured. All following loop iterations then broadcast the sensor data and the motor position data to the designated IP and port. Before being sent, the data is translated to MAS standard messaging format with the inform performative and the 'PPdata' content modifier.

The loop execution speed of the Acquisition and Control VI dictates the maximum update frequency of the MAS. The VI executes once every 825ms. The bulk of this time is the DAQ read, which clocks in at around 700ms mainly due to sensor averaging delays. The motor control subVI call requires an additional 125ms. This delay is an important design constraint for the feedback loop.

3.5.2 Gauges

The Gauges VI is one of the subVIs called by the Acquisition and Control VI. It is a relatively simple VI: it displays the system sensor reading in real time. The Front Panel is displayed in Figure 28.

The Gauges VI displays all the relevant sensor information for the boiler, turbine and generator. The turbine intake and exit temperatures and pressure are combined into the same graphical element. The front panel is shown in Figure 28.

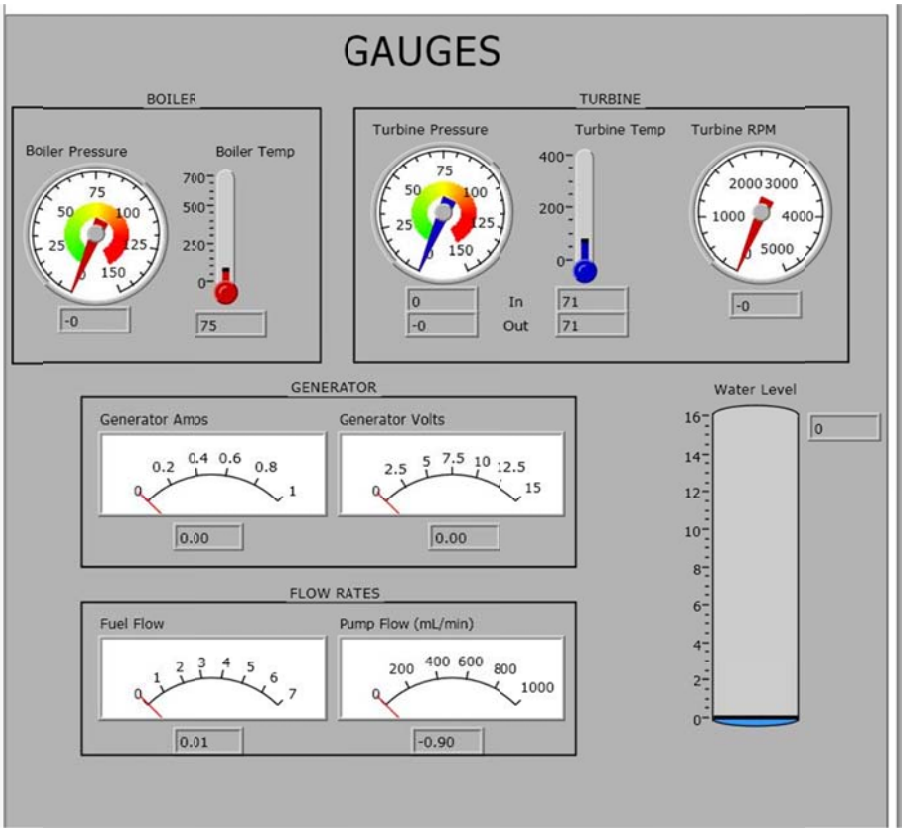


Figure 28. Gauges VI front panel.

3.5.3 Motor Control

The Motor Control subVI controls all the motor behavior and incoming UDP communications. When interacting with the *RankineCycler*, all of the input commands

are performed through this VI. The Acquisition and Control VI automatically activates the Motor Control front panel during the first loop iteration. The Motor Control VI front panel is shown in Figure 29.

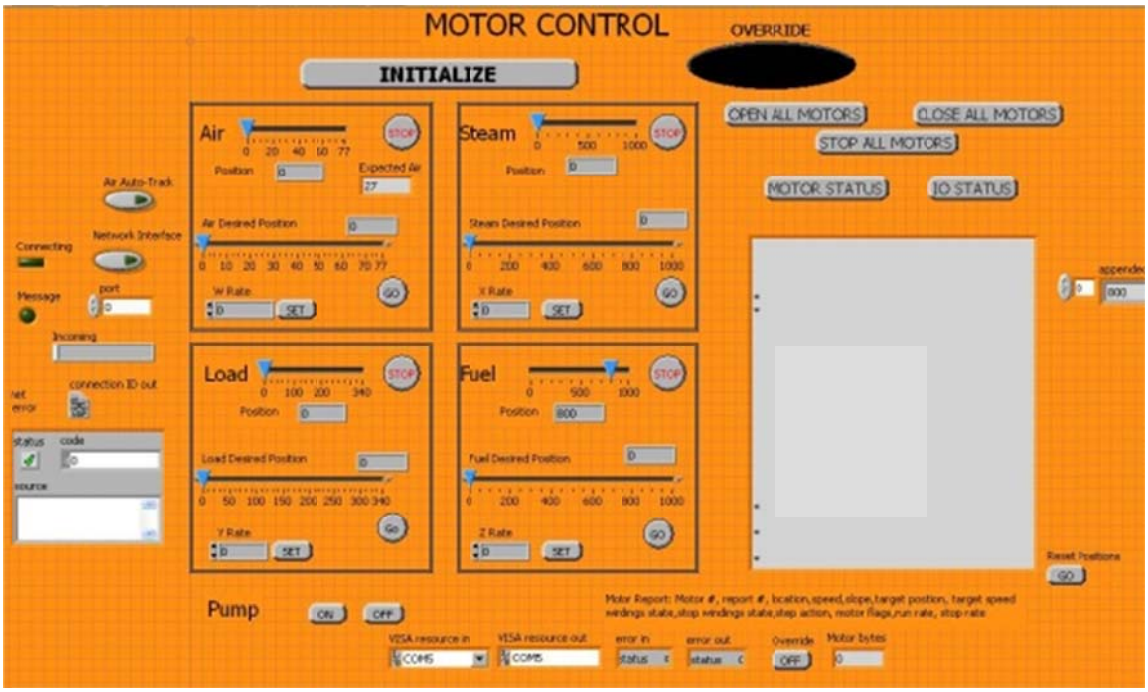


Figure 29. Motor Control front panel.

The center portion of the VI contains most of the functionality of the VI in the motor control sliders. The upper slider and text box in each quadrant displays the motor's current position. The lower slider and text box is used to input the desired motor position. When the "go" button is pressed, the applicable motor moves to the desired position. Each motor control box also includes an emergency stop button and box to modify motor speed. The air motor box also has additional functionality to allow for automated airflow adjustment.

The right side of the front panel has additional options for controlling all the motors simultaneously, along with an override indicator and a box displaying the control

board response. The lower portion of the VI has buttons to toggle the water pump on and off, and indicators for the serial connection status to the control board. The left side of the VI has the controls and indicators for the UDP communication logic.

The motor control VI is the most complicated VI in the system. It operates as an interrupt driven GUI. Because it is a subVI of the Acquisition and Control VI, the motor control program cannot operate in a continuous loop that is ideal for most interrupt driven programs. Each time it is called, the motor control VI reads and responds to 5 interrupts in 25 ms increments, then returns the motor positions to the acquisition and control VI. This behavior is described by the lower portion of the block diagram. On each of its 5 iterations, the VI checks for interrupts and takes the appropriate action. Most of the responses involve sending serial commands to the motor control board.

The block diagram for the Motor Control VI can be found in the appendix. The upper portion of the bock diagram is the UDP listening logic. When the network control button is toggled, the VI opens and begins listening on the designated UDP socket. Each time the motor control VI is called, it checks the socket for incoming data. If it finds any, it reads in the data until there is no more in the buffer, ensuring the result is the last command sent from the MAS. It then translates the text to step positions. These step positions are applied to the desired position sliders in the GUI and the interrupts for the corresponding motors are triggered. In effect, the network interface operates the GUI in the same manner a human user would, just at a much higher frequency.

The middle portion of the block diagram is dedicated to the automated air flow logic. The air motor controls the air flow into the combustion chamber of the boiler. If the air flow is too high, it extinguishes the flame. If it is too low, the burn becomes more

inefficient and generates carbon monoxide exhaust. To prevent this, the VI calculates a good air motor step position with the following equation:

$$air = 14 * \frac{fuel - 75}{725} + 21 - b$$

Where *b* is an adjustment factor to compensate for air pressure changes in the building where the *RankineCycler* resides. *b*=8 is typical. The other two variables represent step positions for their respective motors. The VI displays the result of this calculation in the expected air text box. If the air auto-track button is enabled, the system automatically applies the expected air value to the motor when the fuel motor position is changed.

CHAPTER FOUR

Testing Results

4.1 Objectives

The objective of this research is to implement a Multi-Agent Control System on a small laboratory scale power plant. This is the first step in an effort to implement a MACS on a larger power plant. Section 4.2 discusses load demand curves and their relevance to demonstrating results. Section 4.3 displays data demonstrating the *RankineCycler* and its MAS control system following a demand curve. The remaining sections describe other necessary steps that are required to achieve the results in Section 4.3, specifically, the neural network training process and the data collection required for training.

4.2 Load Demand Curves

Commercial power plants operate in two modes: either serving base load or load demand. Providing base load is typically done by larger plants and involves operating the plant near its peak power production capacity constantly. These plants provide power for constant loads on the grid like industrial plants and always on lighting. However, over the course of an average day there are extra demands put on the grid during certain periods. For example, air conditioning loads increase during the course of the day in warmer climates, then decrease at night. Other technologies, like on-demand water heaters, can also create spikes in demand at certain times of the day. To compensate for changing demands, grid operators issue demand curves based on predictive analysis to

smaller power plants with the capability to change operating output rapidly. These plants serve the load demand.

While MAS controllers could be used to serve both base load and load demand, controlling a plant to provide different outputs over time is the more challenging control problem. For this reason, demonstrating the MAS controlling the *RankineCycler* through a demand curve is the ultimate goal of the project. Figure 30 shows a typical load demand curve for the MAS.

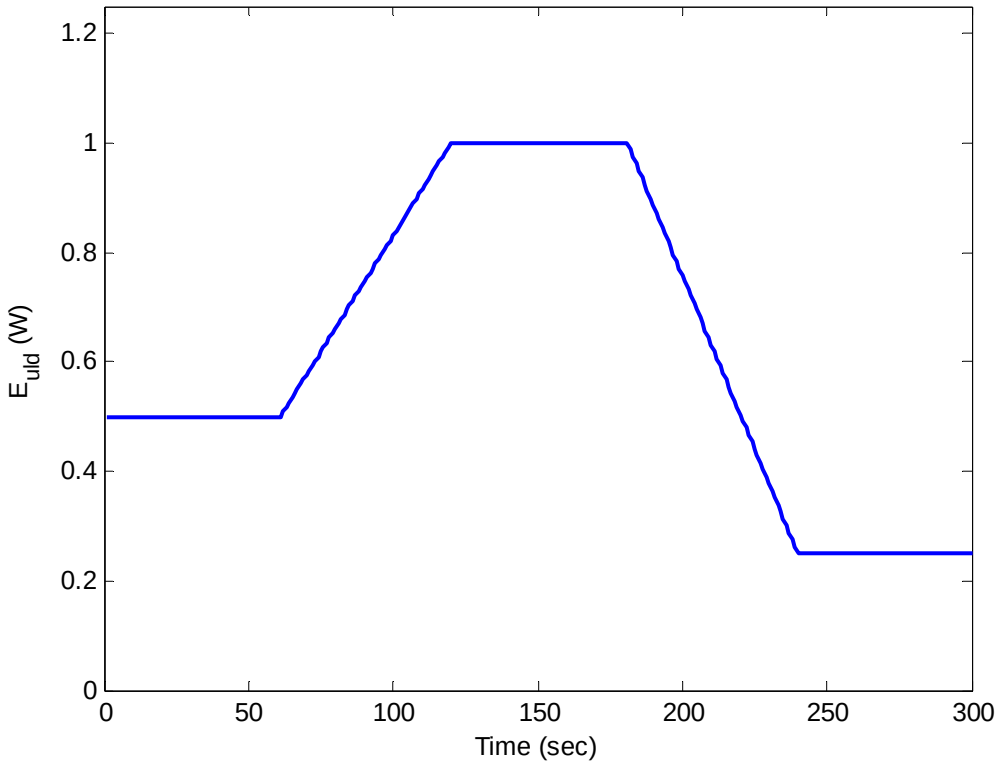


Figure 30. Typical Load Demand Curve.

The load demand curve describes target power output levels over five minutes. One important characteristic of the load demand curve is the gradual slope of power increases. Power plants are never asked to make sudden, severe jumps in output.

Regulations vary by location, but common practice limits power plants to increases below 10% of maximum output over the course of an hour [14]. While the theoretical maximum output of the *RankineCycler*'s generator is 15W, this would require a much larger boiler driving the turbine. The maximum practical output is around 5W, but the highest sustainable output is around 2W. Also, running the *RankineCycler* for multiple hours each testing period is impractical due to its size. Because of this the curve length was shortened to 5 minutes. The gradual increase in power demand was kept to preserve similarity to real demand curves. Another important characteristic of the demand curve is the starting power demand. The end of the warm-up cycle for the *RankineCycler* leaves the system running at an output of about 0.5 W. This power level was chosen to start the demand curve to prevent a shock to the *RankineCycler* when the MAS is engaged.

4.3 Results

The goal of this thesis is to demonstrate a MAS controlling the *RankineCycler* through a demand curve. Figure 31 demonstrates power and power demand vs. time over a short term demand curve. During this test the MAS is configured to provide performance similar to a boiler following control scheme, with the exception of a variable pressure demand.

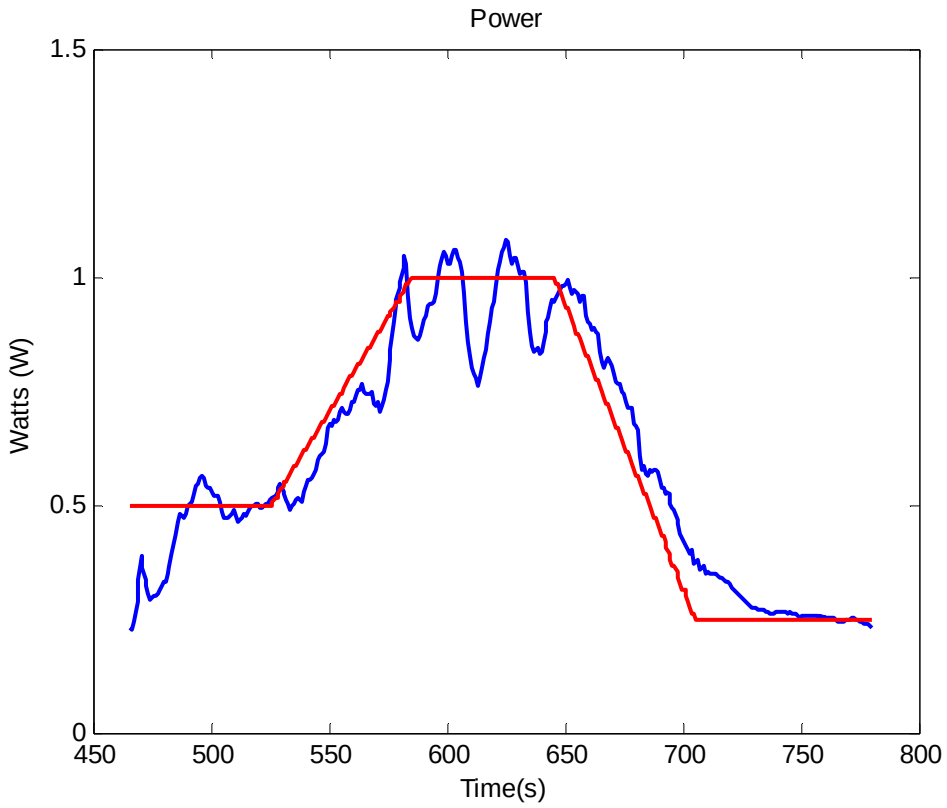


Figure 31. Power vs. Power demand Results.

The voltage level is one indicator of the success of the MAS. Commercial power plants are designed to meet a grid standard frequency, 60 Hz in the US for example. When the power plant is not meeting the demand, the frequency will differ from 60 Hz. Since the *RankineCycler* generator output is run through a diode rectifier, the best indicator of generator frequency is the voltage level across the load resistor. When designing the controller 5V was chosen as the target voltage. Proof of the MAS meeting all the design goals of the project is a constant 5V output regardless of load. Figure 32 displays the voltage and load resistance over the course of the test.

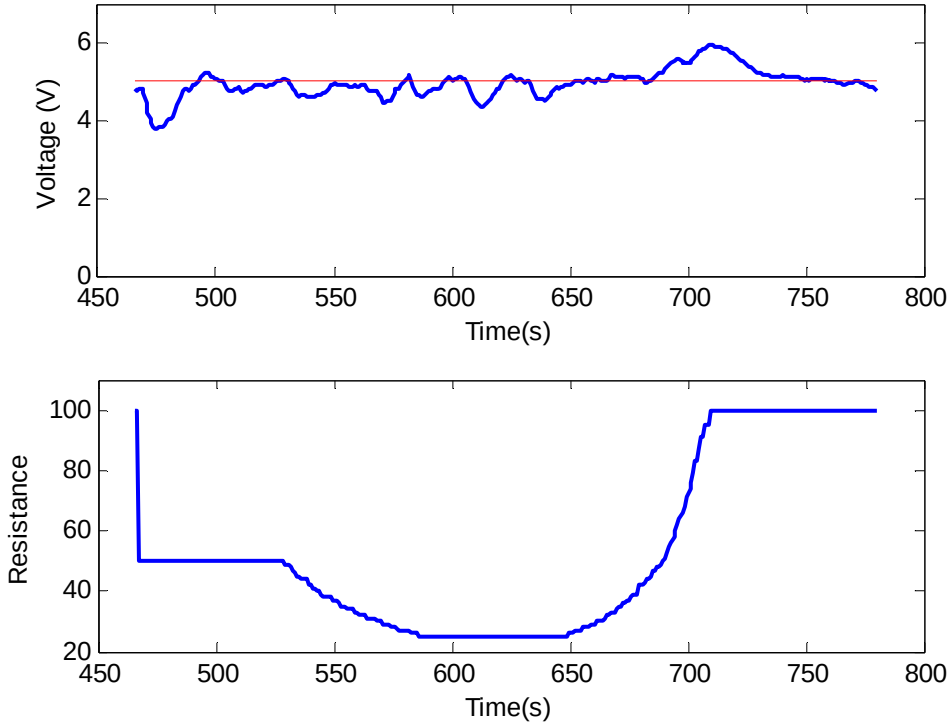


Figure 32. Voltage and load resistance during test.

The output voltage is relatively close to 5V during the test, with most of the variation occurring during load changes. There is a period of sustained overvoltage when the system is throttling down from the 1 W demand to 0.25 W.

Figures 33 and 34 are the fuel and steam valve positions over time and the Pressure and Pressure Demand over time. Figure 34 shows the major flaw with the *RankineCycler*. The pressure fails to meet the demand, or even increase at all despite the completely open fuel valve during the latter half of the test. This means the *RankineCycler* does not have enough input energy to achieve a steady state.

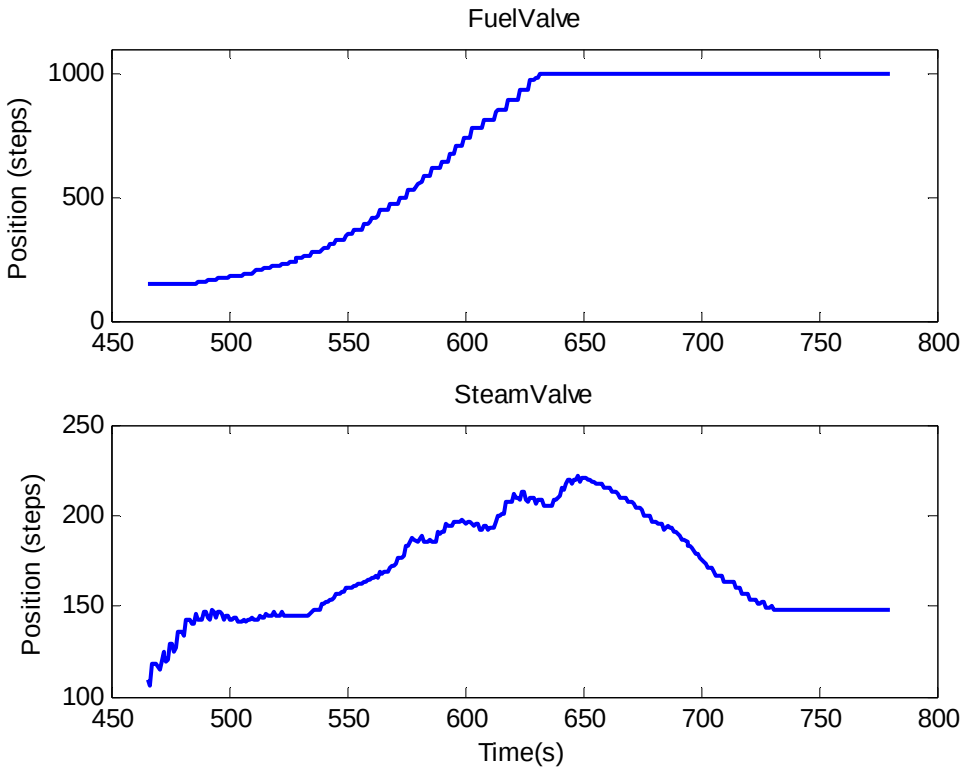


Figure 33. Fuel and Steam Valve positions.

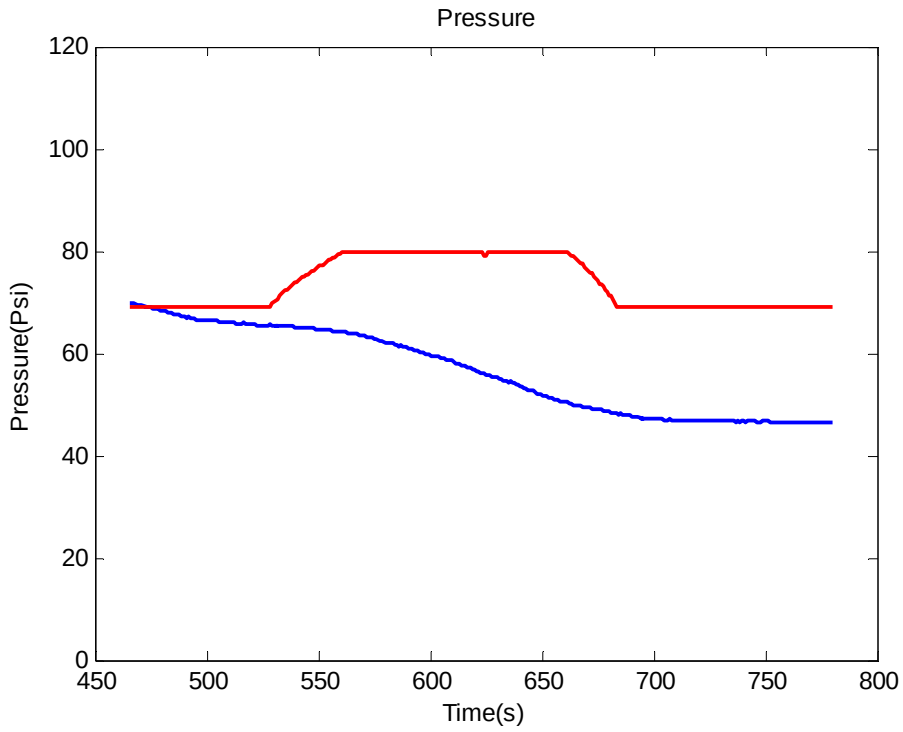


Figure 34. Pressure and Pressure Demand.

4.4 Neural Network Training Process

The MAS relies on two neural networks for the feedforward component of the control system. One is the steady-state predictive neural network that is used by the HPSO to find the optimal setpoints. The other is a dynamic neural network used to simulate the *RankineCycler*. All the work on these neural networks was performed using the MATLAB neural network toolbox. The following section describes the two neural networks and methods used to get the data necessary to train them.

4.4.1 Neural Network Configuration

The more important of the two neural networks is the dynamic neural network. It is used to simulate the *RankineCycler*'s operation over time. This simulation is then used to create the data for the steady-state network.

The dynamic neural network is a layer-recurrent network (LRN). LRNs are typically used in filtering and modeling applications. The trait that sets them apart from other types of neural networks is an internal feedback loop with a delay around each layer of the neural network. This attribute allows for the modeling of dynamic systems [15]. Figure 35 is a diagram of the layer recurrent neural network structure.

The neural network is configured with 11 nodes in the hidden layer and trained using Levenberg-Marquardt backpropagation. The training method was chosen because of speed considerations in the MATLAB neural network toolbox. A MATLAB program was created to obtain the data necessary to train the dynamic neural network; this program and its behavior are described in section 4.4.2.

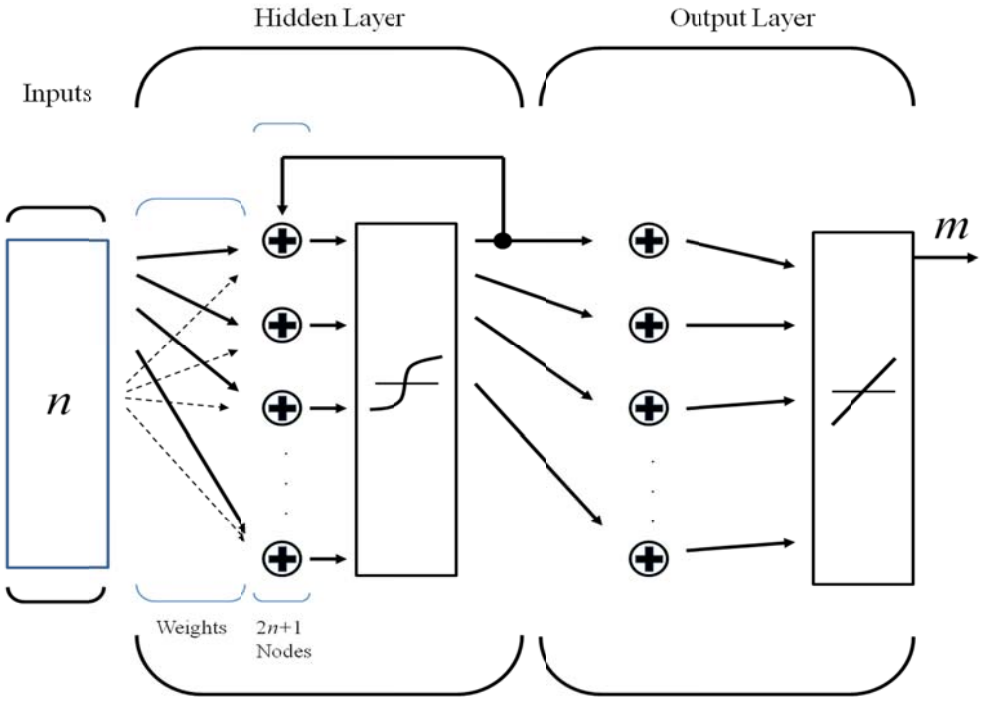


Figure 35. Layer Recurrent Neural Network.

The second neural network used by the MAS is the feedforward neural network used to predict the steady-state conditions of the *RankineCycler*. It is created using the standard MATLAB feed-forward type neural network with 7 nodes in the hidden layer. There is no feedback component between layers [16]. A diagram of the feedforward neural network structure is shown in Figure 36. Note the lack of connection across the hidden layer.

The feedforward neural network uses an input of steam valve position, fuel valve position, and temperature and computes a steady-state power and pressure. The data used to generate this network was obtained by running the dynamic neural network for 400 iterations and determining if the system is at a steady state over the last 100 iterations. A program cycles through the range of possible fuel and steam valve positions in steps of 5.

If a combination of valve position results in a steady state the data point is kept to train the feedforward neural network.

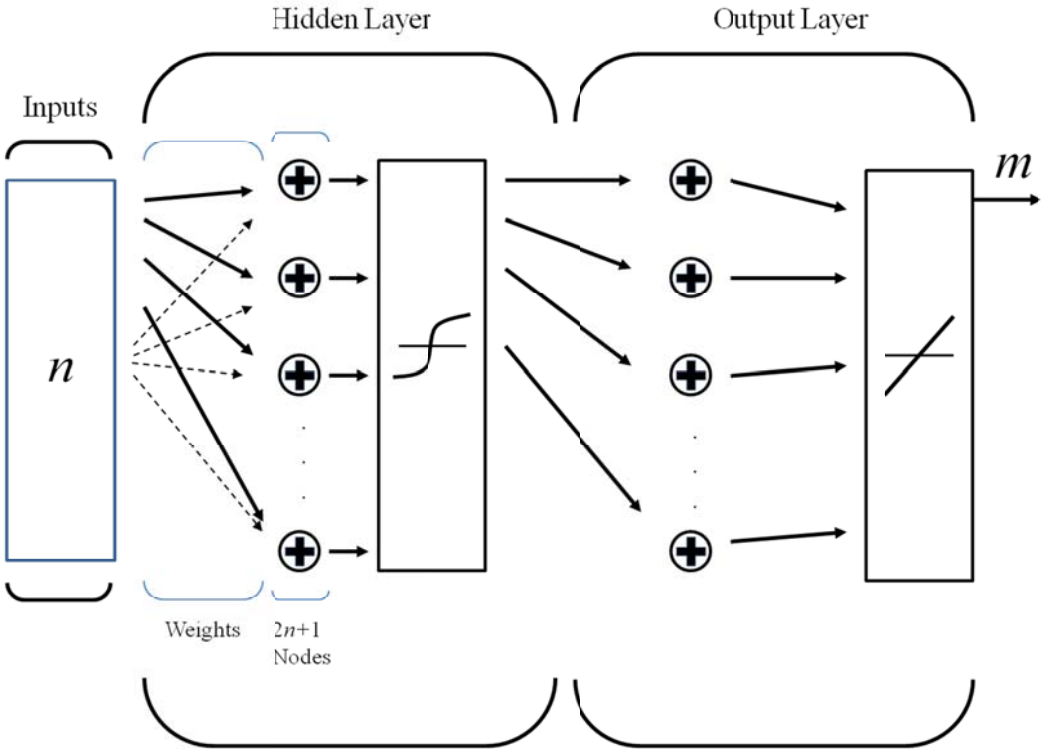


Figure 36. Feedforward neural network structure.

Figure 37 is the output of the dynamic network over 400 seconds with the fuel valve motor at 200 steps and the steam valve motor at 90 steps. The algorithm counts it as a steady state due to limited change in the final 100 seconds of the simulation.

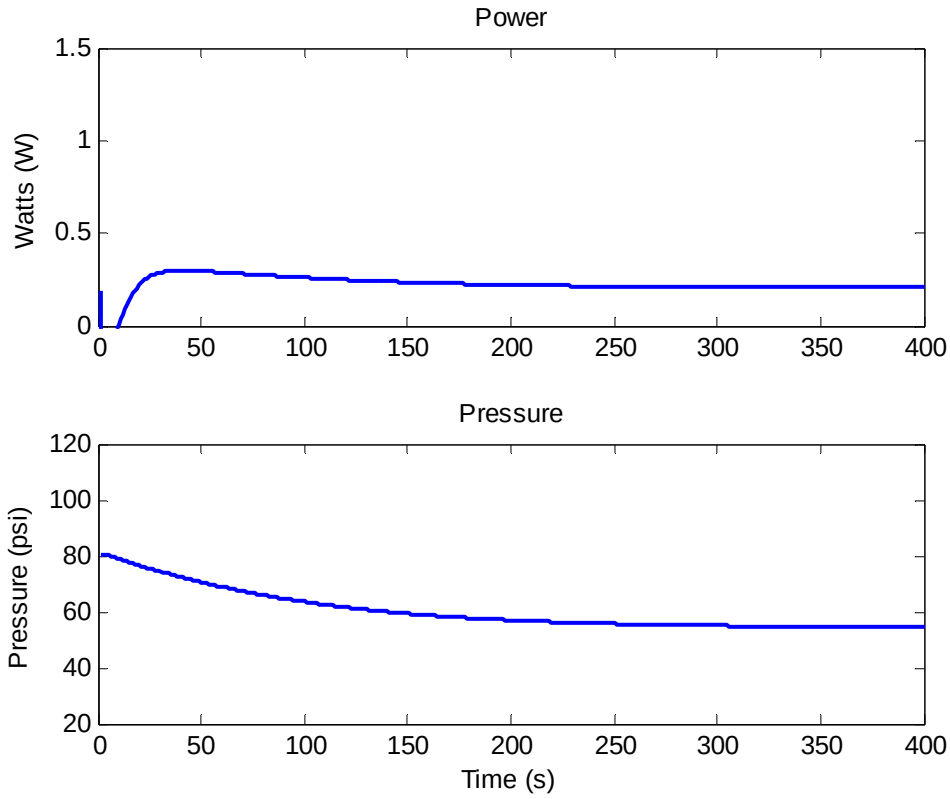


Figure 37. Simulated Steady State Evaluation.

4.4.2 Data Hunter

A key component of the MAS is a dynamic neural network used to predict the behavior of the *RankineCycler*. This dynamic neural network is used to train the feedforward neural network used in the feedforward agent. In a larger MAS this dynamic network would also be used to evaluate different gain values in a gain optimizer agent. In theory this neural network could be trained online while the MAS is controlling the *RankineCycler*, but it still needs a starting point to prevent unacceptable error. This means collecting data on the *RankineCycler* without the MAS controller and using the data to train a baseline dynamic neural network. A MATLAB program called *Datahunter.m* was created for this purpose.

The Data Hunter program is a MATLAB GUI program that uses the MAS messaging format to control and read data from the *RankineCycler*. Its algorithm obtains the status of the *RankineCycler* in real time and then applies a feedback loop to keep the *RankineCycler*'s output as close to 5V as possible at various power levels. A flowchart describing the algorithm is shown in Figure 38.

The program uses several timers to determine actions. Every 0.1s it checks for new data from the *RankineCycler*. When it receives data it stores it in a history variable and updates the GUI with the new information. Once a second it sends updated steam valve, fuel valve, and load motor positions to the *RankineCycler* using the MAS messaging protocol. If the program is in manual mode it send desired steam and fuel valve position based on user input from the GUI. If the program is in automatic mode it changes the steam and fuel valve position every 30 seconds. It decreases the fuel valve position by 100 steps each iteration, and increases the steam valve position by 10 steps and resets the fuel valve to its maximum if the program iterates past the fuel valves minimum position. In this case, the fuel valve minimum is at position 150 of 800; any position lower that this will blow out the flame in the boiler.

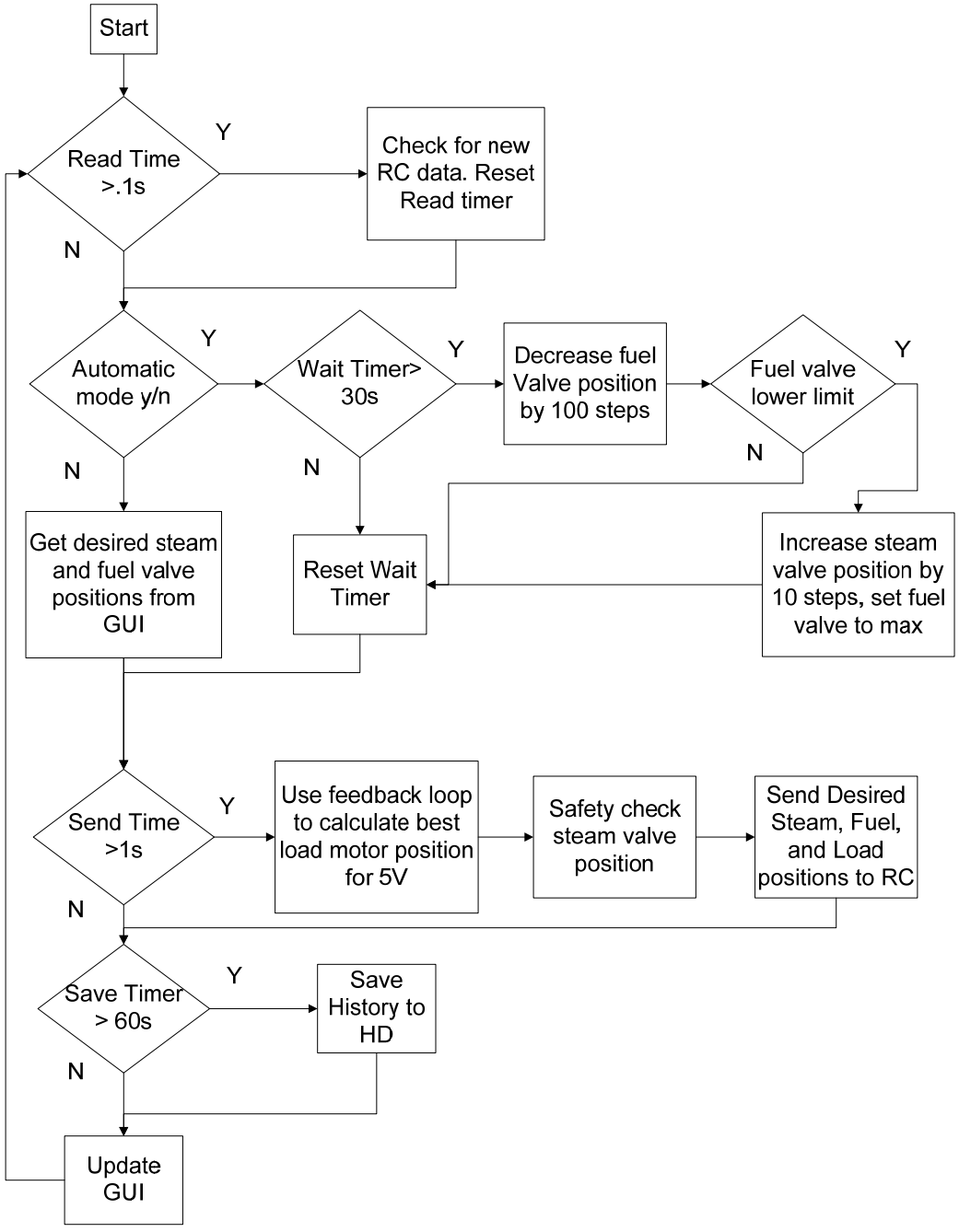


Figure 38. DataHunter Flowchart.

The desired load motor position is calculated by a feedback loop using the voltage as an error term. The loop is designed to keep the voltage as close to 5V as possible. This voltage level was chosen to keep the *RankineCyler* turbine rotation speed around 1700 RPM and prevent rotation speeds beyond recommended limits in the case of voltage swings. The design goal of the MAS is controlling the *RankineCycler* to produce a specific power at 5V, which means the data with which the dynamic neural network is trained is only valid when the Cycler's generator output is 5V. The feedback equation is a standard PI loop with a 20 second integral window.

$$Plm(n) = Plm(n-1) + Kp * e(n) + Ki * \int_{n-20}^{n} e(\tau)\, d\tau$$

Where $Plm(n)$ is the position of the load motor and $e(n) = 5 - V(n)$, with $V(n)$ the generator voltage. Kp and Ki can be adjusted through the GUI. Kp = -0.05 and Ki = -0.001 are the defaults.

Figure 39 is a screen capture of the Data Hunter program's GUI. The top-right side of the GUI includes the controls necessary to connect to the *RankineCycler* and a table displaying the most recent data packet received. The top-left side of the GUI displays the real-time state of the *RankineCycler*. Below the Plant info section is an interface for manually changing the valve and load positions. The setpoint hunt button starts the feedback control and the automatic searching algorithm. The gain values are updated each iteration from the textboxes on the right of the button. This means gains can be changed while the algorithm is running. The axes on the bottom of the GUI display a 15 second voltage history to help determine the success of the feedback loop.

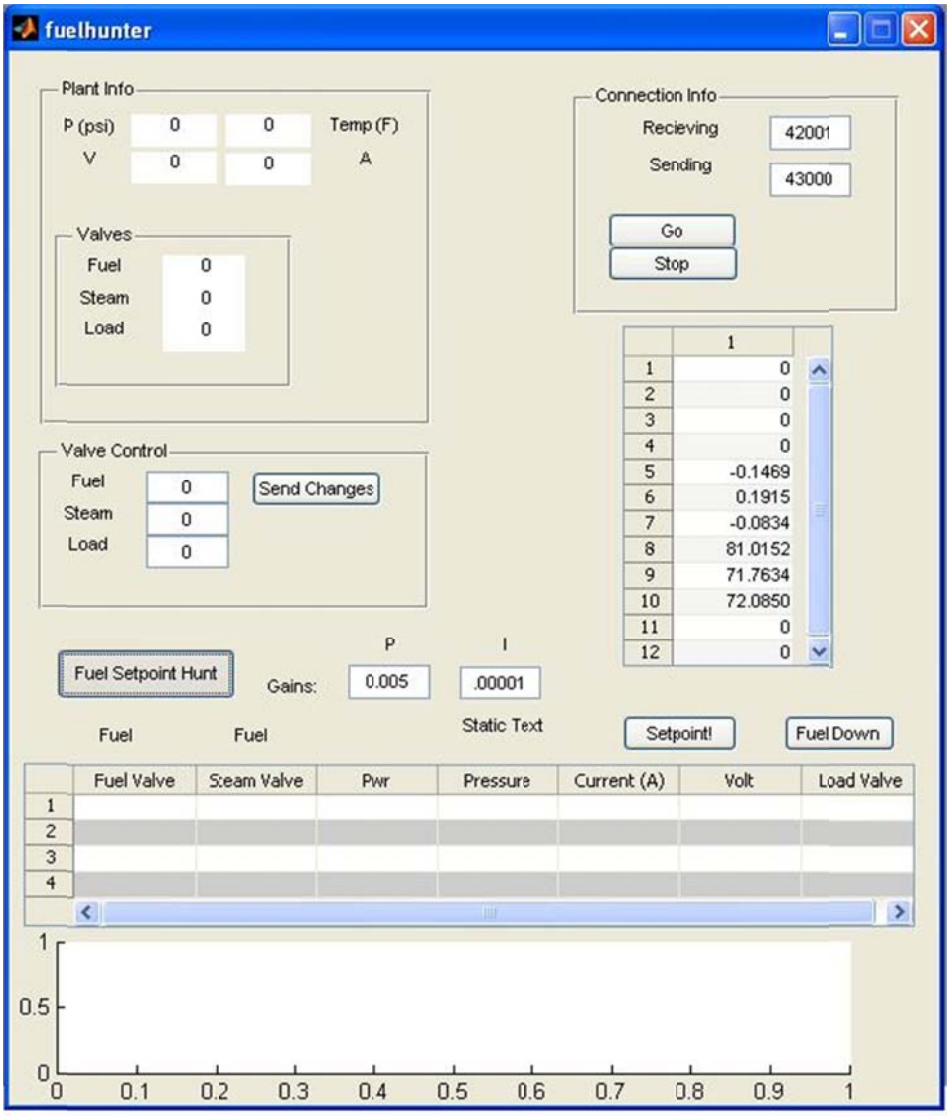


Figure 39. Data Hunter GUI.

The GUI also has a table for recording potential steady-state points along with buttons to add them manually, but in practice this method did not generate a large enough quantity of quality setpoints. In the later versions of the MAS setpoints found in this manner were discarded in favor of setpoints generated from a dynamic network trained by the complete data history of a data hunter test run.

Figure 40 is some sample output from a manually controlled run of the Datahunter program. Multiple manual and automatic runs were combined to form the

data used to train the dynamic network used for the MAS. There are several likely state-state positions in the output, which can be observed as sections where the voltage, power, and pressure graphs are flat. Output around t=700s, 1700s, and 2500s show steady-state conditions.

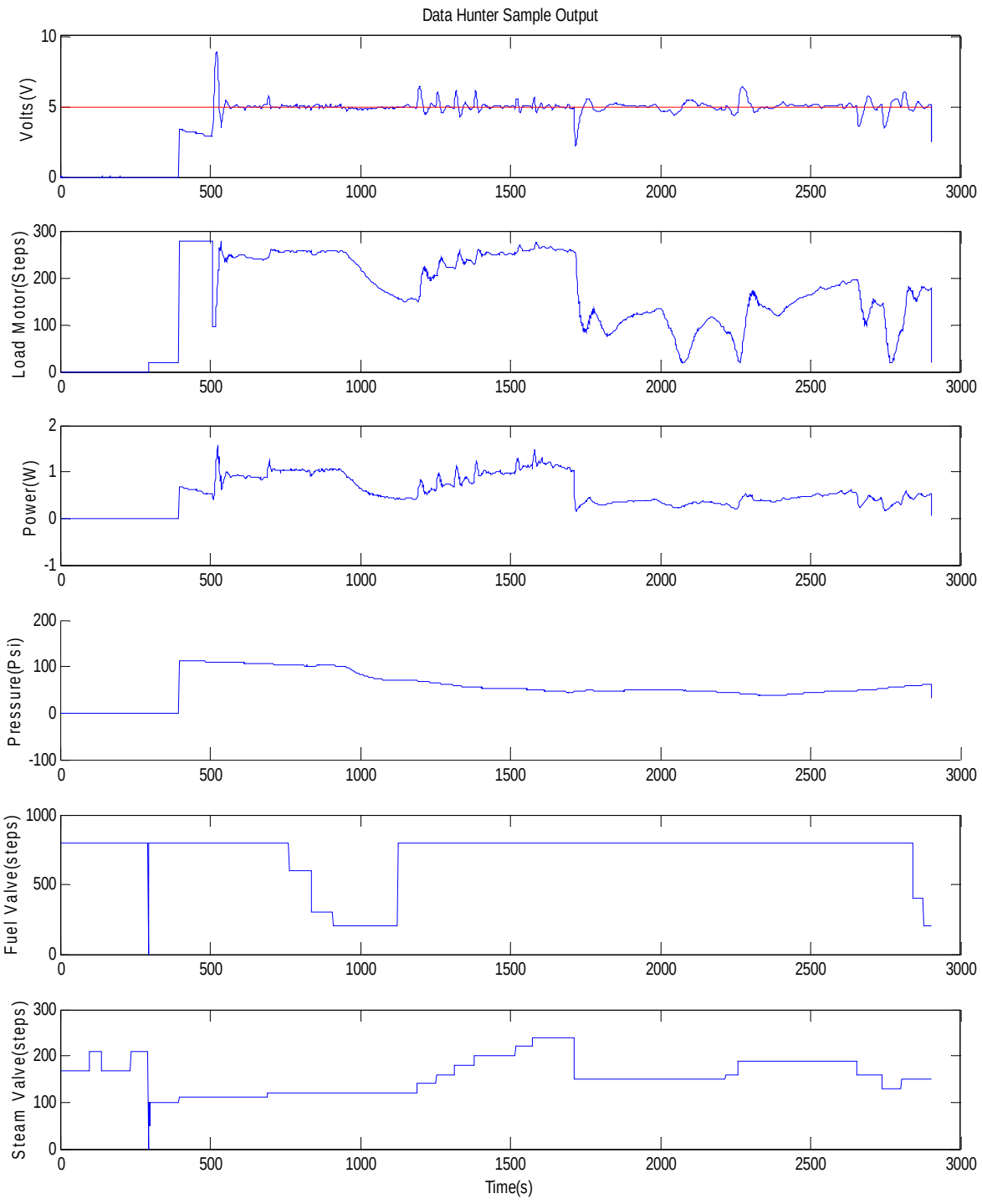


Figure 40. DataHunter Sample Output.

CHAPTER FIVE

Conclusion

The results in Chapter Four demonstrate the operation of the Multi-Agent Control System by matching the *RankineCycler*'s output to a load demand curve. This is the overall goal of the research project and it can be considered a success. However, there are some problems that appeared during the course of the research that prevented the implementation of the entire Multi-Agent System. Most of these issues are linked to the size of the *RankineCycler* and the feedwater system modifications. While these issues make the result slightly less than ideal, the fact that the MAS still manages to control the *RankineCycler* successfully is at minimum a testament to the design flexibility and potential of MAS-based control schemes. This chapter discusses some of the flaws in the project as well as points of interest for further research.

5.1 RankineCycler Size

The MAS as a concept is really meant for implementation on large-scale power plants. This means its use as a controller for the *RankineCycler* is largely a proof of concept exercise. The advantages of a MAS over a less flexible controller are more obvious on a system that can operate over longer time periods and has more variation in possible valve positions.

An important issue that presented difficulties in the research is operational time. This stems from the fact that the *RankineCycler* is not designed to operate indefinitely. Several agent tasks, like the Neural Network Trainer Agent and the Gain Optimizer

Agent, are designed to improve performance over a significant length of time. While the modifications to the *RankineCycler* feedwater system increased its operation time from about 30 minutes to more than an hour; refilling the boiler while in operation has a significant impact on the steam generation and therefore power output of the system. Any neural network trained using data from the refilling process would give invalid results when the pump is not active. Because of this, development of the Neural Network and Gain Optimizer agents was not completed. It is important to note, however, that the algorithm that would be used in a NN training agent was used to generate the dynamic and feedforward neural networks offline. It is just impractical to do this while the MAS is running

The fuel and boiler system are also poorly sized. The fuel input range and boiler design prevent the *RankineCycler* from operating effectively in a turbine following manner. While the fuel input could be improved by increasing the pressure in the fuel lines, this would come at the cost of eliminating most of the safety mechanisms build into the *RankineCycler* before modification. If the *RankineCycler* could function at a higher fuel input rate without damaging the system is unknown.

5.2 Feedwater System and Water Level Sensor

The water level sensor design is flawed. Due to the boiler size, the water level presented in the sight glass varies depending on steam valve position. This makes using it in the control system impractical. Also, difficulties finding a floater that could last inside the sight glass for extended periods make the instrument itself unreliable.

The feedwater system also does not work ideally, but this is also due to boiler size and design. The water pump creates a large change in the boiler pressure when activated,

and restrictions on the use of the pump make it less than ideal for reducing this shock on the system. While this seems to suggest the feedwater system injects water into the system too quickly, it is also inadequate for proper temperature management in the boiler. Even with the pump working at its maximum flow rate the boiler can still surpass recommended temperatures. It is important to note the boiler is not designed to accommodate water flow, unlike commercial power plants. Additionally, vibrations caused by the pump make the water flow sensor data unreliable.

5.3 Points of Interest for Further Research

This project began with the design and implementation of the MAS in a purely simulated environment using a well developed power plant mathematical model. More information on the simulated MAS can be found in the work by Jason Head[17]. Adapting the MAS to the *RankineCycler* was done seperately to show viability outside of the virtual space. Analyzing the differences between the simulated MAS and the *RankineCycler* MAS is important because it reveals topics that deserve further study. The following section describes some of the challenges to the *RankineCycler* implementation and how they differed from the simulated version. To simplify the discussion the simulated MAS implementation will be referred to as SimMAS and the *RankineCycler* version RCMAS.

5.3.1 Control System Frequency

The MAS has several internal "frequencies" that had to be changed for the RCMAS. For example, the feedback agent in the SimMAS calculates and applies new valve positions every 0.1s. In the RCMAS this is not possible because of DAQ delays and motor communication and spin times. The RCMAS applies new valve positions

every 1s, which was chosen because it is close to the maximum speed the *RankineCycler* hardware can handle.

The frequency of feedforward calculations is also worth consideration. The RCMAS feedforward agent is designed for real-time inputs, which is rarely used in commercial power plants. It can handle both the power demand curve and one single power demand point, and applies the last received demand immediately. For example, if the system is following a demand curve and a new demand is input to the interface agent, the RCMAS will control the *RankineCycler* output to the new input and ignore the rest of the curve. It does this by running the optimization algorithm after, and only after, a change in demand. This results in one optimization for each flat section of a demand curve, regardless of length, and new optimized settings every second during a ramping up or down in demand. The one second period during the ramps is limited by neural network evaluation times in MATLAB. The SimMAS feedforward agent optimizes much more frequently, and several times over the duration of one flat section on a demand curve. It does this by taking advantage of code optimization to improve neural network evaluation times and by optimizing for future points in the demand curve. These changes were not implemented into the RCMAS due to time considerations.

The design for a Neural Network Trainer also has an evaluation period associated with it, which prevented implementation on the RCMAS. The NN trainer algorithm has a built in delay because network training is computationally expensive. While the computation time is not a problem due to the parallel nature of the MAS, the data collection time became a problem for the RCMAS. The NN trainer on the SimMAS collects plant data for the real-time equivalent of an hour before updating the networks.

The *RankineCycler* cannot run for periods of that length due to the limitations of the fuel and water systems. Shortened data collection periods can make the neural networks perform worse after training.

The question of update frequency and their overall effect deserves more study. Particularly in the case of the feedback agent, as it should affect stability. Each of the agents has its own period of operation that has an effect on the other agents. The optimal frequency for each agent appears to change depending on the size of the MAS controlled plant.

5.3.2 Gain Selection

Feedback gain selection proved to be one of the more challenging parts of the RCMAS implementation. The feedback gains have a significant effect on the performance of the control system. The RCMAS can display both boiler-following and turbine-following behavior by changing the gain matrices. This attribute never appeared during the development of the SimMAS, and the SimMAS performed adequately using decoupled feedback loops. While this is probably attributed to plant differences like the size of the pipe between the boiler and the turbine, the *RankineCycler*'s sensitivity to gains also suggests a method for gain optimization deserves further study. Based on observation during the tuning process, the optimal gains are dependant not only on performance goals like overshoot, rise time, settling time and stability, but also boiler pressure, steam valve position, water level, boiler water input rate, and current power output. The large number of variables makes development of the Gain Optimizer agent a challenge.

While a Gain Optimizer agent was developed for the SimMAS, it has yet to be adapted to the RCMAS. There are significant complications to the process, as the Gain Optimizer for the SimMAS relies on a highly accurate dynamic neural network, which is hard to verify for the RCMAS.

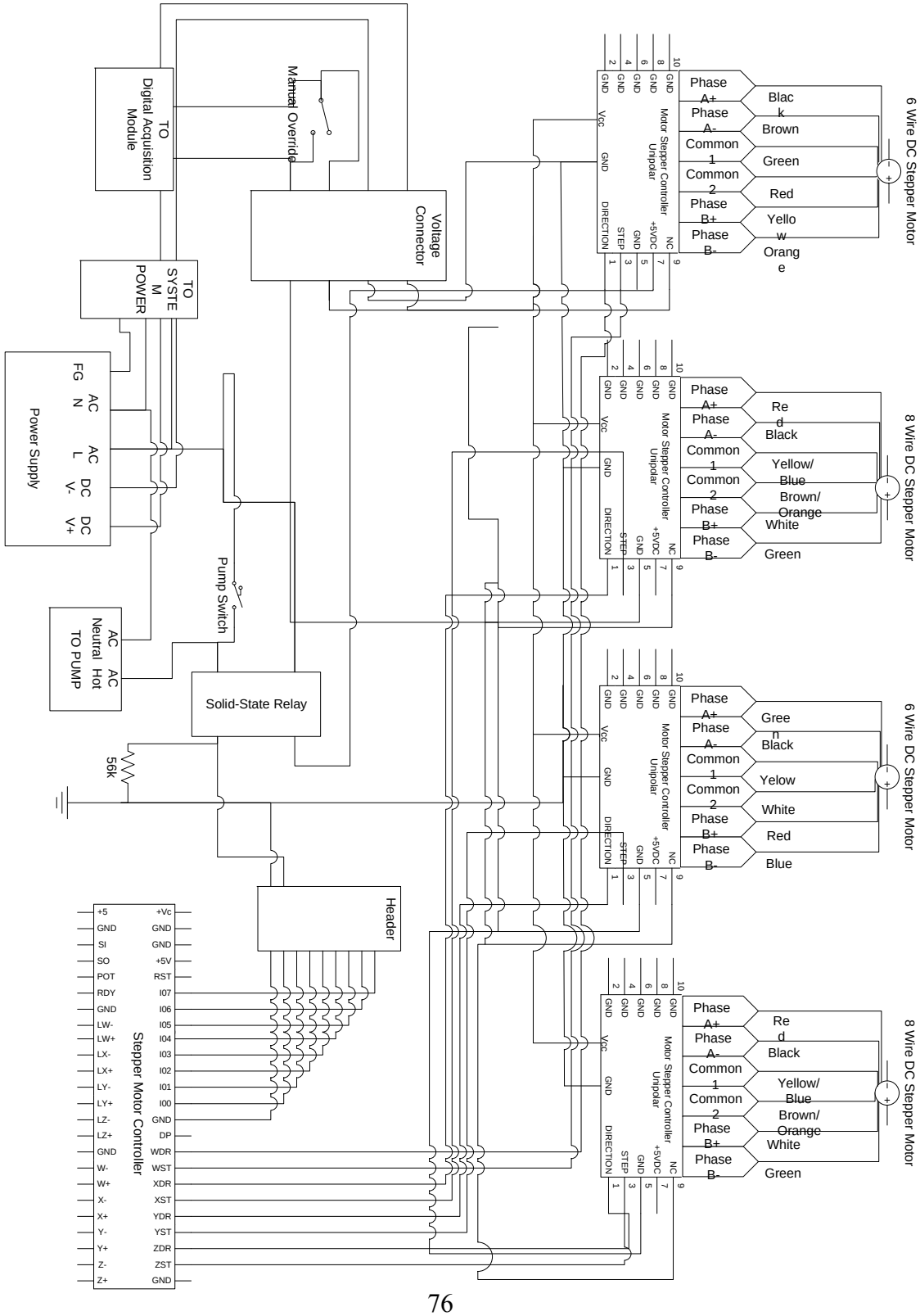
5.3.3 JADE

MATLAB proved to be less than ideal for the implementation of the MAS. MATLAB was chosen because of familiarity and access to neural network tools. While this meant the mathematical challenges of the system were easier to overcome, the parallelization and communication implementation was much harder to fully realize. Any MAS larger than the one on the RCMAS or the SimMAS would become significantly more difficult to implement in MATLAB for various reasons. There are other development environments for multi-agent systems, one of which is the Java Agent Development Framework (JADE). Importing the algorithms to JADE or a similar platform might reveal more possibilities for the MAS implementation.

APPENDICES

APPENDIX A

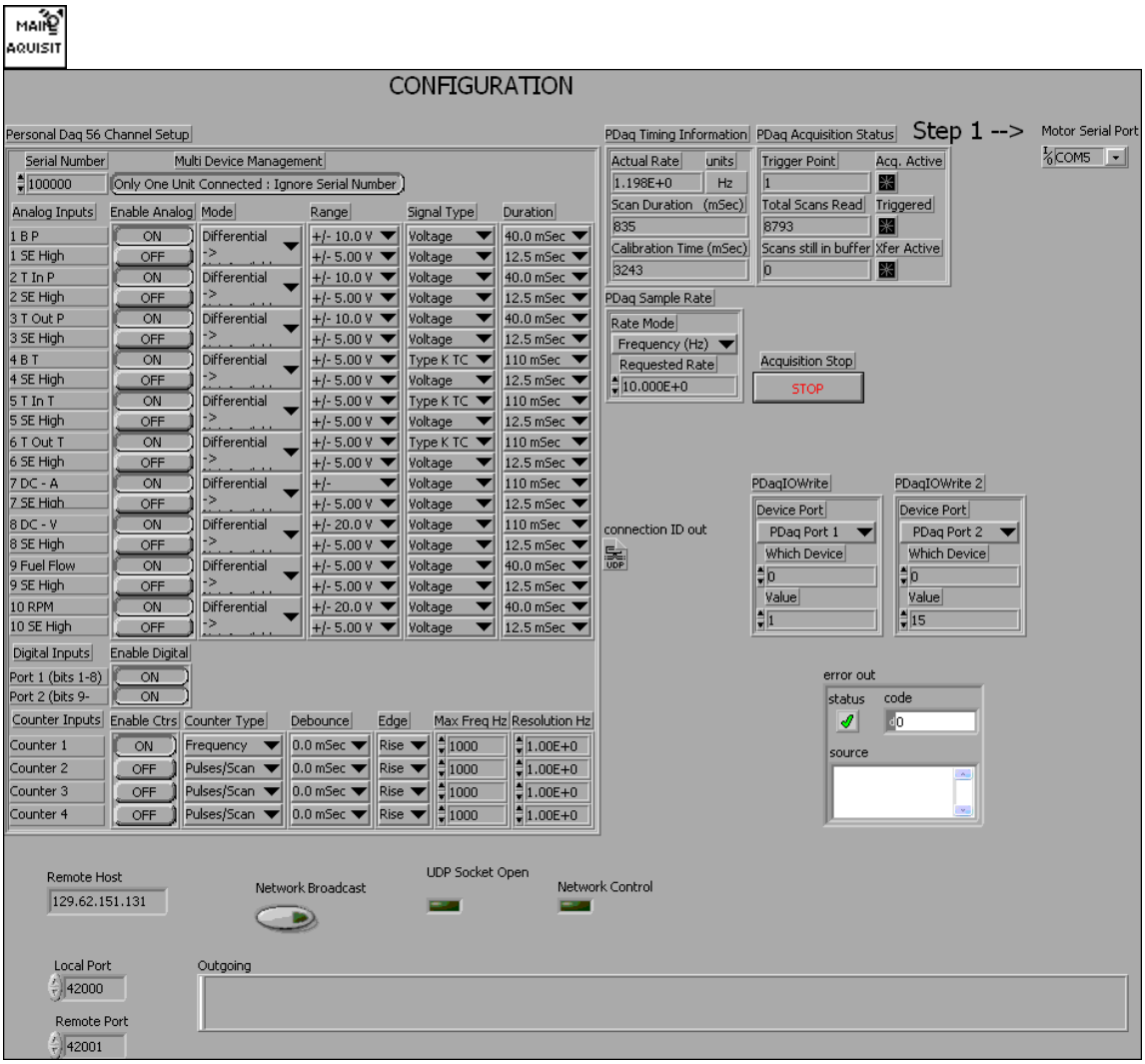
Motor Wiring Diagram

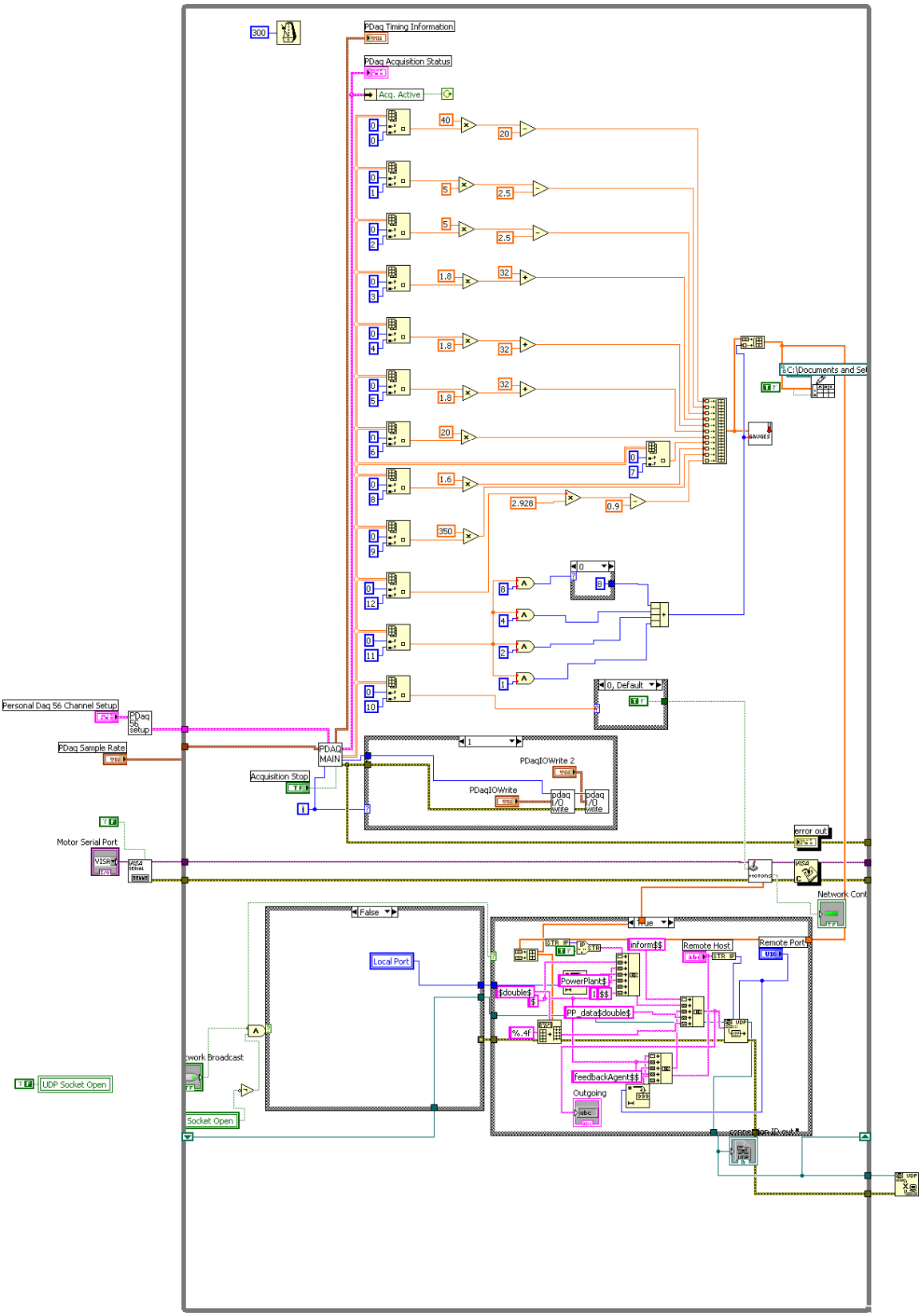


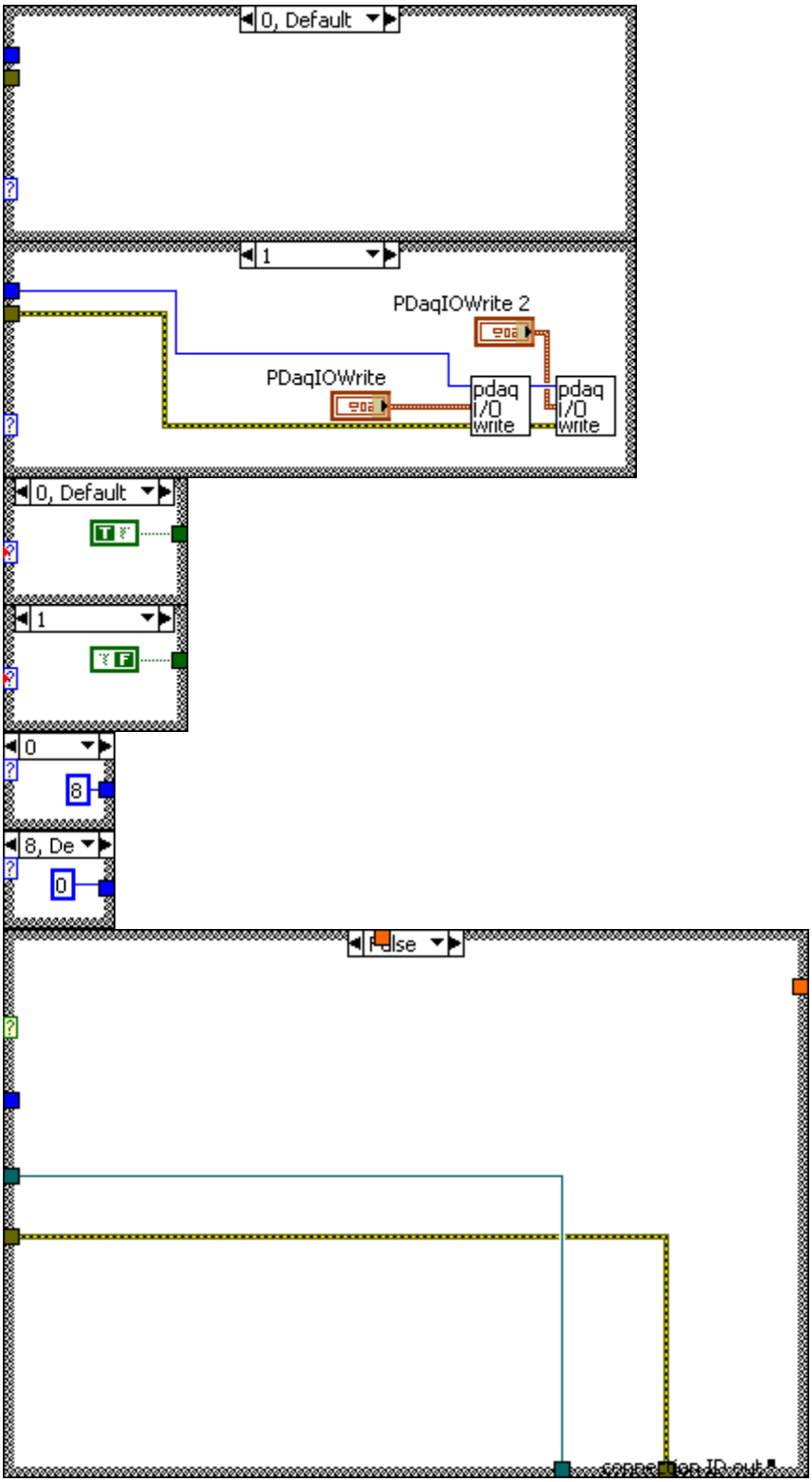
APPENDIX B

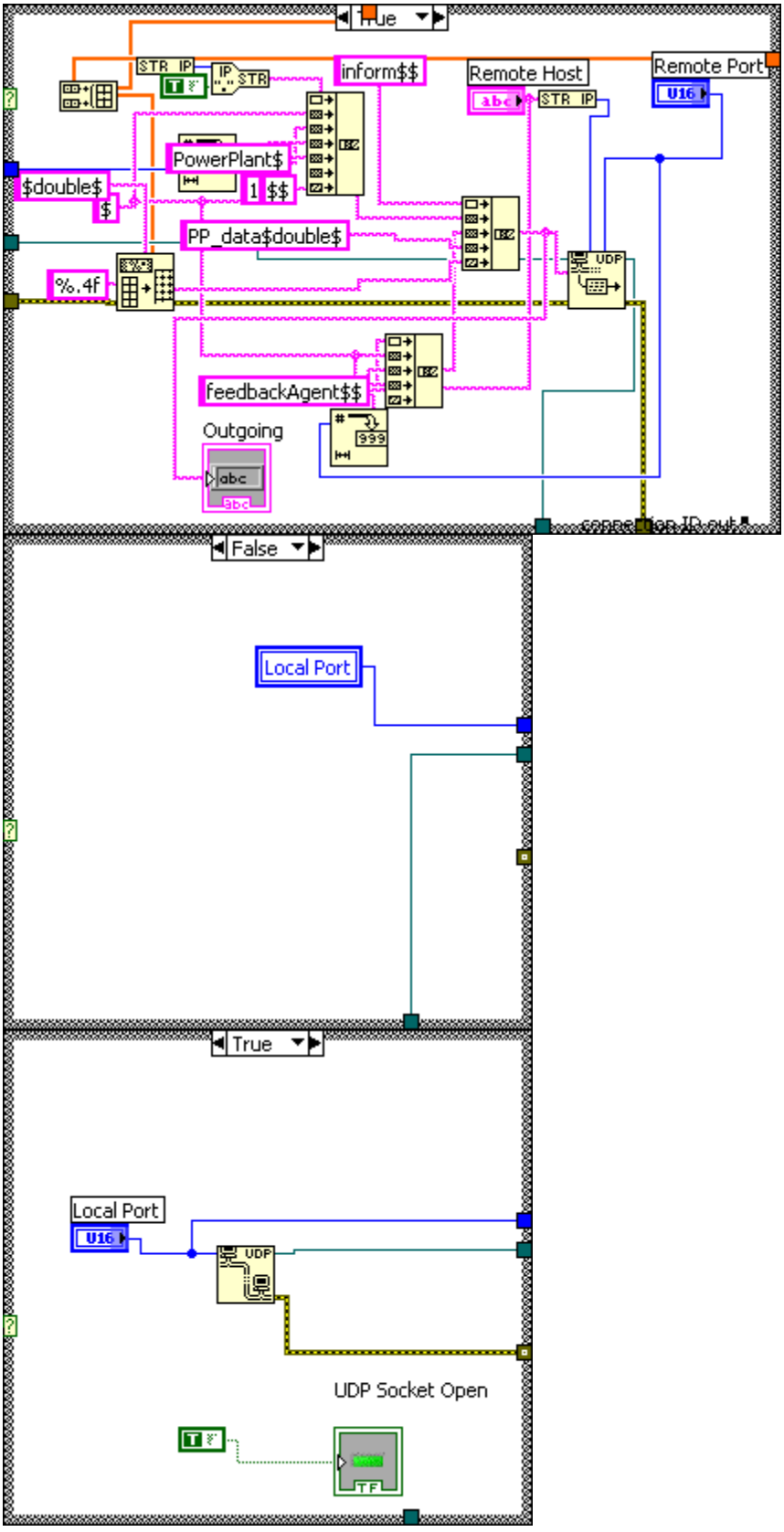
LABView Documentation

B.1 Rankine_Acquisition_and_Control.vi




300
PDaq Timing Information
PDaq Acquisition Status
Acq. Active
40
20
5
2.5
5
2.5
1.8
32
1.8
32
1.8
32
20
1.6
2.928
0.9
350
0
8
4
2
1
0, Default
Personal Daq 56 Channel Setup
PDaq 56 setup
PDaq Sample Rate
PDAQ MAIN
Acquisition Stop
1
PDaqIOWrite 2
PDaqIOWrite
pdaq I/O write
pdaq I/O write
Motor Serial Port
VISA
error out
Network Cont
False
Local Port
True
Inform$$
Remote Host
Remote Port
$doubles$
PowerPlant$
PP_data$doubles$
%.4f
FeedbackAgent$$
Outgoing
UDP Socket Open
Socket Open


0, Default
1
PDaqIOWrite 2
PDaqIOWrite
pdaq
I/O
write
pdaq
I/O
write
0, Default
1
0
8
8, De
0
False


True
STR IP
IP
STR
inform$$
Remote Host
Remote Port
U16
PowerPlant$
1 $$
$double$
$
PP_data$double$
%.4f
UDP
feedbackAgent$$
Outgoing
999
abc
connection ID out
False
Local Port
True
Local Port
U16
UDP
UDP Socket Open
TF

PdaqIOWrite.vi

Personal Daq Main Acquisition Module2.VI

Motor Control.vi

Write To Spreadsheet File (DBL).vi

Write To Spreadsheet File.vi

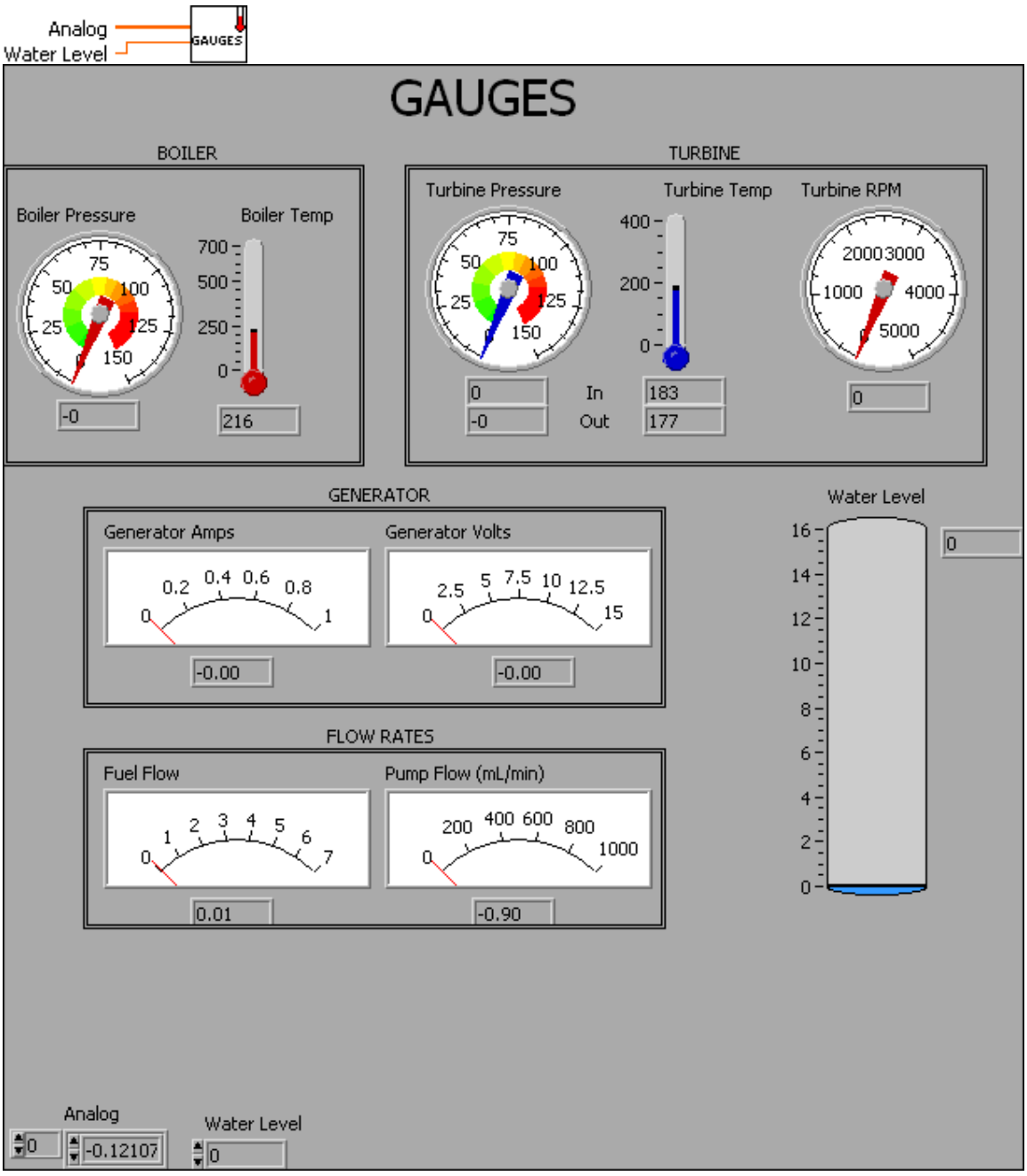
Gauges.vi

VISA Configure Serial Port (Instr).vi

VISA Configure Serial Port

Personal Daq 56 Channel Setup.VI

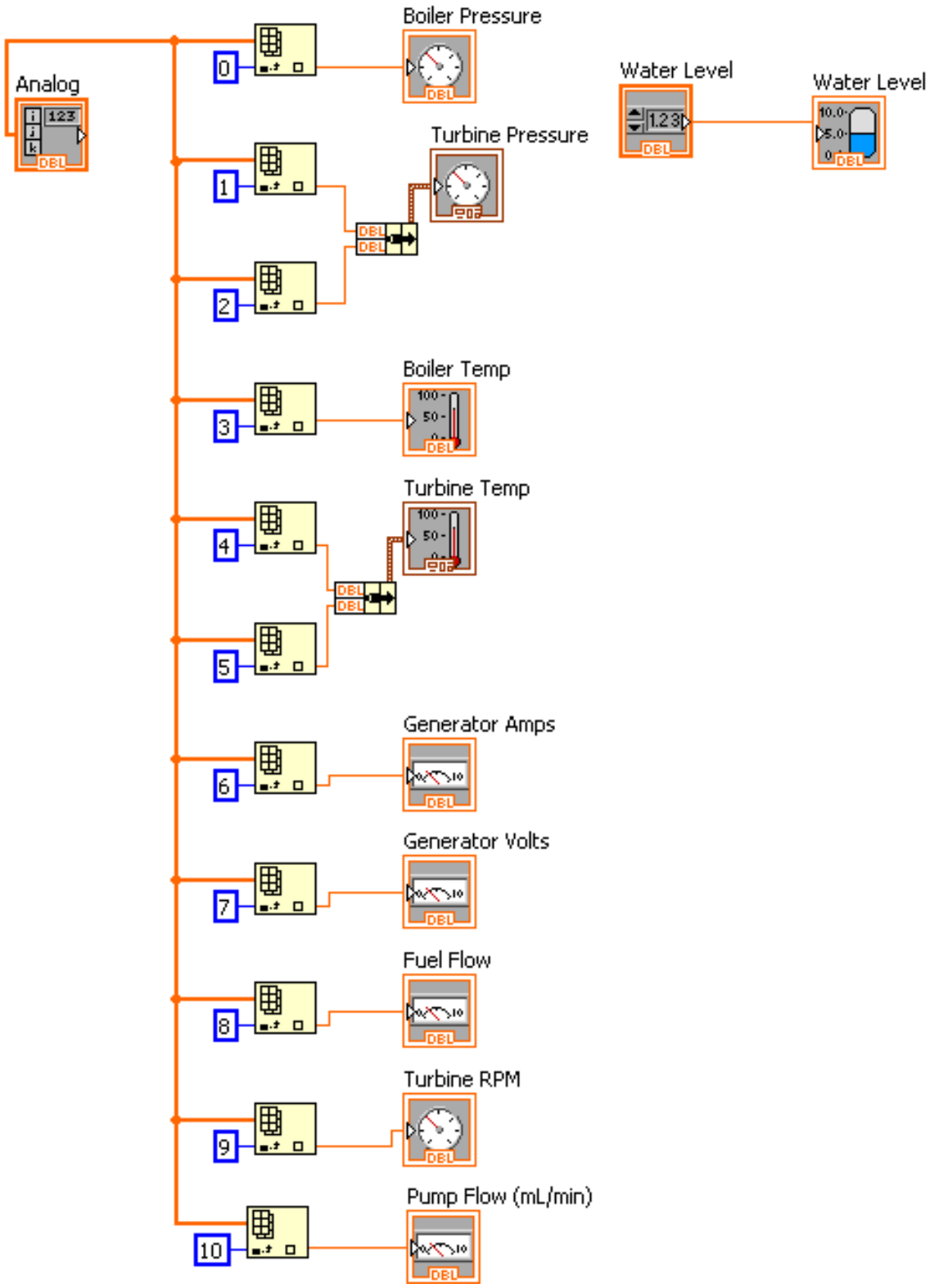
B.2 Gauges.vi



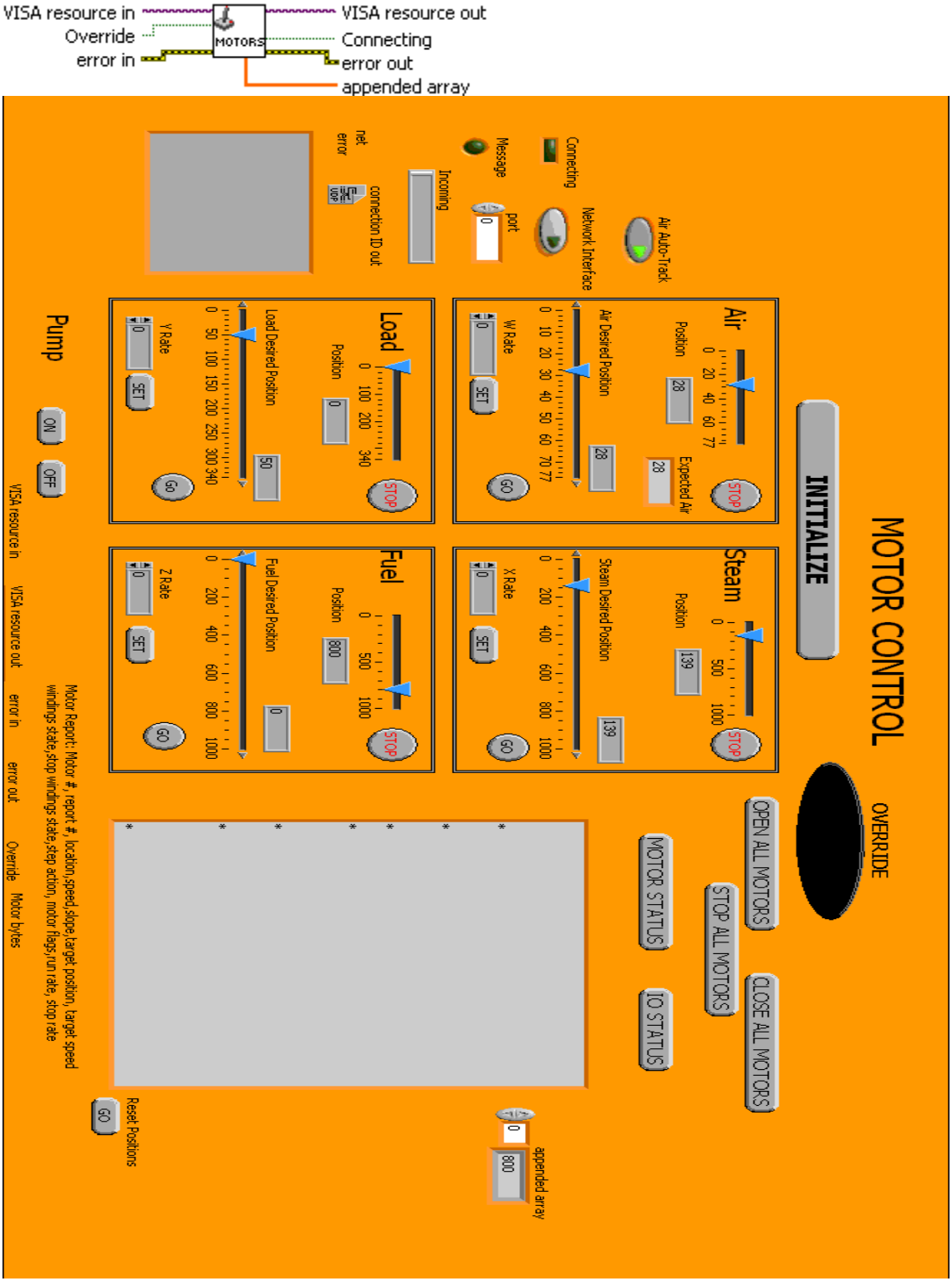
Analog

Numeric

Water Level


Analog
0
Boiler Pressure
1
Turbine Pressure
2
Water Level
Water Level
3
Boiler Temp
4
Turbine Temp
5
6
Generator Amps
7
Generator Volts
8
Fuel Flow
9
Turbine RPM
10
Pump Flow (mL/min)

B.3 Motor Control.vi



I/O VISA resource in

error in error in can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

Right-click the error in control on the front panel and select Explain Error or Explain Warning from the shortcut menu for more information about the error.

status status is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.

Right-click the error in control on the front panel and select Explain Error or Explain Warning from the shortcut menu for more information about the error.

code code is the error or warning code.

Right-click the error in control on the front panel and select Explain Error or Explain Warning from the shortcut menu for more information about the error.

source source describes the origin of the error or warning.

Right-click the error in control on the front panel and select Explain Error or Explain Warning from the shortcut menu for more information about the error.

Override

VISA resource out

error out error in can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

Right-click the error in control on the front panel and select Explain Error or Explain Warning from the shortcut menu for more information about the error.

status status is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.

Right-click the error in control on the front panel and select Explain Error or Explain Warning from the shortcut menu for more information about the error.

code code is the error or warning code.

Right-click the error in control on the front panel and select Explain Error or Explain Warning from the shortcut menu for more information about the error.

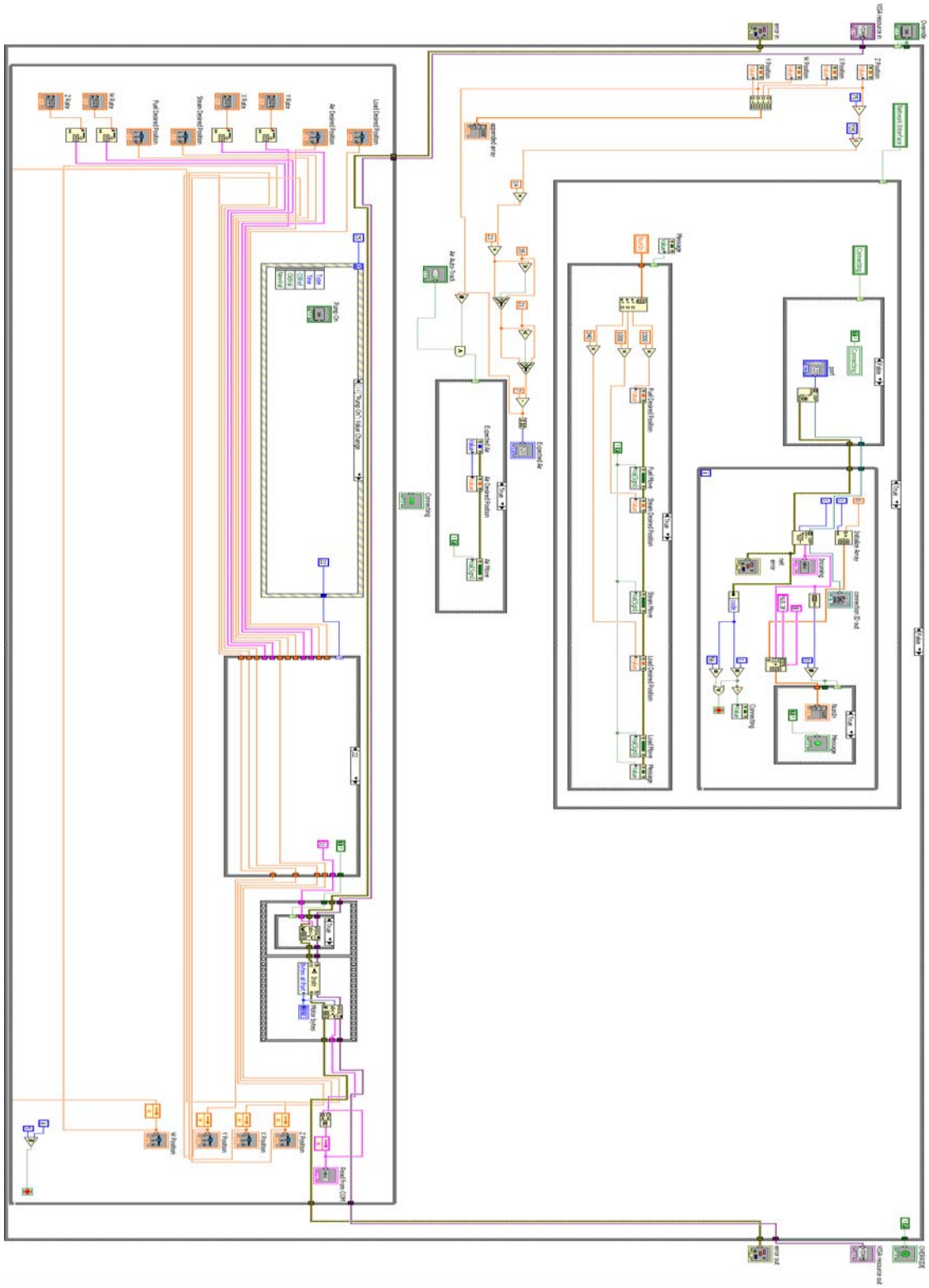
source source describes the origin of the error or warning.

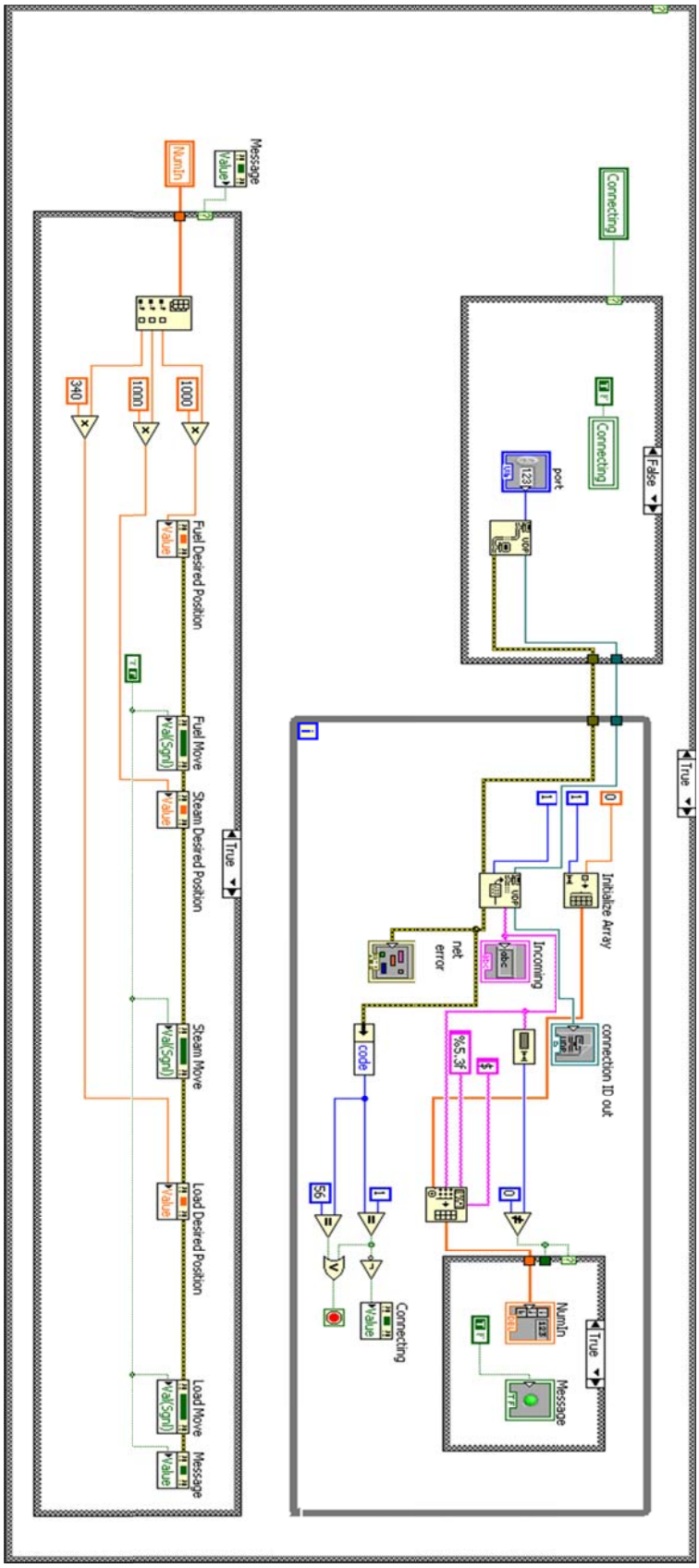
Right-click the error in control on the front panel and select Explain Error or Explain Warning from the shortcut menu for more information about the error.

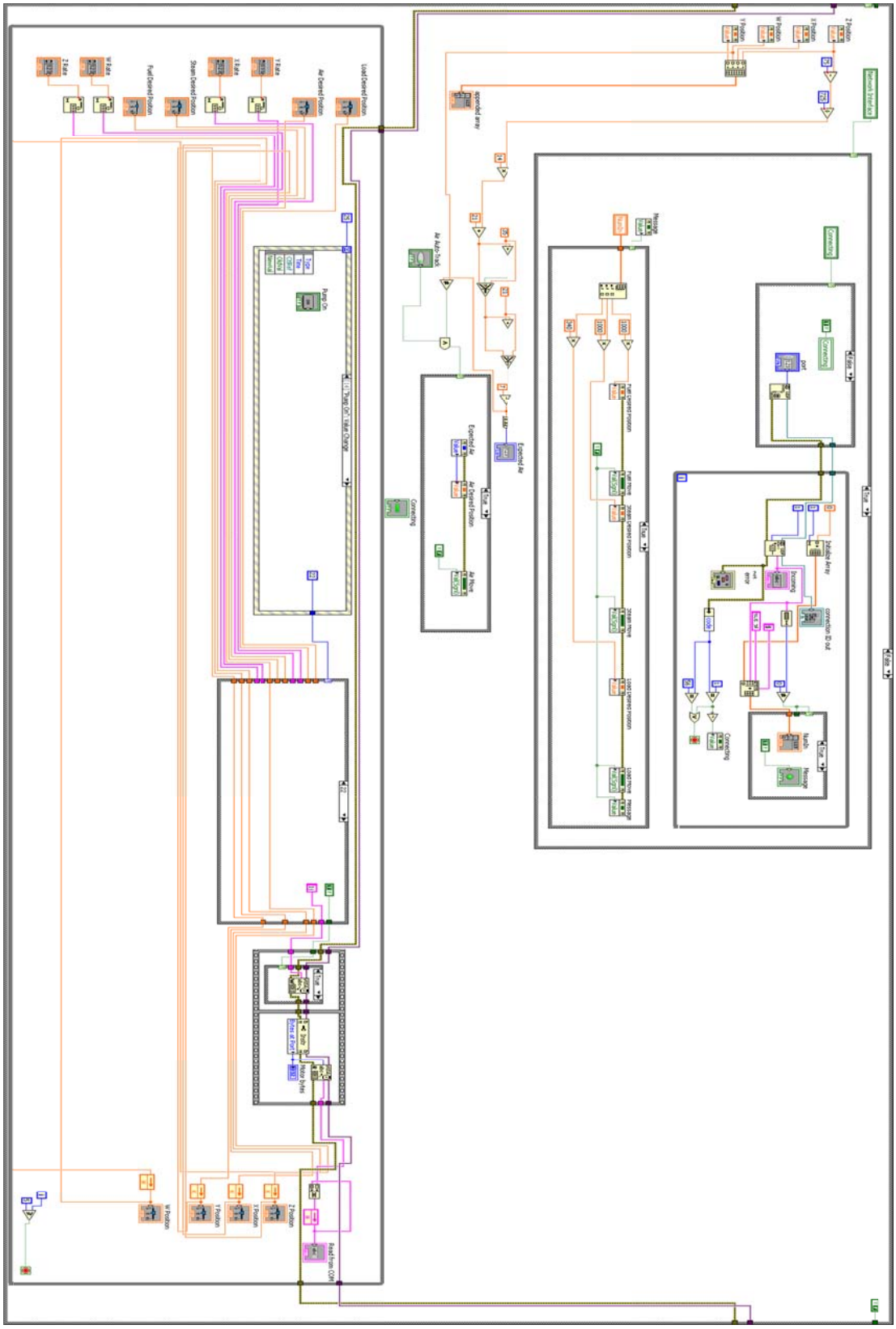
appended array

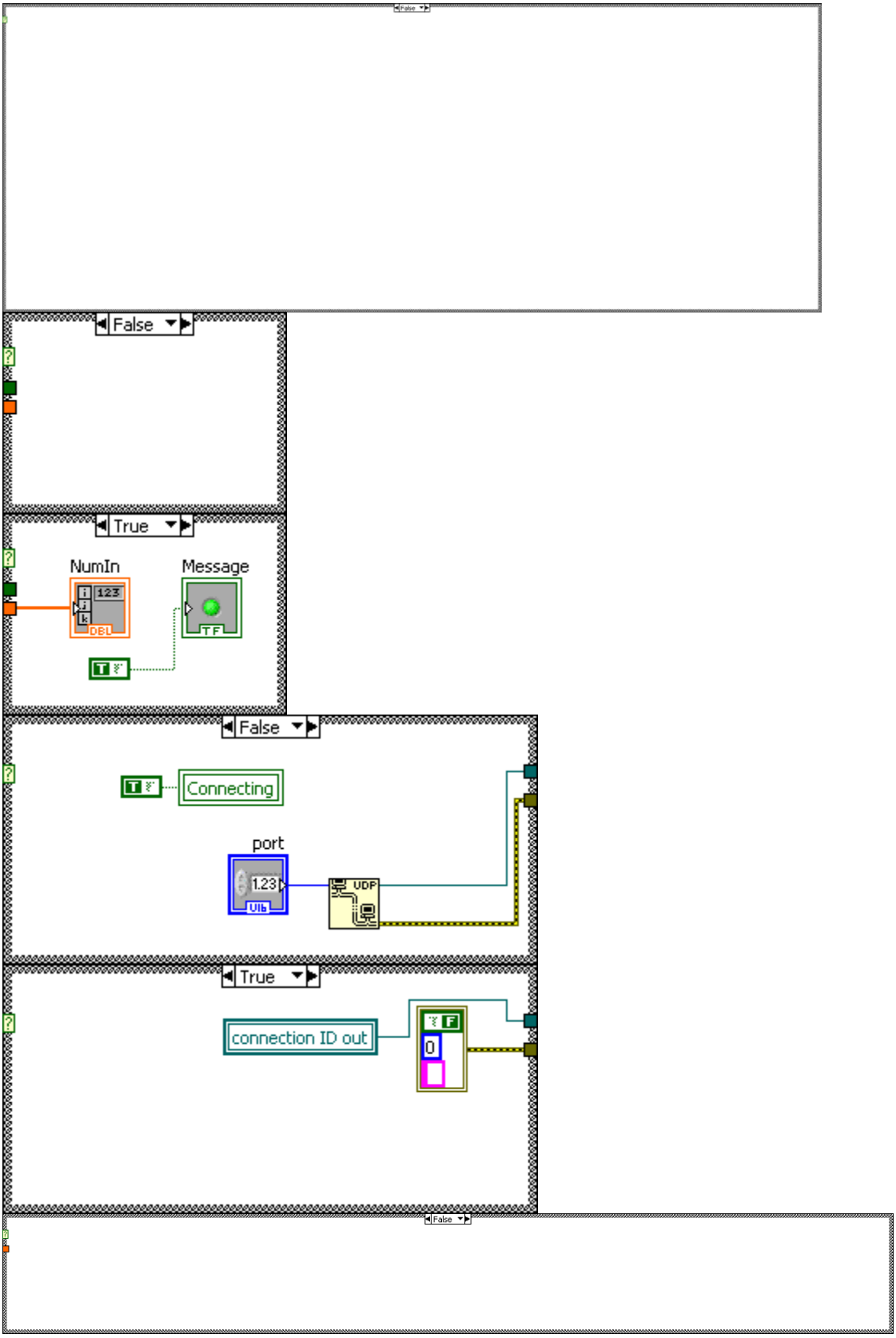
Fuel Desired Position

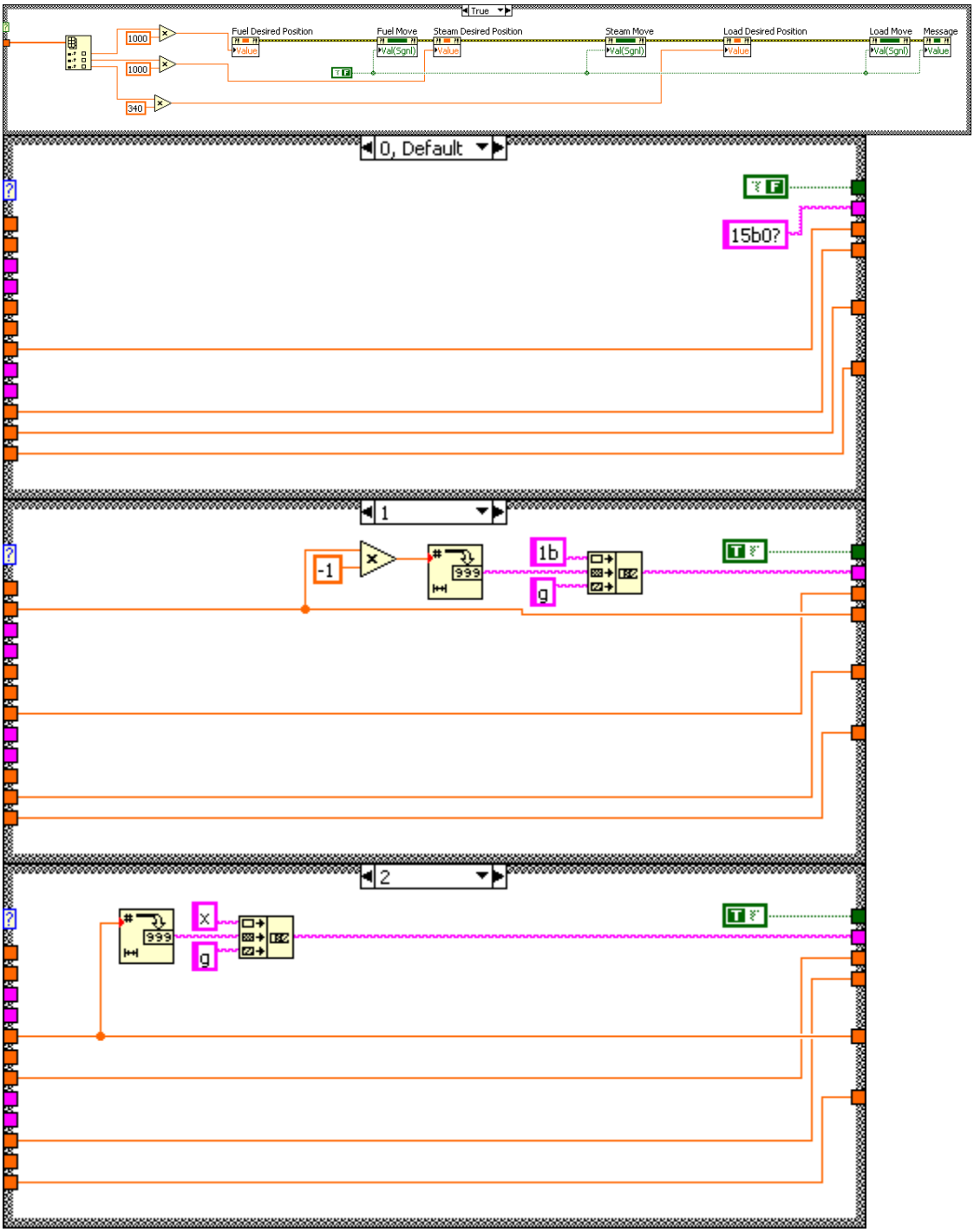
Connecting

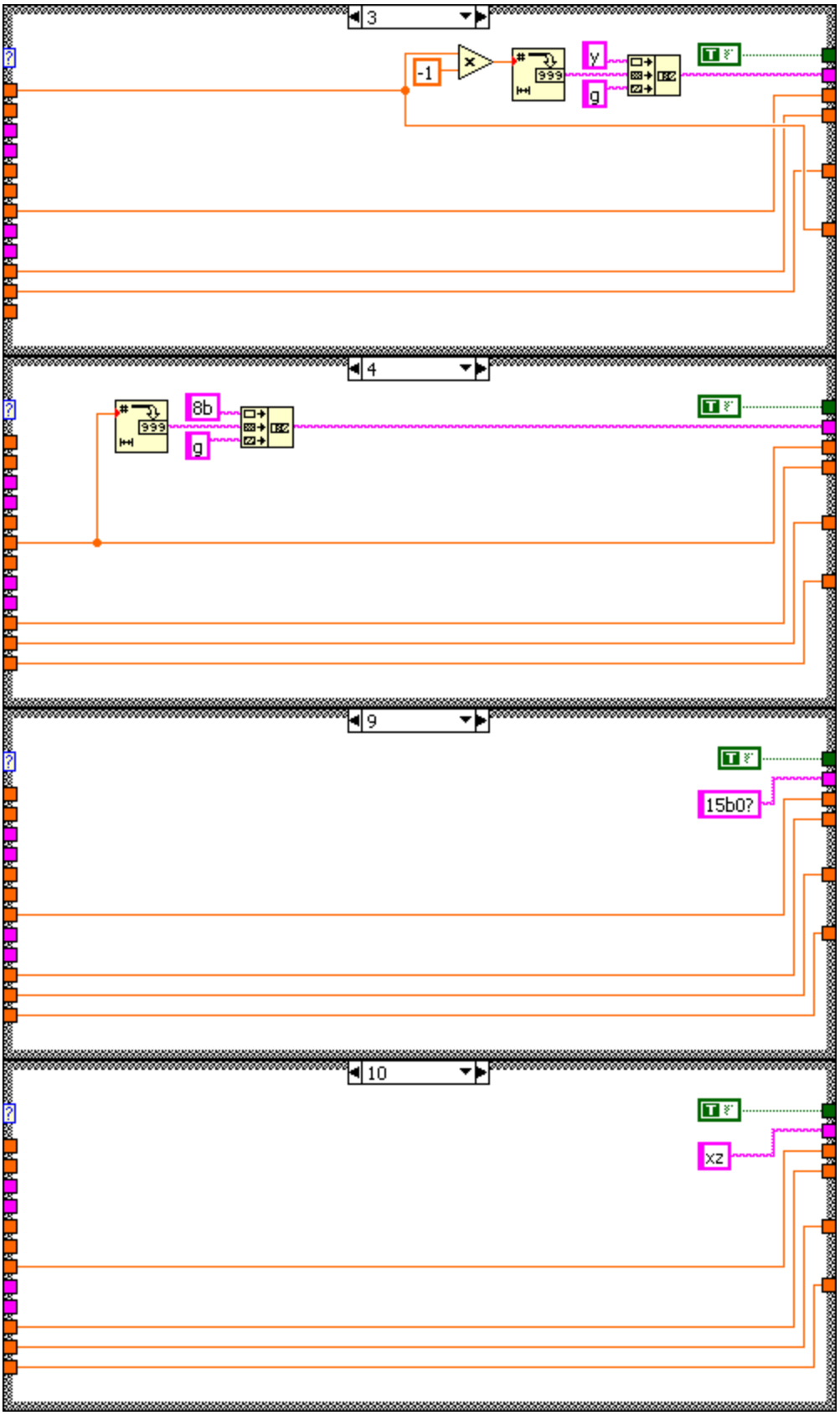


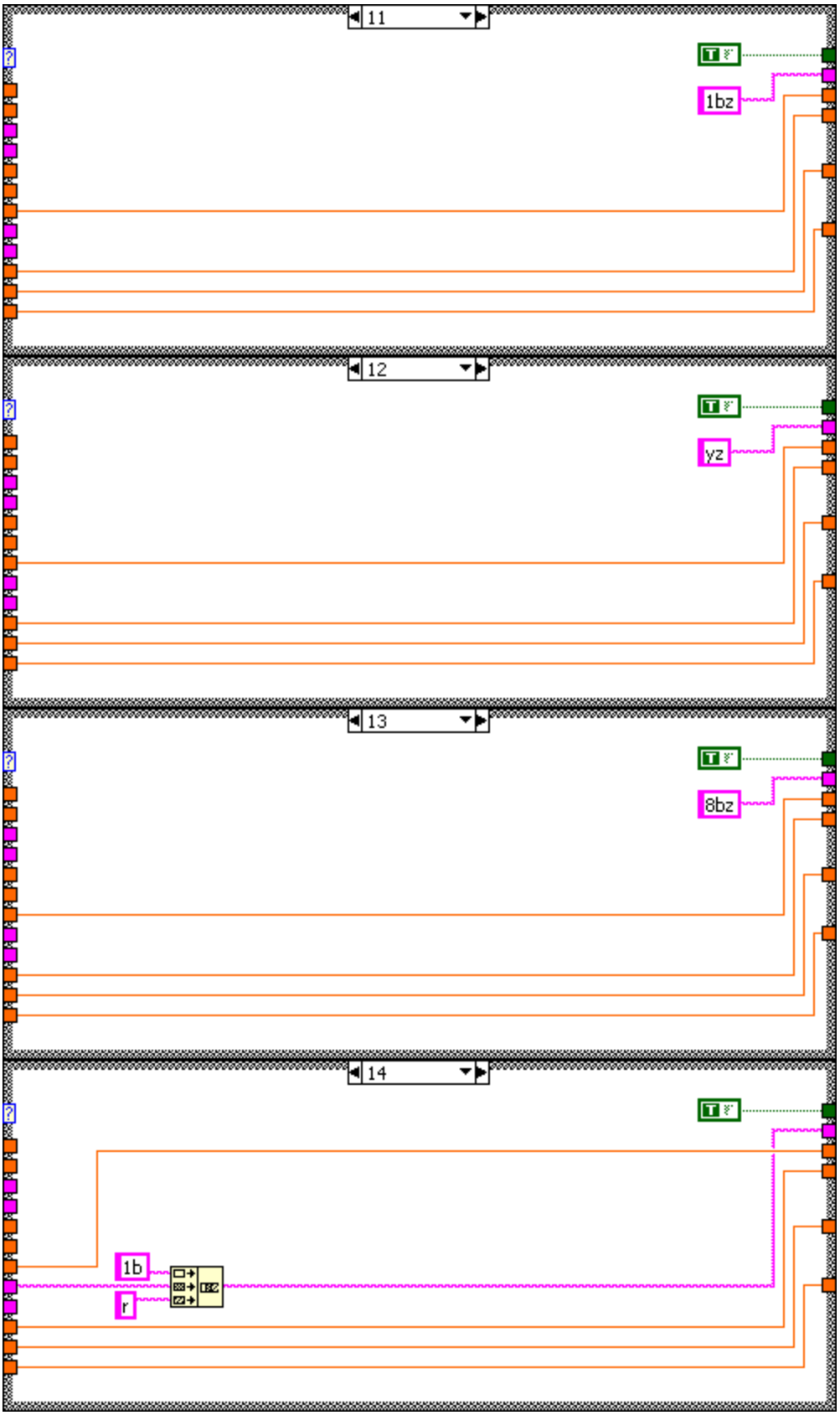

Connecting
Connecting
port
False
True
Initialize Array
Incoming
net error
connection ID out
code
%5.3f
56
Connecting
NumIn
Message
True
Message
NumIn
Value
1000
1000
340
Fuel Desired Position
Fuel Move
Steam Desired Position
Steam Move
Load Desired Position
Load Move
Message
True

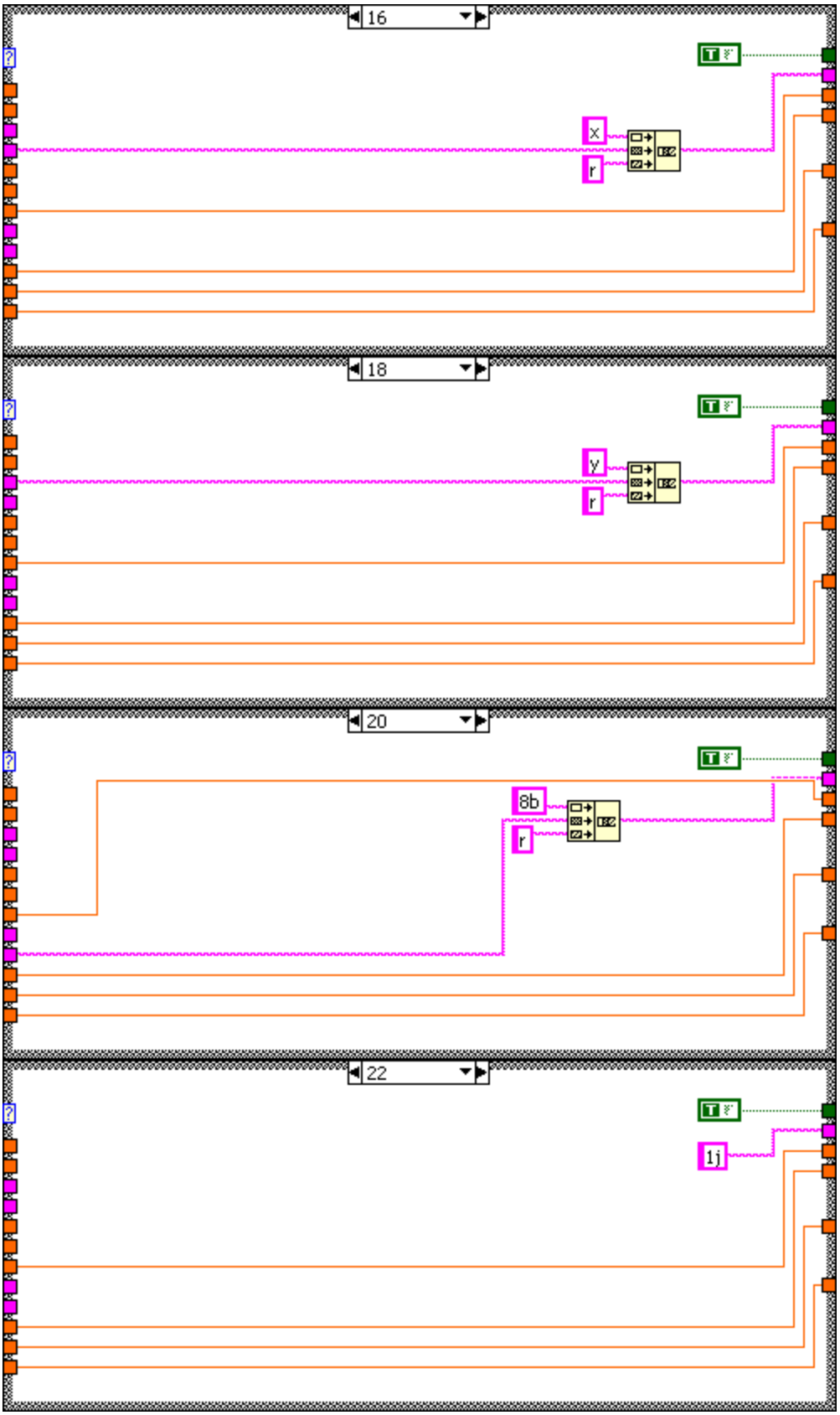


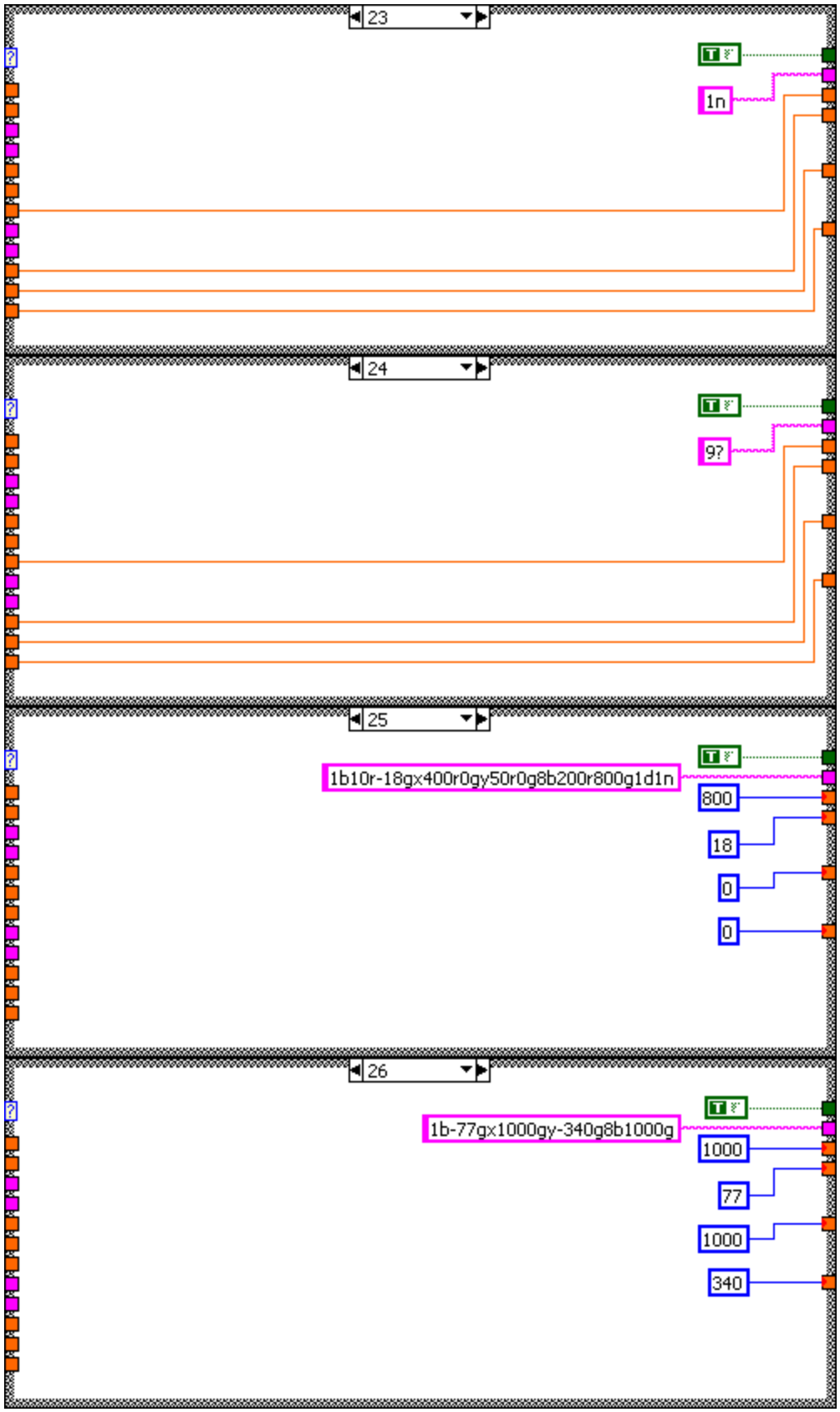

False
True
NumIn
Message
DBL
TF
False
Connecting
port
1.23
UDP
True
connection ID out
0
False

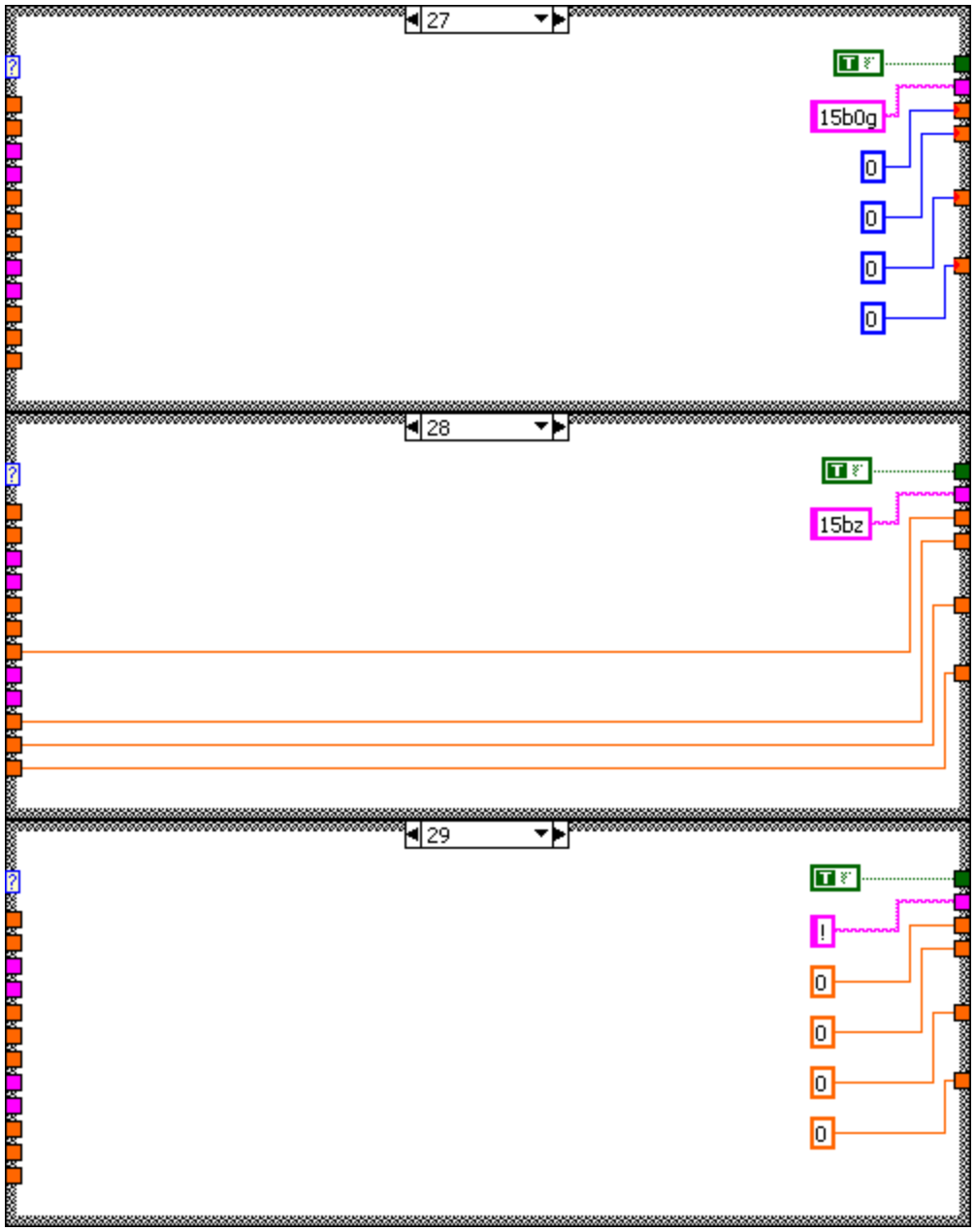

True
1000
1000
340
Fuel Desired Position
Fuel Move
Steam Desired Position
Steam Move
Load Desired Position
Load Move
Message
Value
Val(Sgnl)
0, Default
15b0?
1
-1
999
1b
g
2
X

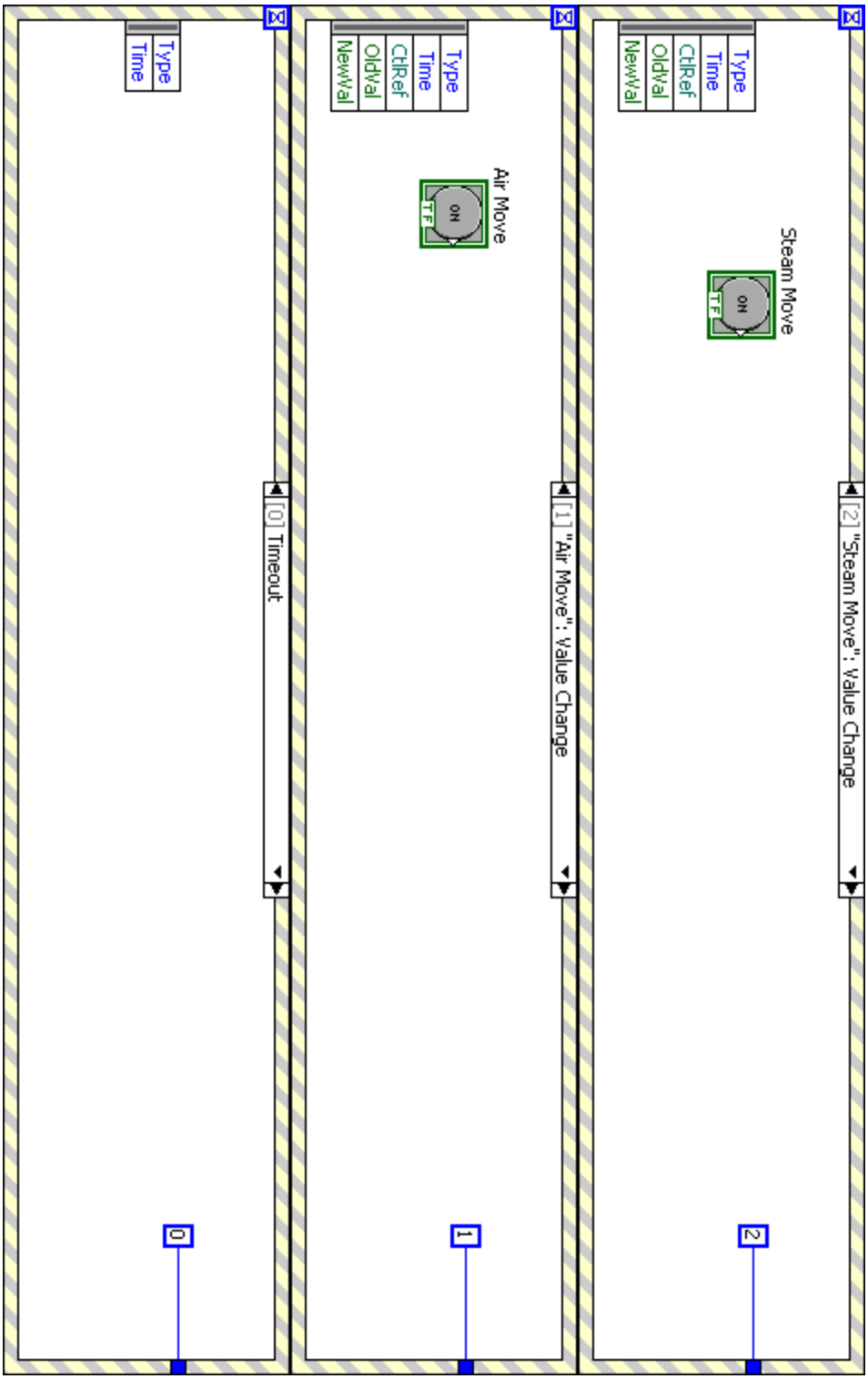

3
-1
999
y
g
4
8b
999
g
9
15b0?
10
xz

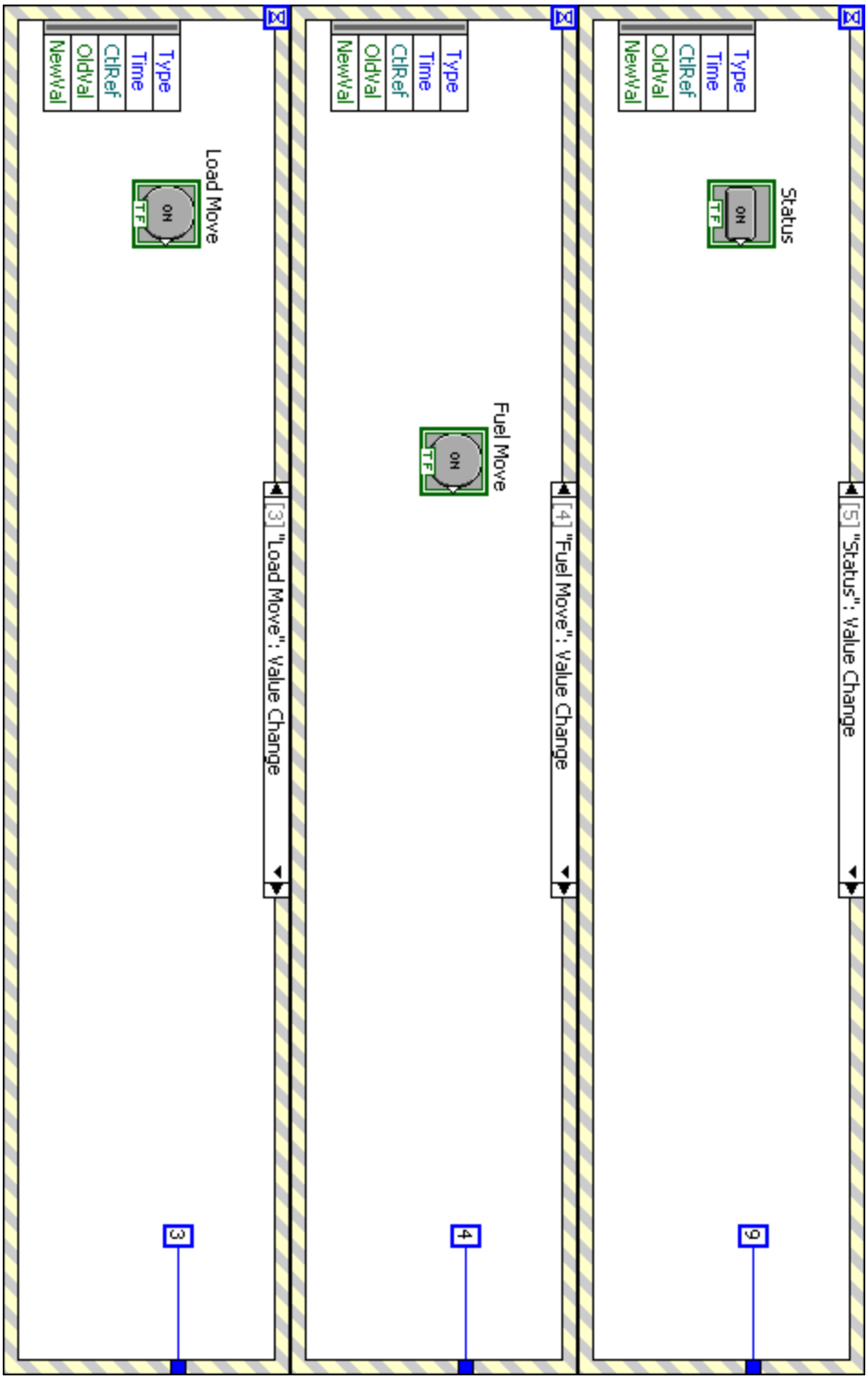

11
1bz
12
yz
13
8bz
14
1b
r

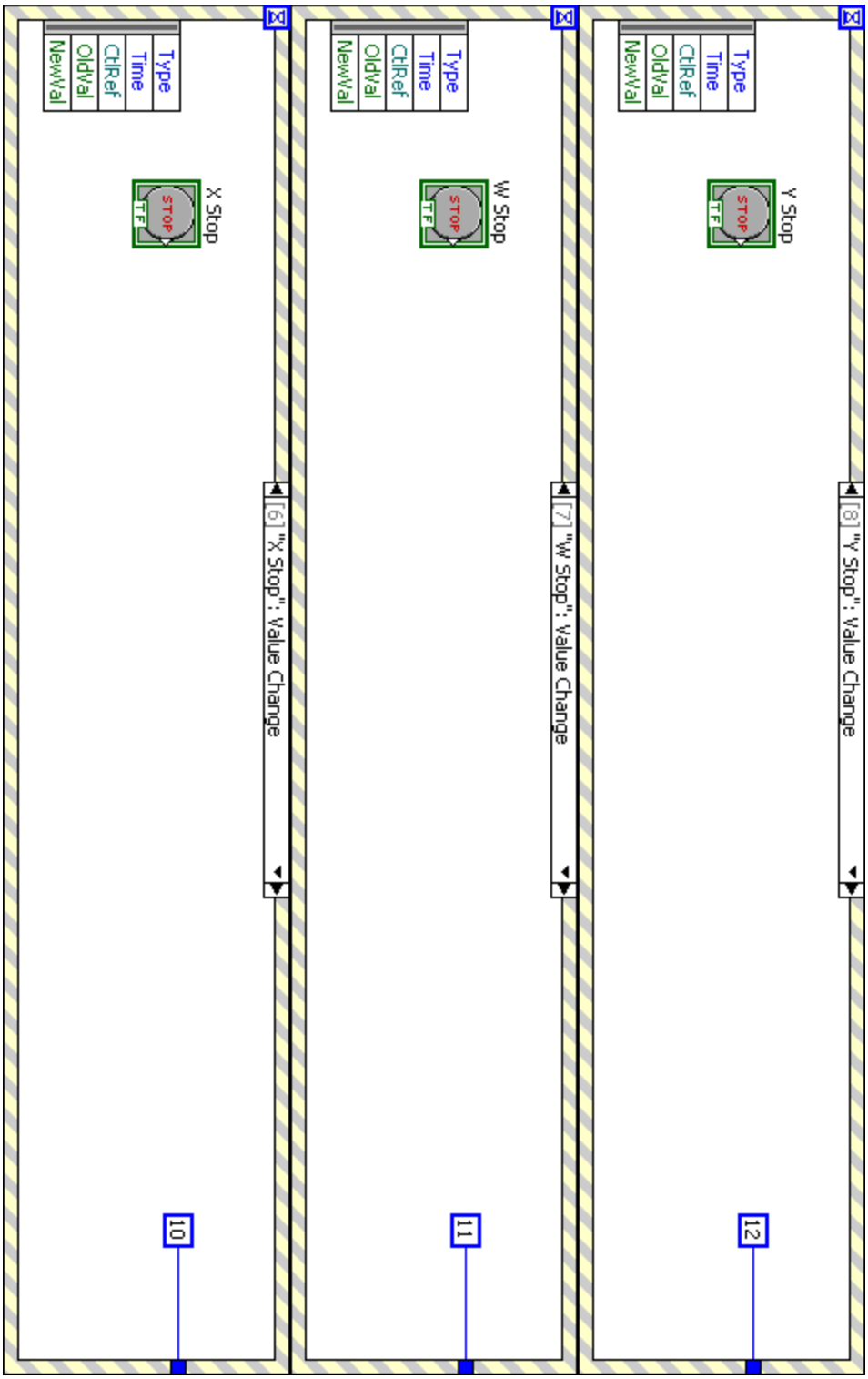

16
x
r
18
y
r
20
8b
r
22
1j

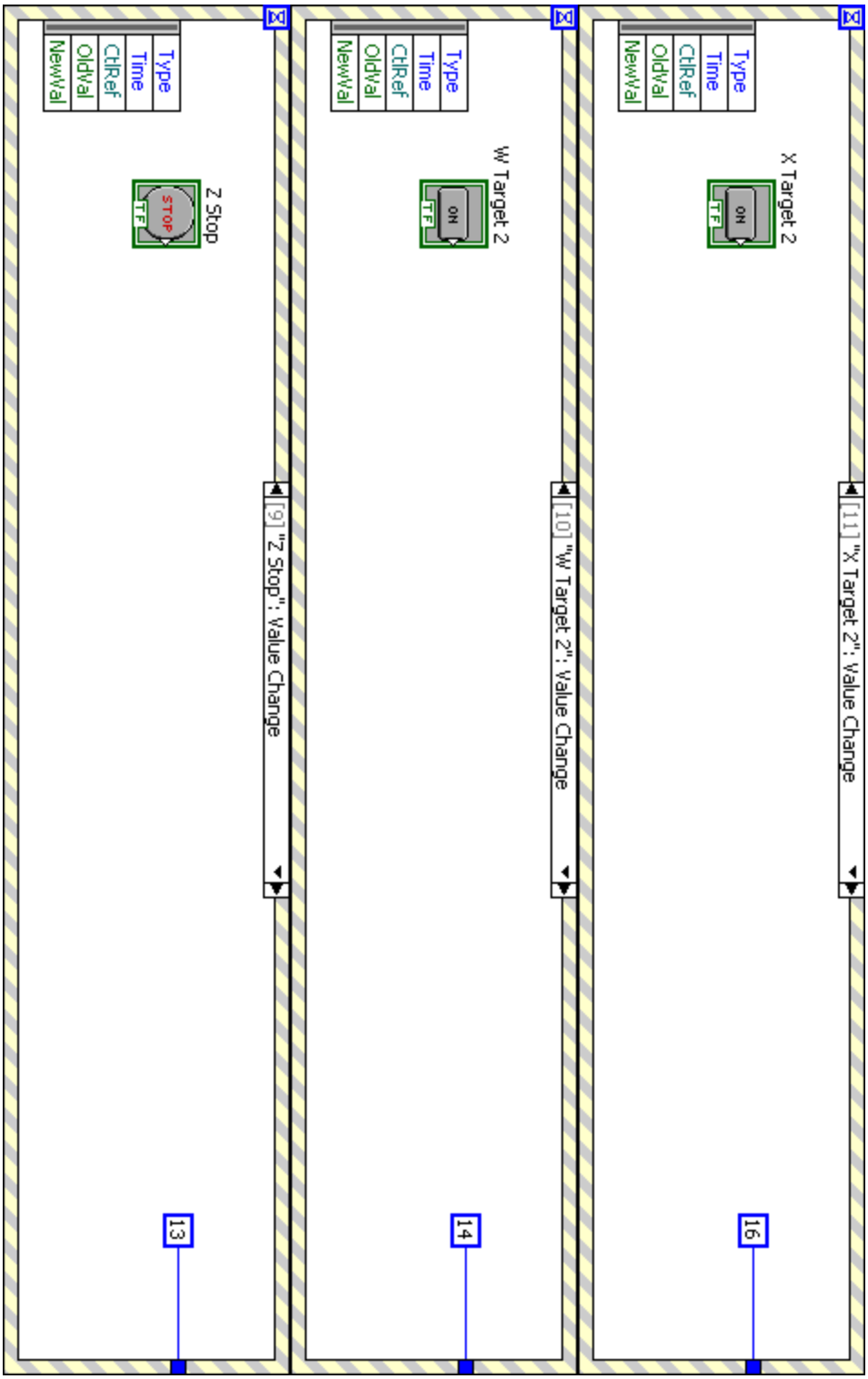

23
1n
24
9?
25
1b10r-18gx400r0gy50r0g8b200r800g1d1n
800
18
0
0
26
1b-77gx1000gy-340g8b1000g
1000
77
1000
340

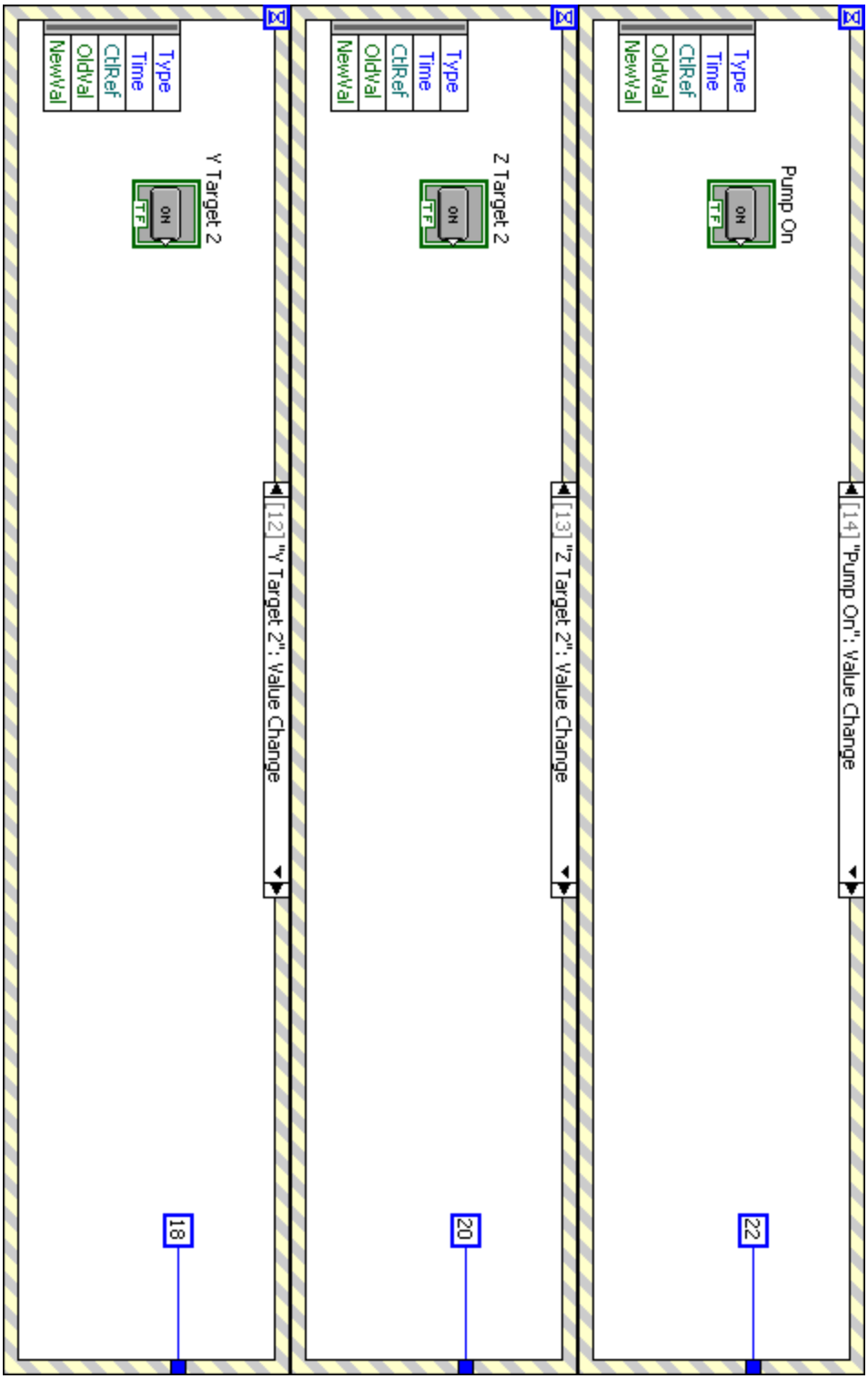

27
15b0g
0
0
0
0
28
15bz
29
!
0
0
0
0

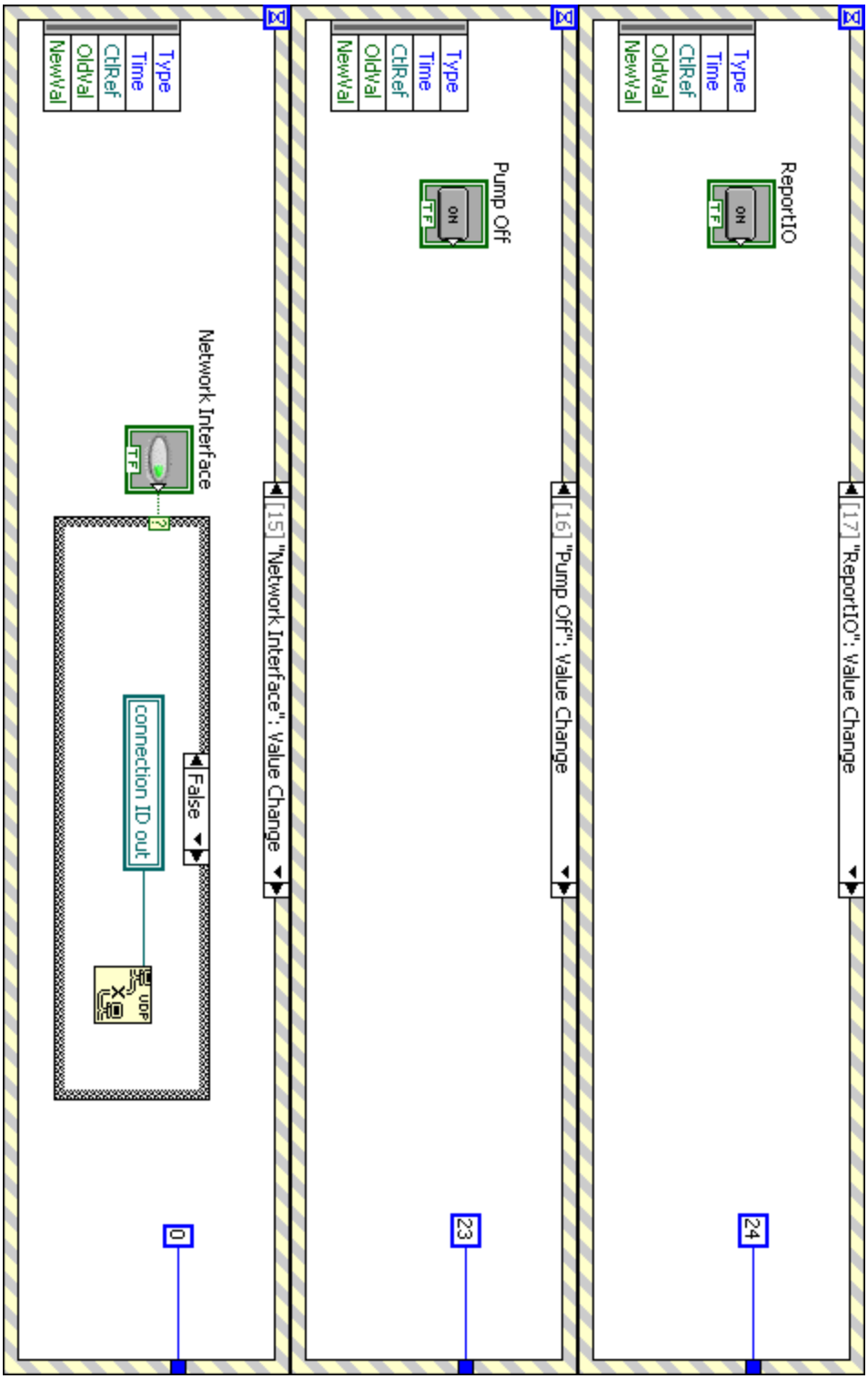

[2] "Steam Move": Value Change
Type
Time
CtlRef
OldVal
NewVal
Steam Move
ON
TF
2
[1] "Air Move": Value Change
Type
Time
CtlRef
OldVal
NewVal
Air Move
ON
TF
1
[0] Timeout
Type
Time
0

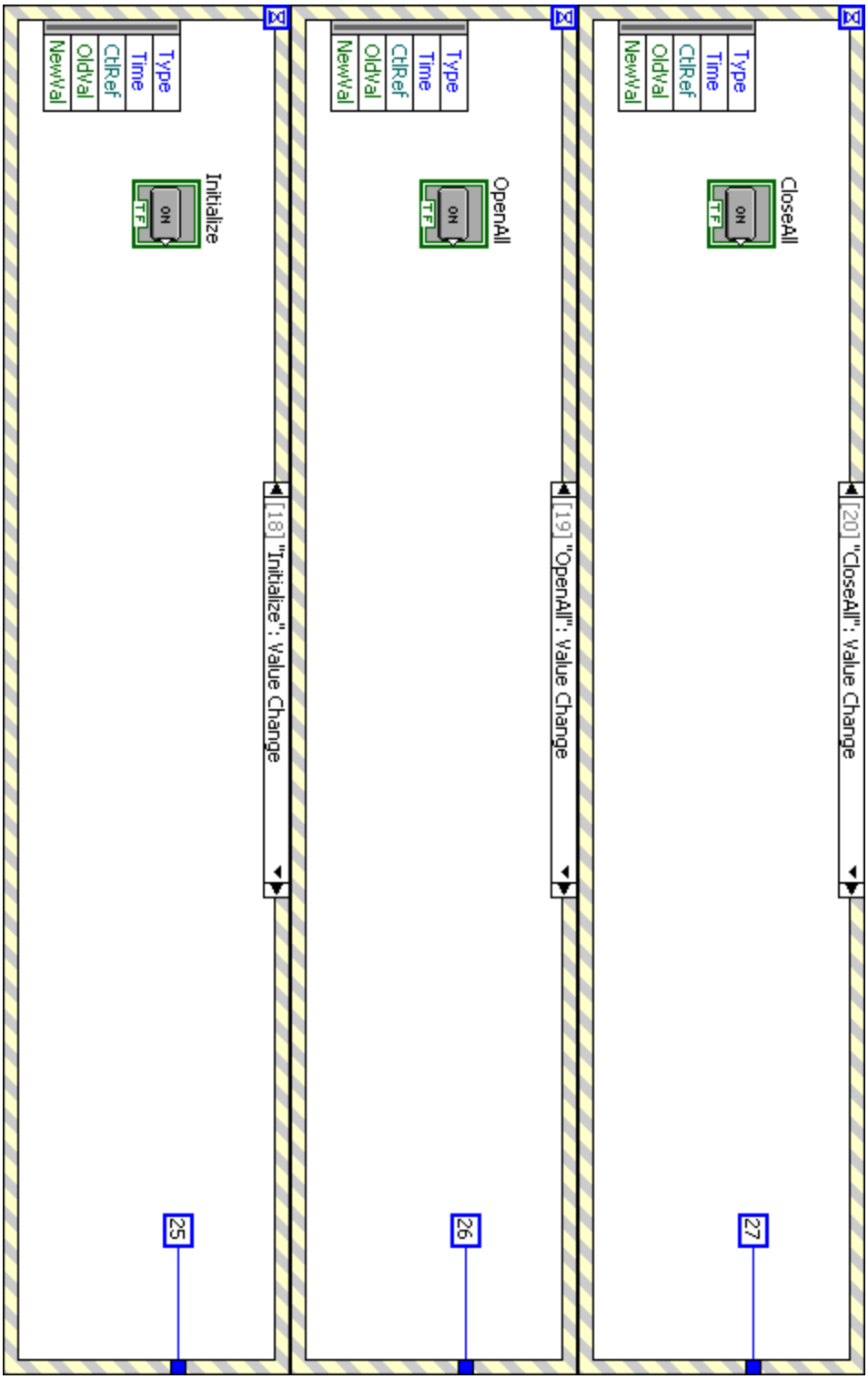

[5] "Status": Value Change
Type
Time
CtlRef
OldVal
NewVal
Status
ON
TF
9
[4] "Fuel Move": Value Change
Type
Time
CtlRef
OldVal
NewVal
Fuel Move
ON
TF
4
[3] "Load Move": Value Change
Type
Time
CtlRef
OldVal
NewVal
Load Move
ON
TF
3

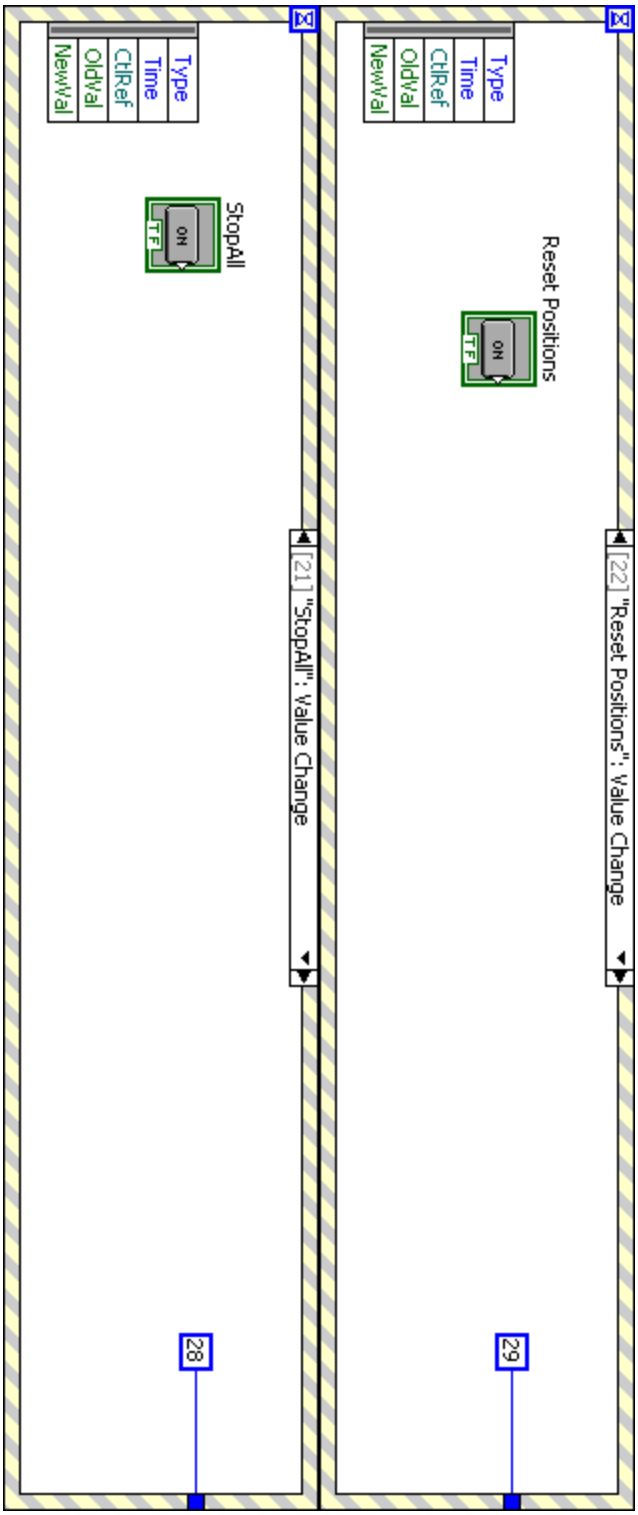

[8] "Y Stop": Value Change
Type
Time
CtlRef
OldVal
NewVal
Y Stop
STOP
TF
12
[7] "W Stop": Value Change
Type
Time
CtlRef
OldVal
NewVal
W Stop
STOP
TF
11
[6] "X Stop": Value Change
Type
Time
CtlRef
OldVal
NewVal
X Stop
STOP
TF
10

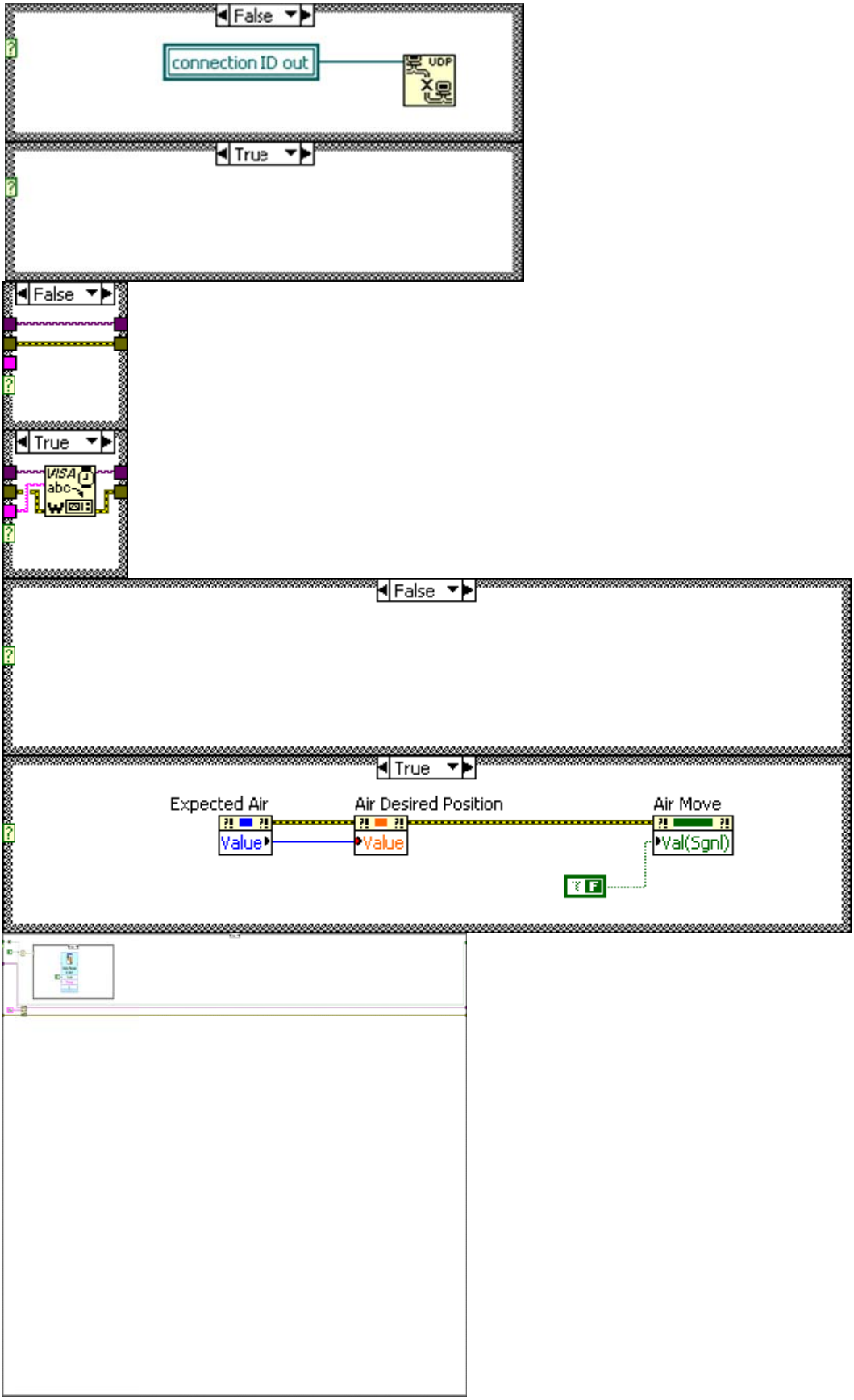

[11] "X Target 2": Value Change
Type
Time
CtlRef
OldVal
NewVal
X Target 2
ON
TF
16
[10] "W Target 2": Value Change
Type
Time
CtlRef
OldVal
NewVal
W Target 2
ON
TF
14
[9] "Z Stop": Value Change
Type
Time
CtlRef
OldVal
NewVal
Z Stop
STOP
TF
13

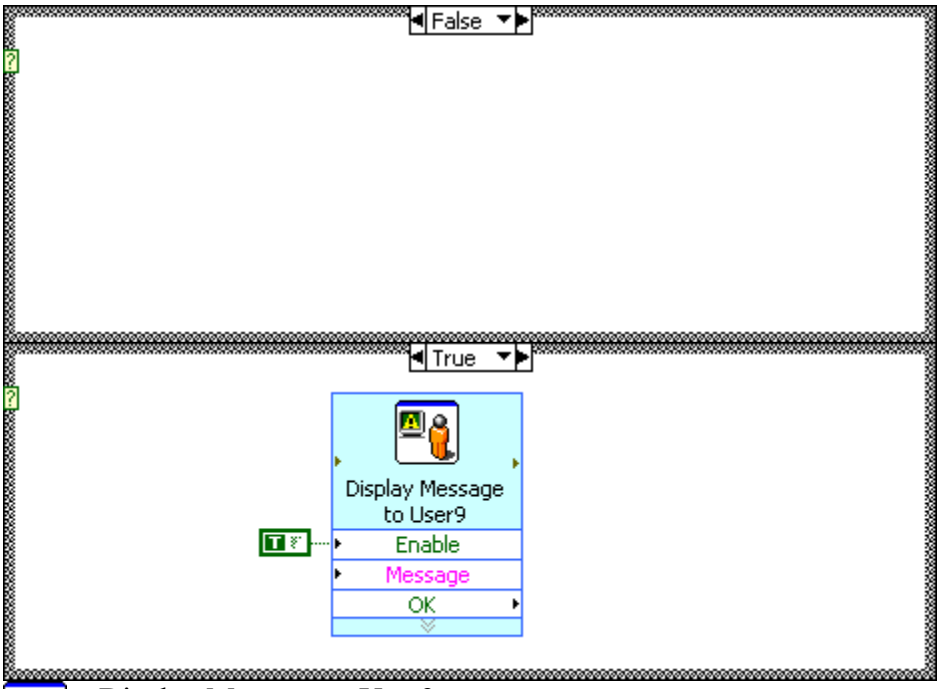

[14] "Pump On": Value Change
Type
Time
CtlRef
OldVal
NewVal
Pump On
ON
TF
22
[13] "Z Target 2": Value Change
Type
Time
CtlRef
OldVal
NewVal
Z Target 2
ON
TF
20
[12] "Y Target 2": Value Change
Type
Time
CtlRef
OldVal
NewVal
Y Target 2
ON
TF
18


[17] "ReportIO": Value Change
Type
Time
CtlRef
OldVal
NewVal
ReportIO
ON
TF
24
[16] "Pump Off": Value Change
Type
Time
CtlRef
OldVal
NewVal
Pump Off
ON
TF
23
[15] "Network Interface": Value Change
Type
Time
CtlRef
OldVal
NewVal
Network Interface
TF
?
False
connection ID out
UDP
0


[20] "CloseAll": Value Change
Type
Time
CtlRef
OldVal
NewVal
CloseAll
TF
27
[19] "OpenAll": Value Change
Type
Time
CtlRef
OldVal
NewVal
OpenAll
TF
26
[18] "Initialize": Value Change
Type
Time
CtlRef
OldVal
NewVal
Initialize
TF
25


[22] "Reset Positions": Value Change
Type
Time
CtlRef
OldVal
NewVal
Reset Positions
ON
TF
29
[21] "StopAll": Value Change
Type
Time
CtlRef
OldVal
NewVal
StopAll
ON
TF
28


False
connection ID out
UDP
True
False
True
VISA
abc
False
True
Expected Air
Value
Air Desired Position
Value
Air Move
Val(Sgnl)



Display Message to User9
Display Message to User
Displays a standard dialog box that contains an alert or a message for users.

This Express VI is configured as follows:

Message: WARNING: System in Override mode.

Boolean Changed__ogtk.vi

Data Changed__ogtk.vi

APPENDIX C

MAS MATLAB Code

C.1 MASagent.m

```
% MASAGENT Runs an agent.
%   Execute this script to start a new agent instance.
%
%   Jason Head, Craig Williams, Jason Gomes, John Miller, 6/18/09
function MASagent()

global agent scomm rcomm
global message

% method = 'tcpip';
method = 'udp';

cd ..; cd Utility
MASpath
cd ..; cd Agents

% Set the path
switch labindex
    case 1
        MAStask(method);
    case 2
        socketHandler(2000,method);
    case 3
        pause(2)
        MAStask(method);
    case 4
        pause(2)
        socketHandler(2000,method);
    case 5
        pause(4)
        MAStask(method);
    case 6
        pause(4)
        socketHandler(2000,method);
    case 7
        pause(6)
        MAStask(method);
    case 8
        pause(6)
        socketHandler(2000,method);
end
```

C.2 MAStask.m

```
function MAStask(method)
% MASTASK Runs an agent task.
%
%   Jason Head, Craig Williams, Jason Gomes, John Miller, 6/18/09

HARDCODE_TASKS = 1;

global agent
global message rcomm scomm
fprintf('Waiting for Agent Structure...')
go = 0;
while ~go
    fprintf('.')
    pause(.5)
    %Checking for agent struct from sockethandler with TAG =11;
    if labProbe(labindex+1,11)
        go = 1;
    end
end

% Initialize agent table with this agent's info
% agent.agentTable{1}=struct('ip',agent.ip,'port',
agent.port,'task','freeAgent','rank',agent.task.rank);

% Let messaging thread begin
first=1;

% Main Loop
run = 1;
while run
    %Receiving agent from lab 2 with TAG=11;
    agent  = labReceive(labindex+1,11);
    if first
    fprintf(['\nStarting Task on port. ' num2str(agent.rport) '\n'])
    first=0;
    end
    % Start stopwatch
    tic
    % fprintf('AgentTable:\n ')
    % display(agent.agentTable{1})

    if HARDCODE_TASKS == 0
        switch agent.task.val

            case 'freeAgent'
                fprintf('\nThis agent is now a free agent\n')
                freeAgent(method);

            case 'interfaceAgent'
                fprintf('\nThis agent is the interface agent\n')
                interfaceAgent(method);
```

```
            case 'delegationAgent'
                fprintf('\nThis agent is the delegation agent\n')
                delegationAgent(method);

            case 'monitoringAgent'
                fprintf('\nThis agent is the monitoring agent\n')
                monitoringAgent(method);

            case 'nnAgent'
                fprintf('\nThis agent is the neural network agent\n')
                nnAgent(method);

            case 'databaseAgent'
                fprintf('\nThis agent is the database agent\n')
                databaseAgent(method);

            case 'controlAgent'
                fprintf('\nThis agent is the control agent\n')
                controlAgent(method);

            case 'feedbackAgent'
                fprintf('\nThis agent is the feedback agent\n')
                feedbackAgent(method);

            case 'gainoptimizerAgent'
                fprintf('\nThis agent is the gain optimizer agent\n')
                gainoptimizerAgent(method);

            otherwise
                fprintf('\nThis agent is now a free agent\n')
                freeAgent(method);

        end
    elseif HARDCODE_TASKS == 1
        switch(agent.ip)
            case '129.62.150.155'
                if agent.rport == 10000
                    fprintf('\nThis agent is the GO NN Agent agent\n')
                    neuralnetworkAgentGO(method)
                elseif agent.rport == 10001
                    fprintf('\nThis agent is the RG NN Agent agent\n')
                    neuralnetworkAgentRG(method);
                elseif agent.rport == 10002

                elseif agent.rport == 10003

                end
            case '129.62.150.156'
            case '129.62.151.131'
                if agent.rport == 10000
                    fprintf('\nThis agent is the delegation agent\n')
                    delegationAgent(method)
                elseif agent.rport == 10001
                    fprintf('\nThis agent is the control agent\n')
                    controlAgent(method);
```

```
                elseif agent.rport == 10002
                    fprintf('\nThis agent is the feedback agent\n')
                    agent.task.val = 'feedbackAgent';
                    feedbackAgent(method);
                elseif agent.rport == 10003
                    fprintf('\nThis agent is the gain optimizer
agent\n')
                    gainoptimizerAgent(method);
                end
            case '129.62.150.158'
        end
    end

    if HARDCODE_TASKS == 0
        % Get elapsed time (how long task took)
        agent.performance = toc;
        % --- Send performance data to Delegation Agent ---
        % Get (primary) delegation agent's IP
        DA = [];
        for n = 1: size(agent.agentTable,2)
            if strcmp(agent.agentTable{n}.task , 'delegationAgent') &&
agent.agentTable{n}.rank == 1
                DA = agent.agentTable{n};
            end
        end

        if ~isempty(DA)
            % Build receiver structure
            receiver =
struct('ip',DA.ip,'rport',DA.rport,'task',struct('val','delegationAgent
','rank', DA.task.rank));

            % Bulid message content
            content = {'data','performance','actual',
agent.performance};

            % Build message
            %         msg =
buildMessage('inform',agent,receiver,content);
            msg = encodeMASmsg('inform',agent,receiver,content);

            % Send message to delegation agent

sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
        end
        %(Place original code here)

        %REVERTING BACK TO FREE AGENT PART OF CODE
        if ~strcmp(agent.task.val, 'freeAgent')
            if agent.taskCounter > agent.taskMax

                %             receiver =
struct('ip',agent.ip,'port',agent.port,'task',agent.task.val);
```

```
                receiver =
struct('ip',agent.ip,'rport',agent.rport,'task',struct('val',agent.task
.val,'rank',agent.task.rank));
                %Bulid message content
                content={'become', 'freeAgent',1};
                %             msg =
buildMessage('request',agent,receiver,content);
                msg = encodeMASmsg('request',agent,receiver,content);

sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
                %         pause(0.2)
                %         pause(0.1)
            end
        end
    end
    %?Do I need to do a labSend here?/ TAG of 10 for done tag
    labSend('done',labindex+1,10);
%     fprintf('\n-----------------------\n')
    %     labBarrier;
    synched = 0;
    while ~synched
        if labProbe(labindex+1,11)
            synched = 1;
        end
    end

end
```

C.3 decodeMASmsg.m

```
function msg = decodeMASmsg(msg)
%MAS Decoder Function.  Works now with any message and special case
where
%we send the agenttable

% msg =
['inform$$129.62.150.157$4000$delegationAgent$1$$129.62.150.158$4001$fe
edbackAgent$1$$FF_data$double$100$string$Jason$double$123.256$$'];

field = 1; ifield = 1; idx2 = 0; write = 0;
performative = []; content = []; replyby = [];
sip = []; sport = []; stask = []; srank = [];
rip = []; rport = []; rtask = []; rrank = [];
msgtype = []; vartype = []; value = [];

%testing flags for agenttable decoding
agenttable = 0;
afield = 0;
% content2 = cell(10,4);
line = 1;
ipwrite = 0;
portwrite = 0;
taskwrite = 0;
rankwrite = 0;

for idx = 1:length(msg)-1
    if strcmp(msg(idx:idx+1),'$$')
        field = field + 1;
        ifield = 0;
    elseif strcmp(msg(idx),'$') && agenttable == 0
        ifield = ifield + 1;
    elseif strcmp(msg(idx),'$') && agenttable
        ifield = ifield + 1;
        afield = afield + 1;
        if afield > 4
            afield = 1;
            line = line + 1;
        end
    elseif ~strcmp(msg(idx:idx+1),'$$')
        switch(field)
            case 1
                performative = [performative msg(idx)];
            case 2
                switch(ifield)
                    case 1
                        sip = [sip msg(idx)];
                    case 2
                        sport = [sport msg(idx)];
                    case 3
                        stask = [stask msg(idx)];
                    case 4
                        srank = [srank msg(idx)];
                end
```

```
            case 3
                switch(ifield)
                    case 1
                        rip = [rip msg(idx)];
                    case 2
                        rport = [rport msg(idx)];
                    case 3
                        rtask = [rtask msg(idx)];
                    case 4
                        rrank = [rrank msg(idx)];
                end
            case 4
                if ifield == 1
                    msgtype = [msgtype msg(idx)];
                    write = 1;
                elseif mod(ifield,2) == 0 && agenttable == 0
                    vartype = [vartype msg(idx)];
                    if strcmp(vartype,'agentTable')
                        agenttable = 1 ;
                    end
                elseif mod(ifield-1,2) == 0 && agenttable == 0
                    value = [value msg(idx)];
                    write = 1;
                elseif afield == 1
                    value = [value msg(idx)];
                    ipwrite = 1;
                elseif afield == 2
                    value = [value msg(idx)];
                    portwrite = 1;
                elseif afield == 3
                    value = [value msg(idx)];
                    taskwrite = 1;
                elseif afield == 4
                    value = [value msg(idx)];
                    rankwrite = 1;
                end
            case 5
                replyby = [replyby msg(idx)];
        end

    end
    if write == 1 && mod(ifield,2) == 0 && agenttable == 0% this
section must come after the the switch to properly get values
        idx2 = idx2 + 1; write = 0;
        if idx2 == 1
            content{1,idx2} = msgtype;
        else
            if strcmp(vartype,'string')
                content = [content value];
                value = []; vartype = [];
            elseif strcmp(vartype,'double')
                content = [content str2double(value)];
                value = []; vartype = [];
            elseif strcmp(vartype,'num')
                content = [content str2num(value)];
                value = []; vartype = [];
            end
```

```
        end
        contentnum = numel(content);
    end
    if ipwrite && mod(afield,2) == 0
       ipwrite = 0;
%         content2{line,1} = [content2{line,1} value];
        content{contentnum+1}{line}.ip = value;
        value = [];

    elseif portwrite && mod(afield-1,2) == 0
        portwrite = 0;
%         content2{line,2} = [content2{line,2} str2double(value)];
        content{contentnum+1}{line}.rport = str2double(value);
        value = [];
    elseif taskwrite && mod(afield-1,3) == 0
        taskwrite = 0;
%         content2{line,3} = [content2{line,3} value];
        content{contentnum+1}{line}.task.val = value;
        value = [];
    elseif rankwrite && mod(afield,4) == 0
        rankwrite = 0;
%         content2{line,4} = [content2{line,4} str2double(value)];
        content{contentnum+1}{line}.task.rank = str2double(value);
        content{contentnum+1}{line}.task.time = 0;
        value = [];
    end

end

% for k =1:line
%     content{contentnum+1}{k}.ip = content2{k,1};
%     content{contentnum+1}{k}.port = content2{k,2};
%     content{contentnum+1}{k}.task.val = content2{k,3};
%     content{contentnum+1}{k}.task.rank = content2{k,4};
%     content{contentnum+1}{k}.task.time = 0;
% end

if agenttable
    content{end} =content{end}';
    content = {content};
else
    content = {content};
end

replyby = str2double(replyby);
msg = struct('performative',performative,...

'sender',struct('ip',sip,'rport',str2double(sport),'task',struct('val',
stask,'rank',str2double(srank))),...

'receiver',struct('ip',rip,'rport',str2double(rport),'task',struct('val
',rtask,'rank',str2double(rrank))),...
    'content',content,...
    'replyby',replyby);
```

C.4 encodeMASmsg.m

```
function msg =
encodeMASmsg(performative,sender,receiver,content,varargin)

% receiver = struct('ip',fbip,'port',fbport,'task','feedbackAgent');
% content = {'FF_data',yd,ustar,running,time,samplerate,Euldp,method};
% msg = buildMessage('request',agent,receiver,content);

msg = [performative ...
    '$$' sender.ip '$' num2str(sender.rport) '$' sender.task.val '$'
num2str(sender.task.rank) ...
    '$$' receiver.ip '$' num2str(receiver.rport) '$' receiver.task.val
... % '$' num2str(receiver.task.rank) ...
    '$$'];

for n=1:numel(content)

    if iscell(content{n})
        msg =[msg 'agentTable$'];
        for k=1:numel(content{n})
            msg = [msg content{n}{k}.ip '$'
num2str(content{n}{k}.rport) '$' ...
                    content{n}{k}.task.val '$'
num2str(content{n}{k}.task.rank) '$'];
%num2str(content{n}{k}.task.time) '$'];
        end
    else
        if n == 1
            msg = [msg content{n} '$'];
        else
            if ischar(content{n})
                msg = [msg 'string$' content{n} '$'];
            elseif numel(content{n}) == 1
                msg = [msg 'double$' num2str(content{n}) '$'];
            elseif numel(content{n}) > 1
                msg = [msg 'num$' num2str(content{n}) '$'];
            end
        end
    end
end

if numel(content) > 1
    msg = [msg '$'];
end

if nargin == 5
    replyby = varargin{1};
    msg = [msg num2str(replyby) '$$'];
end
```

C.5 buildMessage.m

```
function msg =
buildMessage(performative,agent,receiver,content,varargin)
%BUILDMESSAGE Forms an MAS message.
%   MSG=BUILDMESSAGE(PERFORMATIVE,AGENT,RECEIVER,CONTENT) creates an
MAS
%   message with a specific structure. See below for arguments.
%
%   MSG=BUILDMESSAGE(..., REPLY_BY) includes a 'reply_by' field in the
%   message.
%
%   Inputs:
%       PERFORMATIVE - a string
%       AGENT - an 'agent' struct (see MASagent)
%       RECEIVER - a struct with fields: IP, PORT, TASK
%           IP & TASK are strings, PORT is a number
%       CONTENT - a NxM cell array containing N 'sentences' with M
'words',
%           with each word being a keyword (i.e. string) that is part
of
%           the MAS communications language (see messageShell)
%       REPLY_BY - a time stamp (see clock)
%
%   For example:
%       performative = 'inform';
%       % 'agent' is a global struct variable
%       receiver = struct('ip','129.62.150.129','port',4000, ...
%           'task','freeAgent')
%       content = {'data','actual','Euld',[45 46 47 48 49 50]}
%       msg = buildMessage(performative,agent,receiver,content);
%
%   See also MASAGENT, MESSAGESHELL, CLOCK, SENDMASMSG, READMASMSG
%
%   Jason Head, Craig Williams, Jason Gomes, John Miller, 6/29/09

% Get optional input
if nargin>4, reply_by = varargin{1}; end

% Get sender's ip, port, and name from agent struct
sender =
struct('ip',agent.ip,'port',agent.port,'task',agent.task.val,'rank',age
nt.task.rank);

% 'receiver' is a struct that has already been prepared
%
receiver=struct('ip',receiverip,'port',receiverport,'task',receivertask
);

% Determine whether to include 'reply-by' field or not
if exist('reply_by','var')
    msg =
struct('performative',performative,'sender',sender,'receiver',receiver,
...
        'content',{content},'reply_by',reply_by);
else
```

```
    msg =
struct('performative',performative,'sender',sender,'receiver',receiver,
...
        'content',{content});
end
end
```

C.6 createCommChannels.m

```
function varargout =
createCommChannels(method,rport,sport,timeout,varargin)

usercomm = 1; rcomm = []; buffer = 1024;
for n=1:2:numel(varargin)
    switch(varargin{n})
        case 'rcomm'
            usercomm = varargin{2};
        case 'buffer'
            buffer = varargin{n+1};
    end
end

% Create communication object
switch(method)
    case 'tcpip'
        comm = createListenSocket(rport,timeout);
    case 'udp'
        if usercomm == 1
            rcomm = udp('','LocalPort',rport);
            rcomm.inputbuffersize = buffer;
%             rcomm.DatagramTerminateMode = 'off';
%             rcomm.Terminator = '';
            fopen(rcomm);
        end
        scomm = udp('','LocalPort',sport);
%         scomm.DatagramTerminateMode = 'off';
%         scomm.Terminator = '';
end

if nargout == 1
    if strcmp(method,'tcpip')
        varargout = {comm};
    else
        varargout = {scomm};
    end
elseif nargout == 2
    varargout = {rcomm,scomm};
end
```

C.7 sockethandler.m

```
function socketHandler(timeout,method)

% Import needed Java classes
import java.net.*
import java.io.*
global agent rcomm scomm
global lisSocket
fprintf('Initializing Agent Structure...\n')
% Initialize Agent variables
agent = struct( ...
    'task',         struct('val','freeAgent','time',now,'rank',1), ...
    'ip',           getIP(), ...
    'rport',        getPort('range',[10000 10100],'method',method), ...
    'sport',        [], ...
    'status',       struct('val','none','time',now), ...
    'performance',  0, ...
    'agentTable',   [], ...
    'taskCounter',  0, ...
    'taskMax',      3, ...
    'startup',      0, ...
    'daAlive',      0, ...
    'memory',       []);
agent.sport = agent.rport + 100;
load Constants.mat
agent.agentTable{1}=struct('ip', agent.ip, 'rport', agent.rport,
'sport', agent.sport,'task', struct('val', agent.task.val, 'rank',
agent.task.rank, 'time', 0));
%%% HARD CODING AGENTS
agent.agentTable = {struct('ip', '129.62.151.131', 'rport', 10000,
'sport', 10100, 'task', struct('val', 'delegationAgent'      , 'rank',
1, 'time', 0));...
    struct('ip', '129.62.151.131', 'rport', 10001, 'sport', 10101,
'task', struct('val', 'controlAgent'         , 'rank', 1, 'time',
0));...
    struct('ip', '129.62.151.131', 'rport', 10002, 'sport', 10102,
'task', struct('val', 'feedbackAgent'        , 'rank', 1, 'time',
0));...
    struct('ip', '129.62.150.157', 'rport', 10003, 'sport', 10103,
'task', struct('val', 'gainoptimizerAgent'   , 'rank', 1, 'time',
0));...
    struct('ip', '129.62.150.155', 'rport', 10000, 'sport', 10100,
'task', struct('val', 'neuralnetworkAgentGO'   , 'rank', 1, 'time',
0));...
    struct('ip', '129.62.150.155', 'rport', 10001, 'sport', 10101,
'task', struct('val', 'neuralnetworkAgentRG'   , 'rank', 1, 'time',
0))};
%%%

agent.memory=struct('Euld',[],...
    'K', [], ...
    'running',1, ...
    'time',[], ...
    'samplerate', 1/FFdt, ...
    'beta', RGprefs, ...
```

```
    'iterCnt', 1, ...
    'method', 'udp', ...
    'twin', 50, ...
    'dt',FFdt, ...
    'vectimer',[], ...
    'error',GOerrorThresh, ...
    'slowdown',1, ...
    'tmp',[], ...
    'net',[], ...
    'ps1',[], ...
    'ps2',[], ...
    'particles', HPSOpart, ...
    'iterations', HPSOiter, ...
    'meth', HPSOmeth, ...
    'PwrPltIP', PwrPltIP, ...
    'PwrPltPort', PwrPltPort,...
    'oldEuld',[]);
agent.memory.K(:,:,1) = FBgains(:,:,1);
agent.memory.K(:,:,2) = FBgains(:,:,2);
agent.memory.K(:,:,3) = FBgains(:,:,3);
%load rnetg net ps1 ps2
% agent.memory.net = net;
% agent.memory.ps1 = ps1;
% agent.memory.ps2 = ps2;

% Get optional input arguments
if nargin < 1, timeout = 5000; end  % Set default timeout (in ms)
% Creates objects for sending messages in the task thread and sending
and
% receiving messages in the message thread
if strcmp(method,'tcpip')
    rcomm = createCommChannels(method,agent.rport,agent.sport,timeout);
elseif strcmp(method,'udp')
    [rcomm,scomm] =
createCommChannels(method,agent.rport,agent.sport,timeout);
end

fprintf('AgentTable creation\n')

%Sending agent struct to lab 1 with a TAG= 11;
labSend(agent,labindex-1,11);

fprintf('Agent struct sent to lab 1. Starting Listener... \n')
% Init variables
looptime = tic;
while true
    while ~labProbe(labindex-1,10)
        gotOne=0;

        msg = receiveMASmsg(rcomm,method,1);
        if ~isempty(msg)
            msg = decodeMASmsg(msg); % toc(looptime)
            gotOne = 1; looptime = tic;
        end
        %           try
        %               msg = readLSocketMAS(lisSocket);toc(looptime)
```

```
        %                msg = decodeMASmsg(msg); looptime = tic;
        %                if ((strcmp(msg.sender.ip, agent.ip)) &&
(msg.sender.port == agent.port))
        %                    fprintf('Message received from task
thread. \n')
        %                else
        %                    fprintf('%s Message received from %s at
%s:%d \n',
msg.performative,msg.sender.task,msg.sender.ip,msg.sender.port)
        %                end
        %
        %                gotOne=1;
        %            catch me
        %                if strfind(me.message, 'Accept timed out')
        %                    fprintf('.')
        %                elseif strfind(me.message, 'Output argument
"msg"')
        %                    fprintf('!')
        %                else
        %                    display(me.message)
        %                end
        %            end

        try
            if gotOne
                messageShell(msg,scomm,method);
            end
        catch me
            fprintf('\nMessageShell Error:\n')
            rethrow(me)
        end
    end

    %This catches the labSend at the end of MAStask-TAG of 10
    labReceive(labindex-1,10);
    %Have we heard from the DA in this iteration or am I the delegation
    %agent?
    if agent.daAlive == 0 && ~strcmp(agent.task.val,'delegationAgent')
        agent.taskCounter=agent.taskCounter+1;
    else
        agent.taskCounter = 0;
    end
    agent.daAlive = 0;

    %%%%%%This section of code removes agents that are
unresponsive%%%%%%%%%%%%
    %This only happens when the listening thread syncs with the task
thread
    %         remove= [];
    %         for n=2:numel(agent.agentTable)
    %             % Remove agents after 30 sec of no comm.
    %             if (now-
agent.agentTable{n}.task.time)>3.472222015261650e-4 &&
~strcmp(agent.agentTable{n}.task.val,'feedbackAgent')
    %                 %send message to respond
```

```
    %                   msg =
encodeMASmsg('request',agent,agent.agentTable{n}, {'data' 'task'});
    %
sendMASmsg(msg,scomm,agent.agentTable{n}.ip,agent.agentTable{n}.rport,a
gent.sport,1,method);
    %                   %remove from table
    %                   remove = [n remove];
    %               end
    %           end
    %
    %           agentTable2 = {};
    %           for n=1:numel(agent.agentTable)
    %
    %               if ~any(n == remove)
    %                   agentTable2 = [agentTable2 agent.agentTable{n}];
    %               end
    %
    %           end
    %           agent.agentTable = agentTable2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
    agent.memory.iterCnt= agent.memory.iterCnt+1;

    %Agent struct with TAG=11
    labSend(agent,labindex-1,11)
    agent.startup = 0;
    fprintf('Resyncing with task thread\n')
    fprintf('\n------------------------\n')
    %     labBarrier;
    %     synched = 0;
    %     while ~synched
    %         if labProbe(labindex-1,10)
    %             synched = 1;
    %         end
    %     end

end
end
```

C.8 messageShell.m

```
function messageShell(msg,scomm,method)
%This has been converted to a script file - Jun 19, 2010
global agent

%  Message Thread
%  --------------
%  switch new message
%    case request : info
%      send inform - info
%    case request : task
%      agree/refuse
%    case agree : ack
%      do nothing
%      (?) (set flag)
%    case refuse : request
%      (?) (set flag)
%    case inform : agent table
%      store agent table

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The messaging thread needs to handle replies from the delegation
% agent and reset the task iteration counter:
%   if DA responds
%       agent.taskcounter = 0;
%   end
%
% This thread handles incrementing the counter and checking it to
% determine if the limit is reached.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if strcmp(msg.sender.task.val,'delegationAgent')
    agent.daAlive = 1;
end
% Check Sender for existance in agentTable, and add it if it doesn't
exist
% yet
agentExists = 0;

for n = 1:numel(agent.agentTable)
    if strcmp(agent.agentTable{n}.ip,msg.sender.ip)
        if agent.agentTable{n}.rport == msg.sender.rport
%             agent.agentTable{n}.task.sport = msg.sender.task.sport;
            agent.agentTable{n}.task.val = msg.sender.task.val;
            agent.agentTable{n}.task.rank = msg.sender.task.rank;
            agent.agentTable{n}.task.time = now;
            agentExists = 1;
            fprintf('\nAgentTable Correction\n')
        end
    end
end

if agentExists == 0
    agent.agentTable{n+1} = struct('ip',msg.sender.ip, ...
        'rport',msg.sender.rport, ...
        ...%'sport',msg.sender.sport, ...
```

```
        'task',struct('val',msg.sender.task.val, ...
                      'rank',msg.sender.task.rank,...
                      'time',now));
    fprintf('\nAgentTable Addition\n')
end


% display(agent.agentTable{1})
%update agentTable here
% remove= [];
% for n=2:numel(agent.agentTable)
%     if (now-agent.agentTable{n}.time)>5.8032e-5
%         %send message to respond
%         msg = buildMessage('request',agent,agent.agentTable{n},
{'data' 'task'});
%         SendMASmsg(msg);
%         %remove from table
%         remove = [n remove];
%     end
% end
%
% agentTable2 = {};
% for n=1:numel(agent.agentTable)
%
%     if ~any(n == remove)
%         agentTable2 = [agentTable2 agent.agentTable{n}];
%     end
%
% end
% agent.agentTable = agentTable2;


% Set the sender of the incoming message to be the recipient of the
% response message and set the number of retries
receiver=struct('ip',   msg.sender.ip, ...
    'rport', msg.sender.rport, ...
...%     'sport', msg.sender.sport, ...
    'task', struct('val', msg.sender.task.val, ...
                   'rank',  msg.sender.task.rank));
retries=2;

% Processes the message
switch msg.performative

    case 'request'
        reply = perfRequest(msg);
        %run perfRequest

    case 'inform'
        reply = perfInform(msg);
        %run perfInform

    case 'subscribe'
        reply = perfSubscribe();
        %run perfSubscribe
```

```
    case 'agree'
        reply = perfAgree();
        %run perfAgree

    case 'refuse'
        reply = perfRefuse();
        %run perfRefuse

    case 'PPack'
        %ack = msg.content{1};
        ack=msg;
        labSend(ack,labindex-1,4);
        reply=[];

    case 'timer'
        labSend(msg,labindex-1)
        reply=[];

        %     case 'FB_RG'
        %         labSend(msg,1)
        %         reply=[];
        %
        %     case 'FB_PP'
        %         labSend(msg,1)
        %         reply=[];

    case 'notUnderstood'
        reply = [];
    otherwise
        reply = 'notUnderstood';

end


if isempty(reply)
    % If empty, do nothing
else
    % If 'reply' is not a structure, but also isn't empty, create a
message
    % based on what's in 'reply'
    if strcmp(reply,'notUnderstood')   %~isstruct(reply)
        switch reply
            case 'notUnderstood'  % Generic not understood
                performative= 'notUnderstood';
                content={};
%                 performative
%                 agent
%                 receiver
%                 content
                reply =
encodeMASmsg(performative,agent,receiver,content);
            otherwise
                warning('messageshell:reply',...
                    ['messageShell: Invalid ''reply'' value: %s\n'...
```

```
                    'Not taking any action.'], reply)
        end
    end

    % If 'reply' is a struct (either created just above or prior to
that),
    % send as a message
    if ischar(reply) && strcmp(reply,'notUnderstood') == 0
%isstruct(reply) || ischar(reply)
        %         message =
buildMessage(performative,agent,receiver,content);
        %         SendMASmsg(message,retries);
%         SendMASmsg(reply,retries);

sendMASmsg(reply,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
%         fprintf('\nMessage Sent to %s on
%d\n',receiver.ip,receiver.port)
    end

end
% % Take action based on 'reply'
% if isempty(reply)
%     % If empty, do nothing
% else
%     % If 'reply' is not a structure, but also isn't empty, create a
message
%     % based on what's in 'reply'
%     if ~isstruct(reply)
%         switch reply
%             case 'notUnderstood'  % Generic not understood
%                 performative='notUnderstood'; content={};
%             otherwise
%                 warning('messageshell:reply',...
%                     ['messageShell: Invalid ''reply'' value: %s\n'...
%                     'Not taking any action.'], reply)
%         end
%     end
%
%     % If 'reply' is a struct (either created just above or prior to
that),
%     % send as a message
%     if isstruct(reply)
%         %         message =
buildMessage(performative,agent,receiver,content);
%         %         SendMASmsg(message,retries);
%         SendMASmsg(reply,retries,reply.ip,reply.port);
%     end
%
% end

end
```

C.8.1 informData.m

```
function reply = informData(msg)

global agent
global a
switch msg.content{a,2}
%     case 'net'
%         switch msg.content{a,3}
%             case 'FTP'
%
%             otherwise
%                 reply = 'notUnderstood';
%         end

%     case 'Euld'
%         switch msg.content{a,3}
%             case 'FTP'
%             case 'actual'
%             otherwise
%                 reply = 'notUnderstood';
%         end

%     case 'inoutdata'
%         switch msg.content{a,3}
%             case 'FTP'
%             otherwise
%                 reply = 'notUnderstood';
%         end

%     case 'perftable'
%         switch msg.content{a,3}
%             case 'FTP'
%             case 'actual'
%             otherwise
%                 reply = 'notUnderstood';
%         end
%     case 'preferences'
%         switch msg.content{a,3}
%             case 'actual'
%             otherwise
%                 reply = 'notUnderstood';
%         end

%     case 'status'
%         switch msg.content{a,3}
%             case 'actual'
%             otherwise
%                 reply = 'notUnderstood';
%         end

    case 'performance'
        switch msg.content{a,3}
            case 'actual' % Not finished, done to get model working
```

```
                reply =
buildMessage('agree',agent,msg.sender,{'acknowledge'});
            otherwise
                reply = 'notUnderstood';
        end

%     case 'agentTable'
%         switch msg.content{a,3}
%             case 'actual'
%             otherwise
%                 reply = 'notUnderstood';
%         end

    case 'task'
        switch msg.content{a,3}
            case 'actual'

                for n = 1:numel(agent.agentTable)
                    if agent.agentTable{n}.ip == msg.sender.ip
                        if agent.agentTable{n}.port == msg.sender.port
                            agent.agentTable{n}.task =
msg.content{a,4};
                        end
                    end
                end

                reply =
buildMessage('agree',agent,msg.sender,{'acknowledge'});

            otherwise
                reply = 'notUnderstood';
        end

%     case 'agentinfo'
%         switch msg.content{a,3}
%             case 'actual'
%             otherwise
%                 reply = 'notUnderstood';
%         end

    otherwise
        reply = 'notUnderstood';
end
```

C.8.2 perfinform.m

```
function reply = perfInform(msg)
%This performs the perfInform script
global agent
global a
reply=[];
for a=1:size(msg.content,1)

    switch msg.content{a,1}
        case 'data'
            %reply = informData(msg);
            switch msg.content{a,2}
                %     case 'net'
                %         switch msg.content{a,3}
                %             case 'FTP'
                %
                %             otherwise
                %                 reply = 'notUnderstood';
                %         end

                %     case 'Euld'
                %         switch msg.content{a,3}
                %             case 'FTP'
                %             case 'actual'
                %             otherwise
                %                 reply = 'notUnderstood';
                %         end

                %     case 'inoutdata'
                %         switch msg.content{a,3}
                %             case 'FTP'
                %             otherwise
                %                 reply = 'notUnderstood';
                %         end

                %     case 'perftable'
                %         switch msg.content{a,3}
                %             case 'FTP'
                %             case 'actual'
                %             otherwise
                %                 reply = 'notUnderstood';
                %         end
                %     case 'preferences'
                %         switch msg.content{a,3}
                %             case 'actual'
                %             otherwise
                %                 reply = 'notUnderstood';
                %         end

                %     case 'status'
                %         switch msg.content{a,3}
                %             case 'actual'
                %             otherwise
                %                 reply = 'notUnderstood';
```

```
                %         end

                case 'performance'
                    switch msg.content{a,3}
                        case 'actual' % Not finished, done to get model
working
                            reply =
encodeMASmsg('agree',agent,msg.sender,{'acknowledge'});
                        otherwise
                            reply = 'notUnderstood';
                    end

                    %     case 'agentTable'
                    %         switch msg.content{a,3}
                    %             case 'actual'
                    %             otherwise
                    %                 reply = 'notUnderstood';
                    %         end

                case 'task'
                    switch msg.content{a,3}
                        case 'actual'

                            for n = 1:numel(agent.agentTable)
                                if agent.agentTable{n}.ip ==
msg.sender.ip
                                    if agent.agentTable{n}.rport ==
msg.sender.rport
                                        agent.agentTable{n}.task.val =
msg.content{a,4};
                                    end
                                end
                            end

                            reply =
encodeMASmsg('agree',agent,msg.sender,{'acknowledge'});

                        otherwise
                            reply = 'notUnderstood';
                    end

                    %     case 'agentinfo'
                    %         switch msg.content{a,3}
                    %             case 'actual'
                    %             otherwise
                    %                 reply = 'notUnderstood';
                    %         end

                case 'Euld'
                    temp=msg.content;
                    if numel(temp) == 3
                        newE = msg.content{3};
%
agent.memory.Euld=[newE(1:numel(newE)/2);newE(numel(newE)/2+1:end)];
                        agent.memory.Euld=newE;
```

```
                    elseif numel(temp) == 7
                        agent.memory.Euld=msg.content{3};
                        agent.memory.running=msg.content{4};
                        agent.memory.time=msg.content{5};
                        agent.memory.samplerate=msg.content{6};
                        agent.memory.twin=msg.content{7};
                        agent.memory.iterCnt=1;
                    end
                    if numel(agent.memory.Euld) > 1
                        Euld=agent.memory.Euld;
                        Euldvec =
[Euld(1:numel(Euld)/2);Euld(numel(Euld)/2+1:end)];
%                         for n=1:length(Euld)-1
%                             Euldvec(Euld(1,n)+1:Euld(1,n+1)) =
linspace(Euld(2,n),Euld(2,n+1),(Euld(1,n+1)-Euld(1,n)));
%                         end
                        agent.memory.Euld=Euldvec;
                        agent.memory.vectimer=tic;
                    end
                    %Agent struct sent to lab 1 with TAG=11;
                    labSend(agent,labindex-1,11)

                    reply=[];

                case 'beta'
                    agent.memory.beta=msg.content{3};
                    reply=[];

                case 'method'
                    agent.memory.method=msg.content{3};
                    reply=[];

                case 'agenttable'
                    daagenttable = msg.content{4};
                    for n=1:numel(daagenttable)
                        if strcmp(agent.ip,daagenttable{n}.ip) &&
agent.rport == daagenttable{n}.rport
                            row = n;
                        end
                    end
                    tmp = daagenttable{1};
                    daagenttable{1} = daagenttable{row};
                    daagenttable{row} = tmp;
                    agent.agentTable = daagenttable;
                    reply=[];
                otherwise
                    reply = 'notUnderstood';
            end

        case 'FB_data'
            %TAG=9;
            labSend(msg,labindex-1,9)
            reply=[];

        case 'FF_data'
            %TAG=7;
```

```
                labSend(msg,labindex-1,7)
                reply=[];

            case 'PP_data'
                %TAG=5;
                labSend(msg,labindex-1,5)
                reply=[];

            case 'GO_data'
                %TAG=3;
                labSend(msg,labindex-1,3)
                reply=[];

            case'errorHigh'
                labSend(msg.content{2},labindex-1,13)
                reply=[];

            case 'shutdown'
                reply=[];
                %reply.content = {'agree'};

            otherwise
                reply = 'notUnderstood';
        end

    end
```

C.8.3 perfRequest.m

```
function reply = perfRequest(msg)
%Runs perfRequest script
global agent
global a

for a=1:size(msg.content,1)
    switch msg.content{a,1}
        case 'data'
            %reply = requestData(msg);
            %needs "not common" section finished

            reply=[];

            switch msg.content{a,2}

                % Messages common to all agents
                case 'gains'
                    content={'data','gains','actual',agent.memory.K};
                case 'task'
                    content={'data','task','actual',agent.task.val};
                case 'status'
                    content={'data','status','actual',agent.status};
                case 'agenttable'

content={'data','agenttable','actual',agent.agentTable};
                case 'performance'

content={'data','performance','actual',agent.performance};
                case 'agentinfo'
                    content={'data','agentinfo','actual',agent};

                    %     % Not common to all
                    %     case 'net'
                    %         content={'data','net','FTP',comm.net};
                    %     case 'Euld'
                    %
content={'data','Euld','actual',comm.Euld};
                    % %     case 'Euld'
                    % %
content={'data','Euld','FTP',comm.Euld};
                    %     case 'inoutdata'
                    %
content={'data','inoutdata','FTP',comm.inoutdata};
                    %     case 'preferences'
                    %
content={'data','preferences','actual',comm.preferences};
                    %     case 'perftable'
                    %
content={'data','perftable','actual',comm.perftable};
                    % %     case 'perftable'
                    % %
content={'data','perftable','FTP',comm.perftable};
```

```
                otherwise
                    reply = 'notUnderstood';
            end

            % If reply doesn't already contain something, put a message
in it
            if isempty(reply)
                reply =
encodeMASmsg('inform',agent,msg.sender,content);
            end

        case 'become'
            %reply = requestBecome(msg);
            %this function is done
            reply = [];

            if ~strcmp(agent.status,'busy')
                % Change current task and send 'agree' to DA
                switch msg.content{a,2}
                    case 'nnAgent'
                        %In each case, the following line changes the
global task
                        %variable to the approriate task according to
the message
                        agent.task=struct('val','nnAgent',...
                            'time',now,'rank',msg.content{a,3});
                        performative='agree';
                    case 'controlAgent'
                        agent.task=struct('val','controlAgent',...
                            'time',now,'rank',msg.content{a,3});
                        performative='agree';
                    case 'interfaceAgent'
                        agent.task=struct('val','interfaceAgent',...
                            'time',now,'rank',msg.content{a,3});
                        performative='agree';
                    case 'monitoringAgent'
                        agent.task=struct('val','monitoringAgent',...
                            'time',now,'rank',msg.content{a,3});
                        performative='agree';
                    case 'delegationAgent'
                        agent.task=struct('val','delegationAgent',...
                            'time',now','rank',msg.content{a,3});
                        agent.agentTable{1}.task.val='delegationAgent';
                        fprintf('\nRequest to become DA processed\n')
                        performative='agree';
                    case 'databaseAgent'
                        agent.task=struct('val','databaseAgent',...
                            'time',now,'rank',msg.content{a,3});
                        performative='agree';
                    case 'freeAgent'
                        agent.task=struct('val','freeAgent',...
                            'time',now,'rank',msg.content{a,3});
                        performative='agree';
                    case 'feedbackAgent'
                        agent.task=struct('val','feedbackAgent',...
                            'time',now,'rank',msg.content{a,3});
```

```
                        performative='agree';
                    case 'gainoptimizerAgent'

agent.task=struct('val','gainoptimizerAgent',...
                        'time',now,'rank',msg.content{a,3});
                        performative='agree';
                    otherwise
                        reply = 'notUnderstood'; %not understood

                end
                agent.agentTable{1}.task.val=agent.task.val;
                agent.taskMax = 5;
                agent.taskCounter = 0;
                fprintf('\nRequest Switch completed\n')


                %     for n = 1:numel(agent.agentTable)
                %         if agent.agentTable{n}.ip == msg.sender.ip
                %             if agent.agentTable{n}.port ==
msg.sender.port
                %                 agent.agentTable{n}.task =
msg.content{a,2};
                %             end
                %         end
                %     end
                if isempty(reply)
                    agent.startup = 1;
                    reply =
encodeMASmsg(performative,agent,msg.sender,{});
                end
            else
                reply = encodeMASmsg('refuse',agent,msg.sender,'busy');
            end

        case 'ATable'
            save '\\Ada\Group\Robotics\MAS_Power_Group\matlab
code\Parallel MAS System\Task\Interface Agent\toClient.mat'
agent.agentTable
            reply=[];
        case 'PPin/out'
            reply=[];
        case 'taskChange'
        %This section is for handling requests from IA to DA to change
an
        %agent's task. Format for the content must be in the following

%content={taskChange,oldtask,location,port,rank,newtask,newrank}
            reply=[];
            if strcmp(agent.task.val, 'delegationAgent')
                receiver =
struct('ip',msg.content{a,3},'rport',msg.content{a,4},'task',struct('va
l',msg.content{a,1},'rank',msg.content{a,5}));
                content={'become',msg.content{a,6},msg.content{a,7}};
                reply=encodeMASmsg('request',agent,receiver,content);
            end
        otherwise
```

```
            reply = 'notUnderstood';
    end

end



%                 case 'storedata'
%                     switch msg.content{a,2}
%                         case 'net'
%                             backupData('net',comm);
%                             content={'senddata','net'};
%                         case 'Euld'
%                             backupData('Euld',comm);
%                             content={'senddata','Euld'};
%                         case 'inoutdata'
%                             backupData('inoutdata',comm);
%                             content={'senddata','inoutdata'};
%                         case 'performance'
%                             backupData('performance',comm);
%                             content={'senddata','performance'};
%                         case 'status'
%                             backupData('status',comm);
%                             content={'senddata','status'};
%                         case 'preferences'
%                             backupData('preferences',comm);
%                             content={'senddata','preferences'};
%                         case 'agenttable'
%                             backupData('agenttable',comm);
%                             content={'senddata','agenttable'};
%                         case 'perftable'
%                             backupData('perftable',comm);
%                             content={'senddata','perftable'};
%                         case 'agentinfo'
%                             backupData('agentinfo',comm);
%                             content={'senddata','agentinfo'};
%                         otherwise
%                             other=1;
%                     end
%                 case 'senddata'
%                     switch msg.content{a,2}
%                         case 'net'
%                             content={'agree'};
%                             sendData('net',comm);
%                         case 'Euld'
%                             content={'agree'};
%                             sendData('Euld',comm);
%                         case 'inoutdata'
%                             content={'agree'};
%                             sendData('inoutdata',comm);
%                         case 'performance'
%                             content={'agree'};
%                             sendData('performance',comm);
%                         case 'status'
%                             content={'agree'};
```

```
%                             sendData('status',comm);
%                         case 'preferences'
%                             content={'agree'};
%                             sendData('preferences',comm);
%                         case 'agenttable'
%                             content={'agree'};
%                             sendData('agenttable',comm);
%                         case 'perftable'
%                             content={'agree'};
%                             sendData('perftable',comm);
%                         case 'agentinfo'
%                             content={'agree'};
%                             sendData('agentinfo',comm);
%                         otherwise
%                             other=1;
%                     end

end
```

C.8.4 requestBecome.m

```
function reply = requestBecome(msg)
% reply = requestBecome(msg,comm)
% This function controls an agent's response to requests to become
another
% type of agent.
% Msg is the incoming msg structure to change,
% Reply returns a 'not understood' if it does not interpret the message
% correctly. Reply contains a message struct with performative 'refuse'
% and content 'busy' if the agent is too busy to respond.
% Reply returns a message struct with performative 'agree' if the
become
% order is interpreted correctly.
global agent
global a
reply = [];

if ~strcmp(agent.status,'busy')
    % Change current task and send 'agree' to DA
    switch msg.content{a,2}
        case 'nnAgent'
            %In each case, the following line changes the global task
            %variable to the approriate task according to the message
            agent.task=struct('val','nnAgent',...
                'time',now,'rank',msg.content{a,3});
            performative={'agree'};
        case 'controlAgent'
            agent.task=struct('val','controlAgent',...
                'time',now,'rank',msg.content{a,3});
            performative={'agree'};
        case 'interfaceAgent'
            agent.task=struct('val','interfaceAgent',...
                'time',now,'rank',msg.content{a,3});
            performative={'agree'};
        case 'monitoringAgent'
            agent.task=struct('val','monitoringAgent',...
                'time',now,'rank',msg.content{a,3});
            performative={'agree'};
        case 'delegationAgent'
            agent.task=struct('val','delegationAgent',...
                'time',now','rank',msg.content{a,3});
            agent.agentTable{1}.task='delegationAgent';
            fprintf('\nRequest to become DA processed\n')
            performative={'agree'};
        case 'databaseAgent'
            agent.task=struct('val','databaseAgent',...
                'time',now,'rank',msg.content{a,3});
            performative={'agree'};
        case 'freeAgent'
            agent.task=struct('val','freeAgent',...
                'time',now,'rank',msg.content{a,3});

            performative={'agree'};
        otherwise
            reply = 'notUnderstood'; %not understood
```

```
    end
    agent.agentTable{1}.task=agent.task.val;
    agent.taskMax = 5;
    agent.taskCounter = 0;
    fprintf('\nRequest Switch completed\n')


%     for n = 1:numel(agent.agentTable)
%         if agent.agentTable{n}.ip == msg.sender.ip
%             if agent.agentTable{n}.port == msg.sender.port
%                 agent.agentTable{n}.task = msg.content{a,2};
%             end
%         end
%     end
    if isempty(reply)
        agent.startup = 1;
        reply = buildMessage(performative,agent,msg.sender,{});
    end
else
    reply = buildMessage('refuse',agent,msg.sender,'busy');
end

end
```

C.8.5 requestData.m

```
function reply = requestData(msg)
%REQUESTDATA An MAS message parsing function for 'request','data'.
%   REPLY=REQUESTDATA(MSG,COMM) preforms an action and/or returns a
reply
%   to a message received which has PERFORMATIVE='request' and
%   CONTENT={'data',...}.
%
%   MSG is a message struct. COMM is the common agent file struct.
%
%   REPLY can be one of 3 thigs:
%       1) a single string that messageShell can use to create a
generic
%           message
%       2) a message structure containing the actual data
%       3) a message structure containing an FTP location for the data
%
%   See also REQUESTBECOME, PERFREQUEST, MESSAGESHELL
%
%   Jason Head, John Miller, 7/3/09

global agent
global a
reply=[];

switch msg.content{a,2}

    % Messages common to all agents
    case 'task'
        content={'data','task','actual',agent.task};
    case 'status'
        content={'data','status','actual',agent.status};
    case 'agenttable'
        content={'data','agenttable','actual',agent.agentTable};
    case 'performance'
        content={'data','performance','actual',agent.performance};
    case 'agentinfo'
        content={'data','agentinfo','actual',agent};

%     % Not common to all
%     case 'net'
%         content={'data','net','FTP',comm.net};
%     case 'Euld'
%         content={'data','Euld','actual',comm.Euld};
% %     case 'Euld'
% %         content={'data','Euld','FTP',comm.Euld};
%     case 'inoutdata'
%         content={'data','inoutdata','FTP',comm.inoutdata};
%     case 'preferences'
%         content={'data','preferences','actual',comm.preferences};
%     case 'perftable'
%         content={'data','perftable','actual',comm.perftable};
% %     case 'perftable'
% %         content={'data','perftable','FTP',comm.perftable};
```

```
    otherwise
        reply = 'notUnderstood';
end

% If reply doesn't already contain something, put a message in it
if isempty(reply)
    reply = buildMessage('inform',agent,msg.sender,content);
end

end
```

C.9 controlAgent.m

```
function controlAgent(method)

global agent rcomm scomm
global message
agent.taskMax = 100;
% fprintf('Control Agent:\n')

ack = 0;
n = 1;
go = 0;
Euldp = 0;
rgmode = 2;

% Wait for initial Euld
if isempty(agent.memory.Euld)
    fprintf('\nWaiting for initial Euld...\n')
    while ~go
        %Checks for agent info sent with TAG of 11; sent from
perfinform
        if labProbe(labindex+1,11)
            agent = labReceive(labindex+1,11);
            if ~isempty(agent.memory.Euld)
                Euld=agent.memory.Euld;
                idx=agent.memory.iterCnt;
                go = 1;
                fprintf('\nNew Euld received\n')
                [dnu,scomm] =
createCommChannels(method,agent.rport,agent.sport,5000,'rcomm',0);

            end
        end
    end
end

% if strcmp(method,'udp') && isempty(scomm)
%     [dnu,scomm] =
createCommChannels(method,agent.rport,agent.sport,5000,'rcomm',0);
% end

looptimer = tic;
Euld=agent.memory.Euld;
running=agent.memory.running;
time=agent.memory.time;
samplerate=agent.memory.samplerate;
beta=agent.memory.beta;
idx=agent.memory.iterCnt;
twin=agent.memory.twin;
dt = agent.memory.dt;

slowdown = agent.memory.slowdown;
fbmethod=agent.memory.method;
particles = agent.memory.particles;
iterations = agent.memory.iterations;
```

```
meth = agent.memory.meth;

if numel(Euld) > 1
    vec = 1;

        vectimer=agent.memory.vectimer;


else
    vec = 0;

end

if  running

    if vec
        if toc(vectimer)<Euld(1,end)
            Euldp = interp1(Euld(1,:),Euld(2,:),toc(vectimer));
        else
            Euldp=Euld(2,end);
        end
    else
        Euldp = Euld;
    end
    if vec
%         if ~strcmp(method,'GONo') && ~strcmp(method,'FBNo') &&
~strcmp(method,'FBYes')
        if labProbe(labindex+1,13); %mod(idx*fd*dt,twin) == 0 || idx ==
1
            tmp = labReceive(labindex+1,13);
%             try
%                 if idx/dt+(2*twin-1)/dt > numel(Euld)
%                     newE=Ecompress(Euld(idx/dt+twin/dt:end));
%                 else
%
newE=Ecompress(Euld(idx/dt+twin/dt:idx/dt+(2*twin-1)/dt));
%                 end
                newE = Ecompress([repmat(Euldp,1,5)
repmat(Euldp+1,1,500)]);

                fprintf('\nEuld sent to GO agent\n')
%                 save '\\ada\Group\MAS\matlab code\Parallel MAS
System\Task\Gain Optimizer Agent\newE.mat' newE
%             catch me
%               %Do nothing
%             end

[goip,goport]=agentLookup(agent.agentTable,'gainoptimizerAgent',1);
            receiver =
struct('ip',goip,'rport',goport,'task',struct('val','gainoptimizerAgent
'));
            content = {'data','Euld',[newE(1,:) newE(2,:)]};
            if ~isempty(goip) && ~isempty(goport)
                msg = encodeMASmsg('inform',agent,receiver,content);
```

```
sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
                fprintf('\nMessage sent to gainoptimzer \n')
%             else
%
[daip,daport]=agentLookup(agent.agentTable,'delegationAgent',1);
%                 receiver =
struct('ip',daip,'port',daport,'task','delegationAgent');
%                 content = {'data','agenttable'};
%                 if ~isempty(daip) && ~isempty(daport)
%                     msg =
buildMessage('request',agent,receiver,content);
%                     SendMASmsg(msg,1)
%                 end
            end
        end
%         end
    end

    if isempty(agent.memory.oldEuld')
        agent.memory.oldEuld = 0;
    end
%     display(agent.memory.oldEuld)
    %% HPSO
    if agent.memory.oldEuld ~= Euldp
        agent.memory.oldEuld = Euldp;
        ustar = hpso(Euldp);
        Ed = ustar(3);
        Pd = ustar(4);

        % Calculate optimized outputs (u*) and setpoints Ed and Pd

        %% Loadm Calc
        load '\\Ada\Group\MAS\Gomes\Test\ResisMap.mat'
        y =ResisMap(2:32,2);
        x = ResisMap(2:32,1);
        resist = 5^2./Euldp;
        stepvec = round(interp1(y,x,resist));
        rhod = stepvec/340;


        %%

        fprintf('\nEuld: %2.4f Pd: %2.4f \nustar: [%1.3f %1.3f
%1.3f]\n',Ed,Pd,ustar(n,1),ustar(n,2),rhod)
        if vec
            fprintf('Time: %4.2f seconds / %4.2f seconds
\n',toc(vectimer),Euld(1,end))
        end
        yd = [Pd Ed rhod];
        time=toc(looptimer);
        % Wait for ack from power plant
        %         while (ack == 0 && running == 1)
```

```
        %Adding 'method' to content array - 8/30/2010
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% Send to Feedback
        [fbip,fbport]=agentLookup(agent.agentTable,'feedbackAgent',1);
        receiver =
struct('ip',fbip,'rport',fbport,'task',struct('val','feedbackAgent','ra
nk',1));
        content =
{'FF_data',yd,ustar,running,time,samplerate,Euldp,fbmethod};
        if ~isempty(fbip) && ~isempty(fbport)
            msg = encodeMASmsg('inform',agent,receiver,content);

sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
            fprintf('Message sent to feedback \n')
        else
            %
[daip,daport]=agentLookup(agent.agentTable,'delegationAgent',1);
            %         receiver =
struct('ip',daip,'rport',daport,'task',struct('val','delegationAgent','
rank',1));
            %         content = {'data','agenttable'};
            %         if ~isempty(daip) && ~isempty(daport)
            %             msg =
encodeMASmsg('request',agent,receiver,content);
            %
sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
            %         end
        end
    end
    wait = dt-toc(looptimer);
    pausing = tic; while toc(pausing) < wait; end



    %     % Wait for new Euld
    %     go = 0;
    %     fprintf('\nWaiting for new Euld...\n')
    %     while (go == 0 && running == 1)
    %         if labProbe
    %             message = labReceive;
    %             if isstruct(message) &&
strcmp(message.performative,'inform') &&
strcmp(message.content{1},'data')
    %                 if strcmp(message.content{2},'Euld')
    %                     Euld = message.content{3};
    %                     running = message.content{4};
    %                     time = message.content{5};
    %                     samplerate = message.content{6};
    %                     go = 1;
    %                     fprintf('\nNew Euld received\n')
    %                     fprintf('Euld: %d\n',Euld)
    %                     if numel(Euld) > 1
    %                         vec = 1;
    %                         idx = 0;
    %                     else
    %                         vec = 0;
```

```
    %                     end
    %                 elseif strcmp(message.content{2},'beta')
    %                     beta = message.content{3};
    %                 end
    %             end
    %         end
    %     end
end
% fprintf('\nStopping...\n')
if ~running
    [fbip,fbport]=agentLookup(agent.agentTable,'feedbackAgent',1);
    receiver =
struct('ip',fbip,'rport',fbport,'task',struct('val','Power
Plant','rank',1));
    yd=[90 90 10];
    ustar=[0 0 0];
    content = {yd,ustar,running,time,samplerate,Euld};
    if ~isempty(fbip) && ~isempty(fbport)
        msg = encodeMASmsg('inform',agent,receiver,content);

sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
    end
    msg = encodeMASmsg('PowerPlant',agent,receiver,content);

sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
end

% fprintf('\nStopped\n')

% fprintf('And on to the next iteration...\n')
end
```

C.9.1 EPopwin.m

```
function [pmax pmin] = EPopwin(Euld)

range = .10:.10:1.80;

Pmax = [.10  95; ...
        .20  95; ...
        .30  95; ...
        .40  95; ...
        .50  98.5; ...
        .60  102; ...
        .70  103.5; ...
        .80  105; ...
        .90  107; ...
        1.00 109; ...
        1.10 111; ...
        1.20 113; ...
        1.30 115; ...
        1.40 115; ...
        1.50 115; ...
        1.60 115; ...
        1.70 115; ...
        1.80 115];

Pmin = [.10  40; ...
        .20  41.5; ...
        .30  43; ...%(36.3+32)/2; ...
        .40  44.5; ...
        .50  46; ...
        .60  47.5; ...
        .70  49; ...
        .80  50.5; ...
        .90  52; ...
        1.00 53.5; ...
        1.10 55; ...
        1.20 56.5; ...
        1.30 58; ...
        1.40 59.5; ...
        1.50 60; ...
        1.60 60; ...
        1.70 60; ...
        1.80 60];

pmax = interp1(range,Pmax(:,2),Euld);
pmin = interp1(range,Pmin(:,2),Euld);
```

C.9.2 hpso.m

```
function [f]=hpso(Euld)
%HPSO Hybrid-PSO for optimizing power plant inputs.
%   F=HPSO(EULD) optimizes power plant inputs using a hybrid particle
%   swarm optimization at the specified "unit load demand" EULD. Hybrid
%   refers to the fact that, at each step, the "worst" half of the
%   particles are moved to the same location as the "better" half,
while
%   maintaining their personal best and current velocity (i.e. only
%   position changes).
%
%   Required Inputs:
%       EULD - Power "unit load demand"
%
%   Optional Inputs:  (specified in 'parameter',value pairs)
{default}
%       'particles' - number of particles to use                  {40}
%       'iterations' - number of iterations of PSO                {130}
%       'method' - 1=MATLAB ode solver, 2=neural network          {1}
%       'model' - 1=exact ode, 2=approximate ode                  {1}
%       'net' - a neural network object (see NET)                 {}
%       'skew' - a distortion factor for testing the              {1}
%                adaptability of the system
%       'preferences' - the beta values for HPSO (there are 4) {[1 .5 1
0]}
%
%   Outputs:
%       F - vector containing the optimal inputs, outputs, and delta
%           (the maximum difference of all objective functions, i.e.
error)

% Set defaults
particles=100;
% iterations=130;
iterations=140;

graph=0;
tic;

load '\\Ada\Group\MAS\Gomes\Agents\Control Agent\ffnet.mat'
% Define preferences
% beta = ones(particles,1)*[1 .5 1 0];
beta = ones(particles,1)*[1 .1 0 .4];

% Get optional inputs
% for n=1:2:nargin-1
%     switch varargin{n}
%         case 'particles'
%             particles=varargin{n+1};
%         case 'beta'
%             beta=ones(particles,1)*varargin{n+1};
%         case 'iterations'
%             iterations=varargin{n+1};
%         case 'method'
%             method=varargin{n+1};
```

```
%         case 'model'
%             model=varargin{n+1};
%         case 'net'
%             ffnet=varargin{n+1};
%         case 'skew'
%             skew=varargin{n+1};
%         case 'preferences'
%             beta=ones(particles,1)*varargin{n+1};
%         case 'graph'
%             graph=1;
%     end
% end

    ps1 = struct('name'   ,'mapminmax' ,...
                 'xrows'  ,2           ,...
                 'xmax'   ,[800;400]  ,...
                 'xmin'   ,[0;0]     ,...
                 'xrange' ,[]          ,...
                 'yrows'  ,2           ,...
                 'ymax'   ,1           ,...
                 'ymin'   ,-1          ,...
                 'yrange' ,2           );
    ps2 = struct('name'   ,'mapminmax' ,...
                 'xrows'  ,2           ,...
                 'xmax'   ,[3;115]   ,...
                 'xmin'   ,[0;0]  ,...
                 'xrange' ,[]          ,...
                 'yrows'  ,2           ,...
                 'ymax'   ,1           ,...
                 'ymin'   ,-1          ,...
                 'yrange' ,2           );
    ps1.xrange = abs(ps1.xmax-ps1.xmin);
    ps2.xrange = abs(ps2.xmax-ps2.xmin);
%%
f = wingen(Euld);
u1min = f(:,1);
u1max = f(:,2);
u2min = f(:,3);
u2max = f(:,4);

pos = [u1min + (u1max-u1min).*rand(particles,1)... % Random intitial
    u2min + (u2max-u2min).*rand(particles,1)];    % positions for the
     % particles
wmax=.8; % Maximum weight value for the weighting function
wmin=.3; % Minimum weight value for the weighting function
c1=.5;    % Weighting factor #1
c2=.5;    % Weighting factor #2

% Initialize the velocity vector
vel = zeros([particles,2]);

%% NN sim
y = mapminmax('reverse',sim(ffnet,mapminmax('apply',pos',ps1)),ps2);
E = y(1,:)';
P = y(2,:)';
```

```
pbest = [pos E P];

%% eval
% Define objective functions
Jmin = ones(particles,1)*[0 u1max/1000 u2min/1000 0];
J = [abs(Euld-pbest(:,3))/3 pbest(:,1)/1000 pbest(:,2)/1000
abs(pbest(:,4)-80)/115];% J = [J1 J2 J3 J4]

% Calculate the initial delta vector for the initial particle points.
% Also calculate the initial personal bests of the particles
delta=sum(beta.*abs(J-Jmin),2);
pbest=[pbest delta];

% Calculate the initial global best of all the particles
[gbestval gbestindex]=min(delta);
gbest=pbest(gbestindex,1:2);

%% loop
for n=1:iterations
%     fprintf('\nHPSO Iteration: %d\nEuld: %d\n',n,Euld)
    % Sort data to obtain best deltas
    [tmp deltaindex]=sort(delta) ;

    % Replace worst half of positions with the values of the best half
    pos(deltaindex(ceil(particles/2)+1:particles),:) =
pos(deltaindex(1:ceil(particles/2)),:);

    % Caluclate velocities
    pbestvel=pbest(:,1:2)-pos;
    gbestvel=ones(particles,1)*gbest(:,1:2)-pos;
    w=wmax-((wmax-wmin)/iterations)*n;
    vel = w.*vel + c1.*rand.*(pbestvel)+c2.*rand.*(gbestvel);

    % Move particles
    pos=pos+vel;
    if graph
        scatter(pos(:,1),pos(:,2))
        axis([u1min u1max u2min u2max])
%         axis([0 1 0 1 0 1])
        drawnow
%         pause(0.05)
%         pause
    end


    tmp=[find( pos(:,1) > u1max )];
    if ~isempty(tmp)
        over = pos(tmp,1)-u1max;
        pos(tmp,1) = pos(tmp,1) - 2*over;
        vel(tmp,1)=zeros(length(tmp),1);
    end
    tmp=[find(pos(:,1) < u1min )];
    if ~isempty(tmp)
        under = u1min-pos(tmp,1);
        pos(tmp,1) = pos(tmp,1) + 2*under;
```

```
        vel(tmp,1)=zeros(length(tmp),1);
    end
    tmp=[find( pos(:,2) > u2max )];
    if ~isempty(tmp)
        over = pos(tmp,2)-u2max;
        pos(tmp,2) = pos(tmp,2) - 2*over;
        vel(tmp,2)=zeros(length(tmp),1);
    end
    tmp=[find(pos(:,2) < u2min )];
    if ~isempty(tmp)
        under = u2min-pos(tmp,2);
        pos(tmp,2) = pos(tmp,2) + 2*under;
        vel(tmp,2)=zeros(length(tmp),1);
    end


            y =
mapminmax('reverse',sim(ffnet,mapminmax('apply',pos',ps1)),ps2);
            E = y(1,:)';
            P = y(2,:)';

    check = [E P];

    % Recalculate objective functions
   J = [abs(Euld-check(:,1))/3 pos(:,1)/1000 pos(:,2)/1000
abs(check(:,2)-80)/115];

    % Recalculate deltas
    delta = sum(beta.*abs(J-Jmin),2);

    % Checks to see if there are new personal bests better (smaller)
    % than the old ones
    %     if method == 1 % ODE Solver method
    tmp = find(delta < pbest(:,5));
    %     elseif method == 2 % Neural Network method
    %         tmp = find(delta < pbest(:,6));
    %     end

    % If there are better personal bests they are updated
    if ~isempty(tmp)
        pbest(tmp,:) = [pos(tmp,:) check(tmp,:) delta(tmp)];
    end

    % If there are personal bests better than the previous global best
    % the best one is selected and stored as the new global best
    if min(delta) < gbestval
        [gbestval gbestindex]=min(delta);
        gbest=pbest(gbestindex,1:4);
    end
end

f=pbest(gbestindex,:);
u1 = f(1);
u2 = f(2);

E = f(3);
```

```
P = f(4);
f = [u1 u2 E P];
fprintf(1,'HPSO time: %f',toc)

% g = L;
% kssu1 = 0.7965973943;
% kssu2 = 1.1814399210;
% kssu3 = 1.1420265349;
% f=f./[kssu1 kssu2 kssu3 1 1 1];
```

C.9.3 wingen.m

```
function f=wingen(E,varargin)
%-------------------------------------------------------------------------
----
% Generate Power-Pressure combinations with the specified stepsize
within
% the Power-Pressure operating window from Fig. 3 of the paper entitled
% 'Multiobjective Control of Power Plants Using Particle Swarm
Optimization
% Techniques'
%
% Generates E and P values that can be used to calulate combinations of
% control inputs that are bounded by the Power-Pressure window from
Fig. 3
%
% Inputs/Outputs !!!
%-------------------------------------------------------------------------
----
% skew=1;
%
% kssu1 = 0.7965973943;
% kssu2 = 1.1814399210;
% kssu3 = 1.1420265349;

% Get optional inputs
for n=1:2:nargin-1
    switch varargin{n}
        case 'skew'
            skew=varargin{n+1};
    end
end

% % Generate Power and Pressure values between bounds for Power from 0
to 35
% % MW
% if E <= 35
%     Pmin=30;
%     Pmax=240-.5*E;
% end
% % Generate Power and Pressure values between bounds for Power from 36
to
% % 180 MW
% if E > 35 && E <= 180
%     Pmin=3.4483+((110/145)*E);
%     Pmax=240-.5*E;
% end

[Pmax Pmin] = EPopwin(E);

% Calculates admissible combinations of control  variables point-by-
point
% for the three control inputs u1, u2, and u3
u2min=50;
%u3min=((1.1*u2min-0.19)*Pmin)/141;
u1min=200;
```

```
u2max=350;
%u3max=((1.1*u2max-0.19)*Pmax)/141;
u1max=800;

% Slightly skewed model to allow for system correction
% u1min=u1min*skew;u2min=u2min*skew;u3min=u3min*skew;
% u1max=u1max*skew;u2max=u2max*skew;u3max=u3max*skew;

% Discards combinations with individual control values higher than the
% acceptable range of [0 1]
if u1min > u1max
    tmp=u1min;
    u1min=u1max;
    u1max=tmp;
end
if u2min > u2max
    tmp=u2min;
    u2min=u2max;
    u2max=tmp;
end
% if u3min > u3max
%     tmp=u3min;
%     u3min=u3max;
%     u3max=tmp;
% end

% if u1max >= 1
%     u1max=1;
% end
% if u1min >= 1
%     u1min=1;
% end
% if u2max >= 1
%     u2max=1;
% end
% if u2min >= 1
%     u2min=1;
% end
% if u3max >= 1
%     u3max=1;
% end
% if u3min >= 1
%     u3min=1;
% end


f=[u1min u1max u2min u2max E Pmin Pmax];
```

C.10 delegationAgent

```
function delegationAgent(method)
global agent rcomm scomm

fprintf('\n Delegation agent. Iteration %d\n',...
    agent.taskCounter)

% Set the task iteration limit before reverting to free agent
agent.taskMax = 10;

%Tasklist needs to be in this format [{'.......'} {'......'} ]
tasklist=[{'controlAgent'} {'feedbackAgent'}];
%tasklist=[{'interfaceAgent'} {'databaseAgent'} {'nnAgent'}
{'controlAgent'}...
%          {'monitoringAgent'} {'feedbackAgent'}];

%Declaring empty arrays
FA=[];
Neededtasks=[];
notfound=1;

if strcmp(method,'udp') && isempty(scomm)
    [dnu,scomm] =
createCommChannels(method,agent.rport,agent.sport,5000,'rcomm',0);
end

agentTable=agent.agentTable;
% save '\\ada\Group\MAS\matlab code\Parallel MAS System\Task\Interface
Agent\agentTable.mat' agentTable

if ~isempty(agent.agentTable)

    %This top loop finds the list of all needed tasks
    for n=1:numel(tasklist)
        for m=1:numel(agent.agentTable)
            if strcmp(agent.agentTable{m}.task.val, tasklist{n})
                notfound=0;
            end
        end
        if notfound == 1
            Neededtasks=[Neededtasks {tasklist{n}}];
        end
        notfound=1;
    end

    Neededtasks
    %If we have some needed tasks, we first find all available
freeagents,
    %then we send a message to first available one to become needed
task
    if ~isempty(Neededtasks)

        %Creating index of all freeagents
```

```
        for n=1:numel(agent.agentTable)
            %Used 'any' command last time but dont remember why%
            ndx(n,1)=strcmpi(agent.agentTable{n}.task.val,
'freeAgent');
        end

        for n=1:numel(Neededtasks)

            f=find(ndx,1,'first');
            ndx(f,1)=0;             %clearing index of available free
agent

            %Need to add section here in future to deal with the
situation
            %where no free agents are available

            %Sending message to freeagent to change task
            if ~isempty(f)
                FA=agent.agentTable{f};
                receiver =
struct('ip',FA.ip,'rport',FA.rport,'task',struct('val','freeAgent','ran
k',1));
                content={'become', Neededtasks{n},1};
%                 msg = buildMessage('request',agent,receiver,content);
                msg = encodeMASmsg('request',agent,receiver,content);

sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
                fprintf('\n %s told to be %s \n',FA.ip, Neededtasks{n})
            end
        end
    end


    pause(5);
%     labSend('done',2);
end

%%%%%%%%%%%%%%%%%%%%%%%%OLD CODE %%%%%%%%%%%%%%%%%%%%
% for n=1:numel(tasklist)
%     found =0;
%     for a=1:numel(agent.agentTable)
%         if strcmp(agent.agentTable{a}.task, tasklist{n})
%             found=found+1;
%         end
%     end
%
%     %Looking for freeAgent to assign to needed task
%     if found==0
%         for i=1:numel(agent.agentTable)
%             ndx(i,1)=any(strcmpi(agent.agentTable{i}.task,
'freeAgent'));
%         end
%
%         f=find(ndx, 1, 'first');
%
```

```
%         if ~isempty(f)
%             FA=agent.agentTable{f};
%             receiver =
struct('ip',FA.ip,'port',FA.port,'task','freeAgent');
%             content={'become', tasklist{n},1};
%             msg = buildMessage('request',agent,receiver,content);
%             SendMASmsg(msg,1)
%
%
%             %Updating own agentTable
%             FA.task=tasklist{n};
%             agent.agentTable{f}.task=FA.task;
%         end
%
%         %If no free Agents are available, search among remaining
active
%         %agents for lowest priority agent to replace
%         %    if isempty(f)
%         %       for i=numel(agent.agentTable)
%
%         %Send message to freeAgent requesting to change task
%
%     end
% end
%     found=0;
end
```

C.11 feedbackAgent

```
function feedbackAgent(method)

global agent rcomm scomm
global message
agent.taskMax = 30;
fprintf('This does the feedback agent task!\n')


dt = agent.memory.dt;

if strcmp(method,'udp') && isempty(scomm)
    [dnu,scomm] =
createCommChannels(method,agent.rport,agent.sport,5000,'rcomm',0);
end

% Initial variables
pip = '129.62.75.244'
pport = 42001;
com=udp(pip,pport,'LocalPort',42001);
fopen(com)
go = 1;
start = 0;
NNTS = zeros(6,2);
fprintf('\nReference Governor should set data output to port:%d and
ip:%s\n',agent.rport,agent.ip)
% Wait for initial Euld
fprintf('\nWaiting for Reference Governor data...\n')



% while ~go
%     %Checking to see if FF_data has been sent; FF_data Tag =7;
%     if labProbe(labindex+1,7)
%         message = labReceive(labindex+1,7);
%         typeofdata = message.content{1};
%
%         if strcmp(typeofdata,'FF_data')
%             yd = message.content{2};
%             Pd = yd(1); Ed = yd(2); rhod = yd(3);
%             E = Ed; P = Pd; rho = rhod;
%             ustar = message.content{3};
%             ustar = ustar(1:3); uff = ustar;
%             u1 = uff(1); u2 = uff(2); u3 = uff(3);
%             L = 6.5365*rho+3000*((1/rho-0.001538)/((1/(0.8*P-25.6))-
0.001538))+5.5556*((0.85*u2-0.14)*P+45.59*u1-2.51*u3-2.09)-3275;
%             running = message.content{4};
%             time = message.content{5};
%             samplerate = message.content{6};
%             Euld = message.content{7};
%             fbmethod= message.content{8};
%             go = 1;
%         end
%     end
```

```
% end



Kp=[0 .1;...
    .1 0];
Ki=[.005 .001;...
    .0005 0];
Kd=[0 0 ;...
    0 0];


intlength=100;
error=zeros(2,intlength);
Ed=-1;Pd=-1;
FbOn=0;
uff=[0 0];

twin = agent.memory.twin;
gogo = clock; gogo(6) = gogo(6) - twin*4/3;
pmsgtimer = tic;
totaltime = tic;
sendtime=tic;
savetime=tic;
dataLog=[];
loadrange =[0.06 .9];
delta=[0;0];
uffn=[];
uffo=[];

%% Loop
while go
   %% Data In
    if labProbe(labindex+1)
        message = labReceive(labindex+1);
        typeofdata = message.content{1};
       % fprintf('\nReceived %s from message thread\n',typeofdata);

        switch(typeofdata)
            case 'FF_data'
                yd = message.content{2};


                    Pd = yd(1); Ed = message.content{7};
                    ustar = message.content{3};
                    uff = ustar(1:2)/1000;
                    FbOn = 1;
                    Loadmd = yd(3);
                    fprintf(1,'from CA: Ed= %5.3f Pd=%5.3f uff= %5.3f
%5.3f Load= %5.3f \n',...
                        [Ed Pd uff(1) uff(2) Loadmd])
%                     samplerate = message.content{6};
%                     Euld = message.content{7};
%
%                fbmethod = message.content{8};
```

```
            case 'PP_data'


                P = message.content{6};
                E = message.content{12}*message.content{13};
                V=message.content{13};
                T=message.content{9};
                u1 = message.content{2}/1000;
                u2 = message.content{3}/1000;
                Loadm = message.content{5}/340;

               % NNTS = [NNTS(:,2) [P;E;L;u1;u2]];

%                 if sum(NNTS(:,1)) > 0
%
[nnip,nnport]=agentLookup(agent.agentTable,'neuralnetworkAgent',1);
%                     receiver =
struct('ip',nnip,'rport',nnport,'task',struct('val','neuralnetworkAgent
','rank',1));
%                     content =
{'FB_data',NNTS(1),NNTS(2),NNTS(3),NNTS(4),NNTS(5),NNTS(6),...
%
NNTS(7),NNTS(8),NNTS(9),NNTS(10),NNTS(11),NNTS(12)};
%                     if ~isempty(nnip) && ~isempty(nnport)
%                         msg =
encodeMASmsg('inform',agent,receiver,content);
%
sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
%                     end
%                 end
%
                fprintf(1,'from PP: E= %5.3f P=%5.3f V= %5.3f T= %5.3f
\n',[E P V T])
                  save toClient E P V T u1 u2 uff Ed Pd Loadm FbOn Kp
Ki Kd
                dataLog = [dataLog [
cell2mat(message.content(2:end))';toc(totaltime);E;Ed;Pd;uff']];

                if toc(savetime)>30
                    savetime=tic;
                    save(datestr(now,30), 'dataLog')
                    dataLog=[];
                end

            case 'GO_data'
                Kptemp = message.content{2}; Kp=
[Kptemp(1:2);Kptemp(3:4)];
                Kitemp = message.content{3}; Ki=
[Kitemp(1:2);Kitemp(3:4)];
                Kdtemp = message.content{4}; Kd=
[Kdtemp(1:2);Kdtemp(3:4)];

%                 agent.memory.K(:,:,1) = Kp; agent.memory.K(:,:,2) =
Ki;
```

```
                display('New gain Matrices recieved\n')
                display(Kp)
                display(Ki)
                display(Kd)


        end


    end

    %% Data Out
    %Send msg to powerplant%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    if FbOn && (toc(sendtime) >= 1)

        sendtime = tic;
        % normalize everything
        Edn=Ed/3;
        Pdn=Pd/120;
        En=E/3;
        Pn=P/120;
        % uff is normalized when the agent recieves it
          %PI
          err=[Edn-En;Pdn-Pn];
          error=[err error(:,1:end-1)];
          delta=[u1 u2]'+
Kp*error(:,1)+Ki*sum(error,2)+Kd*((error(:,end)-error(:,end-
5))/.825*5);

            u=delta';
%         u = delta' + (uffn-uffo)';
%         uffo=uffn;

        sendme=[u(1) u(2) Loadmd];
        %SafetyCheck
        if sendme(1)-u1>.1;sendme(1)=u1+.1*sign(sendme(1)-u1);end
        if sendme(1)>1;sendme(1)=1; end
        if sendme(1)<.15;sendme(1)=.15;end
        if sendme(2)-u2>.05;sendme(2)=u2+.05*sign(sendme(2)-u2);end
        if sendme(2)>.4; sendme(2)=.4;end
        if sendme(2)<.04; sendme(2)=.04;end
        if sendme(3)>loadrange(end) ;sendme(3)=loadrange(end); end
        if sendme(3)<loadrange(1) ;sendme(3)=loadrange(1); end

        if sum(abs([u1 u2 Loadm]-sendme))>.001
            fprintf(com,'%5.3f$%5.3f$%5.3f',sendme)
            fprintf(1,'to PP : "%5.3f %5.3f %5.3f"\n',sendme)
        end


        %         if e1p > agent.memory.error(1) || e2p >
agent.memory.error(2) || e3p > agent.memory.error(3)
```

```
% %             display([e1p e2p e3p])
%             if etime(clock,gogo) > twin+15
%                 gogo = clock;
%
[caip,caport]=agentLookup(agent.agentTable,'controlAgent',1);
%                 receiver =
struct('ip',caip,'rport',caport,'task',struct('val','controlAgent','ran
k',1));
%                 content = {'errorHigh','true'};
%                 if ~isempty(caip) && ~isempty(caport)
%                     msg =
encodeMASmsg('inform',agent,receiver,content);
%
sendMASmsg(msg,scomm,receiver.ip,receiver.rport,agent.sport,1,method)
%                 end
%             end
%         end

    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end


fprintf('And on to the next iteration...\n')

end
```

C.12 freeAgent

```
function freeAgent(method)

global agent rcomm scomm

fprintf('\nFree agent. Iteration %d\n',agent.taskCounter)

% Set the task iteration limit before reverting to free agent
agent.taskMax = 5;

% Create list of IP address for all agent computers
ipList = {'129.62.150.128';'129.62.150.129'; ...
    '129.62.151.107';'129.62.151.191';'129.62.150.181';...

'129.62.151.171';'129.62.151.52';'129.62.151.108';'129.62.151.131'};
%ipList = getIP([129,129; 62,62; 150,151; 107,191]);

% Set port range
portRange = [10000 10010];

if strcmp(method,'udp') && isempty(scomm)
    [~,scomm] =
createCommChannels(method,agent.rport,agent.sport,5000,'rcomm',0);
end

% -----------------------------------------------------------------
% Need to communicate with agent computers to determine which
% agents exist and whether the delegation agent exists.
% If not, check ip's of computers to see if this computer's ip is
% higher -> If ip is highest: agent.task = delegationAgent;
% -----------------------------------------------------------------

% Poll agent computers to determine each one's task
fprintf('Polling computers for Agents: \n')
for n=1:length(ipList)
    % Loop over port range
    for m=portRange(1):portRange(2)

        % Specify receiver
        receiver =
struct('ip',ipList{n},'rport',m,'task',struct('val','task','rank',1));

        % Specify message content
        content = {'data','task'};

        % Build message
%         msg = buildMessage('request',agent,receiver,content);
        msg = encodeMASmsg('request',agent,receiver,content);
        % Send the message
        sendMASmsg(msg,scomm,receiver.ip,receiver.rport,1,method);

        pause(.05)
    end
```

```
end

% Wait for replies
pause(.5)

% --------------------------------
% if delegationAgent doesn't exist
%    check agent list
%    if lowest ip
%        agent.task = delegation;
% --------------------------------

% Check for delegation agent
if agent.taskCounter >4
    DA = [];
    for n = 1: size(agent.agentTable,2)
        if strcmp(agent.agentTable{n}.task.val , 'delegationAgent') &&
agent.agentTable{n}.task.rank == 1
            DA = agent.agentTable{n};
        end
    end

    if isempty(DA)
        % Check for other free agents
        FA = [];
        for n = 1: size(agent.agentTable,2)
            if strcmp(agent.agentTable{n}.task.val , 'freeAgent')
                FA{n} = agent.agentTable{n};
            end
        end

        % Determine which free agent has the lowest IP
        [lowIdx lowIP lowPort] = lowestIP(FA);


        % Check to see if this agent has the lowest
        if strcmp(agent.ip, lowIP) && agent.rport == lowPort
            % Set task to delegation agent
            %%%%%%%%%% Modifing this section of code%%%%%%%%%%%%%
            %        agent.task.val = 'delegationAgent';
            %        agent.task.time = now;
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            receiver =
struct('ip',agent.ip,'rport',agent.rport,'task',struct('val',agent.task
.val,'rank',agent.task.rank));
            % Bulid message content
            content={'become', 'delegationAgent',1};
%             msg = buildMessage('request',agent,receiver,content);
            msg = encodeMASmsg('request',agent,receiver,content);
            sendMASmsg(msg,receiver.ip,receiver.rport,1,method)
            pause(.2)
        end
    end
end
end
```

C.13 interfaceAgentGUI

```
function varargout = interfaceAgentGUI(varargin)
% INTERFACEAGENTGUI M-file for interfaceAgentGUI.fig
%      INTERFACEAGENTGUI, by itself, creates a new INTERFACEAGENTGUI or
raises the existing
%      singleton*.
%
%      H = INTERFACEAGENTGUI returns the handle to a new
INTERFACEAGENTGUI or the handle to
%      the existing singleton*.
%
%      INTERFACEAGENTGUI('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in INTERFACEAGENTGUI.M with the given
input arguments.
%
%      INTERFACEAGENTGUI('Property','Value',...) creates a new
INTERFACEAGENTGUI or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before interfaceAgentGUI_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to interfaceAgentGUI_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to running (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help interfaceAgentGUI

% Last Modified by GUIDE v2.5 14-Apr-2012 07:29:16

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @interfaceAgentGUI_OpeningFcn, ...
    'gui_OutputFcn',  @interfaceAgentGUI_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before interfaceAgentGUI is made visible.
function interfaceAgentGUI_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to interfaceAgentGUI (see VARARGIN)
global rcomm scomm method iaport iasport

% Choose default command line output for interfaceAgentGUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes interfaceAgentGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% Creates objects for tcp\ip or udp communication
method = 'udp';
inport = 10004; outport = 10104; error = 1;
while error == 1
    try
        [rcomm,scomm] = createCommChannels(method,inport,outport,1);
        error = 0;
    catch me
        display(me.message)
        error = 1;
        inport = inport + 1;
        outport = outport + 1;
    end
end
iaport = inport; iasport = outport;

set(gcf,'CloseRequestFcn',@closefcn)
function closefcn(src,event)
global rcomm scomm
try
    if exist('rcomm','var')
        if strcmp(rcomm.Status,'open')
            fclose(rcomm)
        end
    end
    if exist('scomm','var')
        if strcmp(scomm.Status,'open')
            fclose(scomm)
        end
    end
catch me
    display(me.message)
end
```

```
delete(gcf)


% --- Outputs from this function are returned to the command line.
function varargout = interfaceAgentGUI_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as
a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% % --- Executes on button press in pushbutton1.
% function pushbutton1_Callback(hObject, eventdata, handles)
% % hObject    handle to pushbutton1 (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
% filepath = get(handles.edit1,'String');
% load(filepath) % loads variables 'demandCurve'
% Euld = demandCurve; clear demandCurve
% for n=1:length(Euld)-1
%     E(Euld(1,n)+1:Euld(1,n+1)) =
linspace(Euld(2,n),Euld(2,n+1),(Euld(1,n+1)-Euld(1,n)));
% end
% dt = str2double(get(hadles.edit9,'String'));
% plot(handles.axes2,dt:dt:numel(E)*dt,E)
% xlabel(handles.axes2,'Time (sec)')
% ylabel(handles.axes2,'{E_u_l_d (MW)}')
```

```
% axis(handles.axes2,[0 length(E) min(E)-5 max(E)+5])
% if get(handles.grid_on_off_button,'Value')
%     grid(handles.axes2,'on')
% end


% --- Executes on button press in demand_power_apply_button.
function demand_power_apply_button_Callback(hObject, eventdata,
handles)
% hObject    handle to demand_power_apply_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global scomm method
dt = str2double(get(handles.edit9,'String'));
filepath = get(handles.edit1,'String');
if ~isempty(filepath)
    load(filepath) % loads variable 'demandCurve'
    Euld = [demandCurve(1,:) demandCurve(2,:)];
    table = get(handles.uitable1,'Data');
    [caip,caport] = agentLookup(table,'Control Agent',1,'isIA',1);
    %     [iaip,iaport] = agentLookup(table,'interfaceAgent',1);
    iaip = '129.62.151.131';iaport=10004;iasport = 10104;
    sender =
struct('ip',iaip,'rport',iaport,'task',struct('val','interfaceAgent','r
ank',1));
    receiver =
struct('ip',caip,'rport',caport,'task',struct('val','controlAgent','ran
k',1));
    content =
{'data','Euld',Euld,1,1,1/dt,str2num(get(handles.edit8,'String'))};
    if ~isempty(caip) && ~isempty(caport)
        %         msg = buildMessage('inform',sender,receiver,content);
        msg = encodeMASmsg('inform',sender,receiver,content);

sendMASmsg(msg,scomm,receiver.ip,receiver.rport,iasport,1,method)
        fprintf('\nSent message\n')
    end
end


function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as
a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in manual_power_apply_button.
function manual_power_apply_button_Callback(hObject, eventdata,
handles)
% hObject    handle to manual_power_apply_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global scomm method
dt = str2num(get(handles.edit9,'String'));
Euld = str2num(get(handles.edit2,'String'));
if ~isempty(Euld)
    table = get(handles.uitable1,'Data');
    [caip,caport] = agentLookup(table,'Control Agent',1,'isIA',1);
    %     [iaip,iaport] = agentLookup(table,'interfaceAgent',1);
    iaip = '129.62.151.131';iaport=10004;iasport = 10104;
    sender =
struct('ip',iaip,'rport',iaport,'task',struct('val','interfaceAgent','r
ank',1));
    receiver =
struct('ip',caip,'rport',caport,'task',struct('val','controlAgent','ran
k',1));
    content =
{'data','Euld',Euld,1,1,1/dt,str2num(get(handles.edit8,'String'))};
    if ~isempty(caip) && ~isempty(caport)
        %         msg = buildMessage('inform',sender,receiver,content);
        msg = encodeMASmsg('inform',sender,receiver,content);

sendMASmsg(msg,scomm,receiver.ip,receiver.rport,iasport,1,method)
        fprintf('\nSent message\n')
    end
end

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
```

```
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu2 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in task_change_apply_button.
function task_change_apply_button_Callback(hObject, eventdata, handles)
% hObject    handle to task_change_apply_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global scomm method
MASpath
opt1 = get(handles.popupmenu1,'Value')-1;
opt2 = get(handles.popupmenu2,'Value');
table = get(handles.uitable1,'Data');
oldTask = table(opt1,:);
newTask = get(handles.popupmenu2,'String');
newTask = cell2mat(newTask(opt2));
```

```
switch(newTask)
    case 'Free Agent'
        newTask = 'freeAgent';
    case 'Delegation Agent'
        newTask = 'delegationAgent';
    case 'Control Agent'
        newTask = 'controlAgent';
    case 'Feedback Agent'
        newTask = 'feedbackAgent';
    case 'Neural Network Agent'
        newTask = 'nnAgent';
    case 'Database Agent'
        newTask = 'databaseAgent';
    case 'Interface Agent'
        newTask = 'interfaceAgent';
    case 'Monitoring Agent'
        newTask = 'monitoringAgent';
end
[daip,daport] = agentLookup(table,'Delegation Agent',1,'isIA',1);
% [iaip,iaport] = agentLookup(table,'interfaceAgent',1);
iaip = '129.62.150.157';iaport=10004;iasport = 10104;
sender =
struct('ip',iaip,'rport',iaport,'task',struct('val','interfaceAgent','r
ank',1));
receiver =
struct('ip',daip,'rport',daport,'task',struct('val','delegationAgent','
rank',1));
content = {'taskChange',oldTask(1),oldTask(2),oldTask(3),1,newTask,1};
if ~isempty(daip) && ~isempty(daport)
    %     msg = buildMessage('request',sender,receiver,content);
    msg = encodeMASmsg('request',sender,receiver,content);
    sendMASmsg(msg,scomm,receiver.ip,receiver.rport,iasport,1,method)
    fprintf('\nMessage sent to DA\n')
end

% set(handles.uitable1,'Data',data)

% --- Executes on button press in preferences_apply_button.
function preferences_apply_button_Callback(hObject, eventdata, handles)
% hObject    handle to preferences_apply_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global scomm method
if ~isempty(get(handles.edit3,'String')) &&
~isempty(get(handles.edit4,'String')) &&
~isempty(get(handles.edit5,'String')) &&
~isempty(get(handles.edit6,'String'))
    beta(1) = str2num(get(handles.edit3,'String'));
    beta(2) = str2num(get(handles.edit4,'String'));
    beta(3) = str2num(get(handles.edit5,'String'));
    beta(4) = str2num(get(handles.edit6,'String'));
    set(handles.text42,'String',num2str(beta(1)));
    set(handles.text43,'String',num2str(beta(2)));
    set(handles.text44,'String',num2str(beta(3)));
    set(handles.text45,'String',num2str(beta(4)));
    table = get(handles.uitable1,'Data');
```

```
    [caip,caport] = agentLookup(table,'Control Agent',1,'isIA',1);
    %     [iaip,iaport] = agentLookup(table,'interfaceAgent',1);
    iaip = '129.62.150.157';iaport=10004;iasport = 10104;;
    sender =
struct('ip',iaip,'rport',iaport,'task',struct('val','interfaceAgent','r
ank',1));
    receiver =
struct('ip',caip,'rport',caport,'task',struct('val','controlAgent','ran
k',1));
    content = {'data','beta',beta};
    if ~isempty(caip) && ~isempty(caport)
        %         msg = buildMessage('inform',sender,receiver,content);
        msg = encodeMASmsg('inform',sender,receiver,content);

sendMASmsg(msg,scomm,receiver.ip,receiver.rport,iasport,1,method)
        fprintf('\nSent message\n')
    end
end

% --- Executes on button press in demand_power_browse_button.
function demand_power_browse_button_Callback(hObject, eventdata,
handles)
% hObject    handle to demand_power_browse_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
current_dir = cd;
try cd('Data\Demand Curves'); catch; end
[filename pathname] = uigetfile({'.mat'},'Select Demand Power Curve
File','demandCurve.mat');
filepath = [pathname filename];
dt = str2double(get(handles.edit9,'String'));
if filepath
    set(handles.edit1,'String',filepath)
    load(filepath) % loads variables 'demandCurve' and 'tstep'
    Euld = demandCurve; clear demandCurve
    for n=1:length(Euld)-1
        E(Euld(1,n)+1:Euld(1,n+1)) =
linspace(Euld(2,n),Euld(2,n+1),(Euld(1,n+1)-Euld(1,n)));
    end
    plot(handles.axes2,dt:dt:numel(E)*dt,E)
    xlabel(handles.axes2,'Time (sec)')
    ylabel(handles.axes2,'{E_u_l_d (W)}')
    axis(handles.axes2,[0 length(E)*dt min(E)-.5 max(E)+.5])
    if get(handles.grid_on_off_button,'Value')
        grid(handles.axes2,'on')
    end
else
    %Do nothing
end
cd(current_dir)

% --- Executes on button press in test_button.
function test_button_Callback(hObject, eventdata, handles)
% hObject    handle to test_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% data = {'Delegation Agent    ','129.62.150.128',4000; ...
%         'Control Agent       ','129.62.150.129',4000; ...
%         'Feedback Agent      ','129.62.150.181',4000; ...
%         'Neural Network Agent','129.62.150.182',4000; ...
%         'Monitoring Agent    ','129.62.150.191',4000; ...
%         'Database Agent      ','129.62.150.108',4000; ...
%         'Gain Optimizer Agent','129.62.150.110',4000; ...
%         'Free Agent          ','129.62.150.111',4000; ...
%         'Free Agent          ','129.62.150.112',4000; ...
%         'Free Agent          ','129.62.150.113',4000};
data = {'Delegation Agent'     ,'129.62.151.131',10000,10100;...
    'Control Agent'        ,'129.62.151.131',10001,10101;...
    'Feedback Agent'       ,'129.62.151.131',10002,10102;...
    'Interface Agent'      ,'129.62.151.131',10004,10104;...
    'Power Plant'          ,'129.62.68.239',43000,10150};
set(handles.uitable1,'Data',data)
tmp = get(handles.uitable1,'Data');
list{1} = '- Select Agent to change -';
for n=1:length(get(handles.uitable1,'Data'))
    if n < 10
        list{n+1} = ['0' num2str(n) ' ' tmp{n,1}];
    elseif n >= 10 && n < 100
        list{n+1} = [num2str(n) ' ' tmp{n,1}];
    end
end
set(handles.popupmenu1,'String',list)


% --- Executes on button press in grid_on_off_button.
function grid_on_off_button_Callback(hObject, eventdata, handles)
% hObject    handle to grid_on_off_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if get(handles.grid_on_off_button,'Value') == 1
    grid(handles.axes2,'on')
else
    grid(handles.axes2,'off')
end


% --- Executes on button press in start_IA_button.
function start_IA_button_Callback(hObject, eventdata, handles)
% hObject    handle to start_IA_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global running
%run ..\..\MASpath.m
disp = 100;
Edout=[zeros(1,disp)];
Eout =[zeros(1,disp)];

Pdout=[zeros(1,disp)];
Pout=[zeros(1,disp)];
running = 1;

t = 1:disp;
```

```
set(handles.uitable1,'Data',cell(1,3))
while running
    try
        if exist('toClient.mat','file') == 2
            load toClient
            Edout=[Edout(2:end) Ed];
            Eout=[Eout(2:end) E];
            Pdout=[Pdout(2:end) Pd];
            Pout=[Pout(2:end) P];
            %             delete('toClient.mat')
            plot(handles.axes1,t,Eout,'b',t,Edout,'r')
            axis([0 disp min(min(Eout,Edout))-.5
max(max(Eout,Edout))+.5])
            legend(handles.axes1,'E','Ed','Location','NorthWest')

            plot(handles.axes5,t,Pout,'b',t,Pdout,'r')
            axis([0 disp min(min(Pout,Pdout))-5
max(max(Pout,Pdout))+5])
            legend('P','Pd','Location','NorthWest')


            set(handles.text9,'String',num2str(Ed))
            set(handles.text10,'String',num2str(E))
            set(handles.text27,'String',num2str(Pd))
            set(handles.text28,'String',num2str(P))
            set(handles.text33,'String',num2str(0))
            set(handles.text34,'String',num2str(Loadm))
%             set(handles.uitable1,'Data',[]);
        end


        %% SET TEMPERARYILY TO HARD CODE ADDRESSES
%         table = {'Delegation Agent'     ,'129.62.150.157',4000;...
%             'Control Agent'        ,'129.62.150.157',4001;...
%             'Feedback Agent'       ,'129.62.150.157',4002;...
%             'Gain Optimizer Agent' ,'129.62.150.157',4003;...
%             'Neural Network Agent' ,'129.62.150.157',4004;...
%             'Interface Agent'      ,'129.62.150.157',4010;...
%             'Power Plant'          ,'129.62.150.158',5000};
%         set(handles.uitable1,'Data',table)
%         list{1} = '- Select Agent to change -';
%         for n=1:size(table,1)
%             if n < 10
%                 list{n+1} = ['0' num2str(n) ' ' table{n,1}];
%             elseif n >= 10 && n < 100
%                 list{n+1} = [num2str(n) ' ' table{n,1}];
%             end
%         end
%         set(handles.popupmenu1,'String',list)
        %         end
        pause(1)

    catch me
        display(me);
        if strfind(me.message,'Unable to read MAT-file')
            pause(0.5)
```

```
        end
    end

end


% --- Executes on button press in stop_IA_button.
function stop_IA_button_Callback(hObject, eventdata, handles)
% hObject    handle to stop_IA_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global running rcomm scomm
running = 0;
fclose(rcomm);
fclose(scomm);


function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as
a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as
a double


% --- Executes during object creation, after setting all properties.
```

```
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as
a double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as
a double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in gainoptimizingapply.
function gainoptimizingapply_Callback(hObject, eventdata, handles)
% hObject    handle to gainoptimizingapply (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global scomm method
optionnum=get(handles.gainoptimizingpopup,'Value');
option=get(handles.gainoptimizingpopup,'String');
option=cell2mat(option(optionnum));


switch(option)
    case 'ON'
        fbmethod='GOYes';
        fprintf('\nGain Optimizing ON\n')

    case 'OFF'
        fprintf('\nGain Optimizing OFF\n')
        fbmethod='GONo';

    otherwise
        fprintf('\nDefault: Gain Optimizing ON\n')
        fbmethod='GOYes';
end
table = get(handles.uitable1,'Data');
[caip,caport] = agentLookup(table,'Control Agent',1,'isIA',1);
%     [iaip,iaport] = agentLookup(table,'interfaceAgent',1);
iaip = '129.62.150.157';iaport=10004;iasport = 10104;
sender =
struct('ip',iaip,'rport',iaport,'task',struct('val','interfaceAgent','r
ank',1));
receiver =
struct('ip',caip,'rport',caport,'task',struct('val','controlAgent','ran
k',1));
content = {'data','method',fbmethod};
if ~isempty(caip) && ~isempty(caport)
    %     msg = buildMessage('inform',sender,receiver,content);
    msg = encodeMASmsg('inform',sender,receiver,content);
    sendMASmsg(msg,scomm,receiver.ip,receiver.rport,iasport,1,method)
    fprintf('\nSent message\n')
end

% --- Executes on selection change in gainoptimizingpopup.
```

```
function gainoptimizingpopup_Callback(hObject, eventdata, handles)
% hObject    handle to gainoptimizingpopup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns gainoptimizingpopup
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
gainoptimizingpopup


% --- Executes during object creation, after setting all properties.
function gainoptimizingpopup_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gainoptimizingpopup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in feedbackapply.
function feedbackapply_Callback(hObject, eventdata, handles)
% hObject    handle to feedbackapply (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global scomm method
optionnum=get(handles.feedbackpopup,'Value');
option=get(handles.feedbackpopup,'String');
option=cell2mat(option(optionnum));

switch(option)
    case 'ON'
        fprintf('\nFeedback ON\n')
        fbmethod='FBYes';

    case 'OFF'
        fprintf('\nFeedback OFF\n')
        fbmethod='FBNo';

    otherwise
        fprintf('\nDefault: Feedback ON\n')
        fbmethod='FBYes';

end
table = get(handles.uitable1,'Data');
[caip,caport] = agentLookup(table,'Control Agent',1,'isIA',1);
%     [iaip,iaport] = agentLookup(table,'interfaceAgent',1);
iaip = '129.62.150.157';iaport=10004;iasport = 10104;
```

```
sender =
struct('ip',iaip,'rport',iaport,'task',struct('val','interfaceAgent','r
ank',1));
receiver =
struct('ip',caip,'rport',caport,'task',struct('val','controlAgent','ran
k',1));
content = {'data','method',fbmethod};
if ~isempty(caip) && ~isempty(caport)
    %     msg = buildMessage('inform',sender,receiver,content);
    msg = encodeMASmsg('inform',sender,receiver,content);
    sendMASmsg(msg,scomm,receiver.ip,receiver.rport,iasport,1,method)
    fprintf('\nSent message\n')
end


% --- Executes on selection change in feedbackpopup.
function feedbackpopup_Callback(hObject, eventdata, handles)
% hObject    handle to feedbackpopup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns feedbackpopup
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
feedbackpopup


% --- Executes during object creation, after setting all properties.
function feedbackpopup_CreateFcn(hObject, eventdata, handles)
% hObject    handle to feedbackpopup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as
a double


% --- Executes during object creation, after setting all properties.
```

```
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as
a double


% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function gainA_Callback(hObject, eventdata, handles)
% hObject    handle to gainA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gainA as text
```

```
%        str2double(get(hObject,'String')) returns contents of gainA as
a double


% --- Executes during object creation, after setting all properties.
function gainA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gainA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function gainC_Callback(hObject, eventdata, handles)
% hObject    handle to gainC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gainC as text
%        str2double(get(hObject,'String')) returns contents of gainC as
a double


% --- Executes during object creation, after setting all properties.
function gainC_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gainC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function gainG_Callback(hObject, eventdata, handles)
% hObject    handle to gainG (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gainG as text
%        str2double(get(hObject,'String')) returns contents of gainG as
a double
```

```
% --- Executes during object creation, after setting all properties.
function gainG_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gainG (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function gainB_Callback(hObject, eventdata, handles)
% hObject    handle to gainB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gainB as text
%        str2double(get(hObject,'String')) returns contents of gainB as
a double


% --- Executes during object creation, after setting all properties.
function gainB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gainB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function gainD_Callback(hObject, eventdata, handles)
% hObject    handle to gainD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gainD as text
%        str2double(get(hObject,'String')) returns contents of gainD as
a double
```

```
% --- Executes during object creation, after setting all properties.
function gainD_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gainD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function gainE_Callback(hObject, eventdata, handles)
% hObject    handle to gainE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gainE as text
%        str2double(get(hObject,'String')) returns contents of gainE as
a double


% --- Executes during object creation, after setting all properties.
function gainE_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gainE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function gainF_Callback(hObject, eventdata, handles)
% hObject    handle to gainF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gainF as text
%        str2double(get(hObject,'String')) returns contents of gainF as
a double


% --- Executes during object creation, after setting all properties.
function gainF_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to gainF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function gainH_Callback(hObject, eventdata, handles)
% hObject    handle to gainH (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gainH as text
%        str2double(get(hObject,'String')) returns contents of gainH as
a double


% --- Executes during object creation, after setting all properties.
function gainH_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gainH (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit18_Callback(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit18 as text
%        str2double(get(hObject,'String')) returns contents of edit18
as a double


% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in applyGainValues.
function applyGainValues_Callback(hObject, eventdata, handles)
% hObject    handle to applyGainValues (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global scomm method
Kp1 = str2double(get(handles.gainA,'String'));
Kp2 = str2double(get(handles.gainB,'String'));
Kp3 = str2double(get(handles.gainC,'String'));
Kp4 = str2double(get(handles.gainD,'String'));
Ki1 = str2double(get(handles.gainE,'String'));
Ki2 = str2double(get(handles.gainF,'String'));
Ki3 = str2double(get(handles.gainG,'String'));
Ki4 = str2double(get(handles.gainH,'String'));

Kd1 = str2double(get(handles.Kd1,'String'));
Kd2 = str2double(get(handles.Kd2,'String'));
Kd3 = str2double(get(handles.Kd3,'String'));
Kd4 = str2double(get(handles.Kd4,'String'));



table = get(handles.uitable1,'Data');
for n=1:size(table,1)
    agentTable{n}.ip = table{n,2};
    agentTable{n}.rport = table{n,3};
    switch(table{n,1})
        case 'Delegation Agent',  agentTable{n}.task.val =
'delegationAgent';
        case 'Control Agent',  agentTable{n}.task.val = 'controlAgent';
        case 'Feedback Agent',  agentTable{n}.task.val =
'feedbackAgent';
        case 'Gain Optimizer Agent',  agentTable{n}.task.val =
'gainoptimizerAgent';
        case 'Interface Agent',  agentTable{n}.task.val =
'interfaceAgent';
        case 'Power Plant',  agentTable{n}.task.val = 'PowerPlant';
    end
    agentTable{n}.task.rank = 1;
end

[fbip,fbport]=agentLookup(agentTable','feedbackAgent',1,'isIA',0);
% [fbip,fbport]=agentLookup(agentTable,'feedbackAgent',1);
iaip = '129.62.151.131';iaport=10004;iasport = 10104;
```

```
sender =
struct('ip',iaip,'rport',iaport,'task',struct('val','interfaceAgent','r
ank',1));
receiver =
struct('ip',fbip,'rport',fbport,'task',struct('val','feedbackAgent','ra
nk',1));
content = {'GO_data',[Kp1 Kp2 Kp3 Kp4],[Ki1 Ki2 Ki3 Ki4],[Kd1 Kd2 Kd3
Kd4]};
msg = encodeMASmsg('inform',sender,receiver,content);
sendMASmsg(msg,scomm,receiver.ip,receiver.rport,iasport,1,method)
fprintf('\nFeedback gains sent to feedback agent\n')


% --- Executes on button press in clearGainValues.
function clearGainValues_Callback(hObject, eventdata, handles)
% hObject    handle to clearGainValues (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.gainA,'String','0');
set(handles.gainB,'String','0');
set(handles.gainC,'String','0');
set(handles.gainD,'String','0');
set(handles.gainE,'String','0');
set(handles.gainF,'String','0');
set(handles.gainG,'String','0');
set(handles.gainH,'String','0');

set(handles.Kd1,'String','0');
set(handles.Kd2,'String','0');
set(handles.Kd3,'String','0');
set(handles.Kd4,'String','0');



% --- Executes on button press in getGainValues.
function getGainValues_Callback(hObject, eventdata, handles)
% hObject    handle to getGainValues (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% [fbip,fbport] = agentLookup(table,'Feedback Agent',1,'isIA',1);
% sender =
struct('ip',iaip,'rport',iaport,'task',struct('val','interfaceAgent','r
ank',1));
% receiver =
struct('ip',fbip,'rport',fbport,'task',struct('val','feedbackAgent','ra
nk',1));
% content = {'request','data','gains'};
% if ~isempty(caip) && ~isempty(caport)
%     msg = encodeMASmsg('request',sender,receiver,content);
%     sendMASmsg(msg,scomm,receiver.ip,receiver.rport,iasport,1,method)
%     fprintf('\nSent message\n')
% end
load toClient.mat

% msg_rec = 0;
```

```
% while msg_rec == 0
%     try
%         msg = receiveMASmsg(rcomm,method,1); msg_rec = 1;
%         msg = decodeMASmsg(msg);
%         K = msg.content{4};
%         Kp = K(:,:,1);
%         Ki = K(:,:,2);
%         Kd = K(:,:,3);
        set(handles.gainA,'String',num2str(Kp(1,1)));
        set(handles.gainB,'String',num2str(Kp(1,2)));
        set(handles.gainC,'String',num2str(Kp(2,1)));
        set(handles.gainD,'String',num2str(Kp(2,2)));

        set(handles.gainF,'String',num2str(Ki(1,2)));
        set(handles.gainH,'String',num2str(Ki(2,2)));
        set(handles.gainG,'String',num2str(Ki(2,1)));
        set(handles.gainE,'String',num2str(Ki(1,1)));

        set(handles.Kd1,'String',num2str(Kd(1,1)));
        set(handles.Kd2,'String',num2str(Kd(1,2)));
        set(handles.Kd3,'String',num2str(Kd(2,1)));
        set(handles.Kd4,'String',num2str(Kd(2,2)));

%     catch
%     end


% --- Executes during object creation, after setting all properties.
function text5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called



function kd1_Callback(hObject, eventdata, handles)
% hObject    handle to kd1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of kd1 as text
%        str2double(get(hObject,'String')) returns contents of kd1 as a
double


% --- Executes during object creation, after setting all properties.
function kd1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to kd1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function kd2_Callback(hObject, eventdata, handles)
% hObject    handle to kd2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of kd2 as text
%        str2double(get(hObject,'String')) returns contents of kd2 as a
double


% --- Executes during object creation, after setting all properties.
function kd2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to kd2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function kd3_Callback(hObject, eventdata, handles)
% hObject    handle to kd3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of kd3 as text
%        str2double(get(hObject,'String')) returns contents of kd3 as a
double


% --- Executes during object creation, after setting all properties.
function kd3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to kd3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function kd4_Callback(hObject, eventdata, handles)
% hObject    handle to kd4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of kd4 as text
%        str2double(get(hObject,'String')) returns contents of kd4 as a
double


% --- Executes during object creation, after setting all properties.
function kd4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to kd4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

C.13.1 agentLookup.m

```
function [ip,port] = agentLookup(table,agent,rank,varargin)

% agentLookup(field,search)
%
%  This function is used to find data in the agentTable. It returns all
%  data pertaining to the item searched for.
%
%  table - contains the agentTable structure
%  agent - describes which agent't information is being searched for
%

isIA = 0;
for n=1:2:nargin-3
    switch varargin{n}
        case 'isIA'
            isIA = varargin{n+1};
    end
end

if isIA
    ip = []; port = [];
    for n=1:size(table,1)
        if exist('table','var') == 1
            if strcmp(table{n,1},agent) == 1
                ip = table{n,2};
                port = table{n,3};
            end
        end
    end
elseif ~isIA
    ip = []; port = [];
    for n=1:size(table,1)
        if exist('table','var') == 1
            if strcmp(table{n}.task.val,agent) == 1
                ip = table{n}.ip;
                port = table{n}.rport;
                break
            end
        end
    end
end
```

APPENDIX D

Operation Instructions

D.1 *RankineCycler* Warmup

1. Make sure the *RC* is connected to power, and to the laptop through both USB cables.

2. Turn on the *RC* master switch and the laptop. The laptop username is “Turbine Technologies” pw: “Turbine2!”. After the laptop boots load the Acquisition and Control VI. The AC, Gauges, and Motor control VIs should appear after loading.

3. Flip the switch on the right of the burner switch to “Man”. This should de-energize the motors. Make sure all four motors are in their closed position. This means full counter-clockwise rotation for the load motor knob, full clockwise for the rest. Do not close the fuel valve too tightly. Flip the control switch back to computer control. You should hear the motors re-engage.

4. Make sure the AC VI has “com 5” selected in the upper right hand side. (note: it may not be com 5, pick whichever one is NOT com1)

5. Run the AC VI. A popup should appear referencing DAQ calibration. If it does not appear, the DAQ needs to be reset, unplug it and reconnect. The DAQ is connected to the uppermost USB slot on the RC

6. Once the AC Vi is running, in the MC VI make sure the “Air auto-track” button is set to false. Click “Initialize” on the MC front panel. The air and fuel motors should move to starting positions. Open the propane tank and turn the fuel regulator to the “on” position.

7. Turn the burner switch on. The blower fan should engage. After 45s the burner will attempt to ignite. If you hear a slight clicking noise and it does not light, that means the blower setting is probably too high. Lower the air motor position in the MC VI and try again. (13-18 steps usually works).

8. Once the burner ignites, increase the air motor position to the “Expected Air” value gradually over about 30 seconds. Then set “Air Auto Track” to true.

9. You will now have to wait as the boiler warms up. Keep an eye on the Pressure inside the boiler. When it reaches about 100 psi, open the steam valve by 20 steps. This should create a large pressure drop and steam will begin to flow.

10. You will have to wait some more as the Turbine heats up. The target temperate is about 215º F on both the inlet and the outlet. Keep opening the steam valve very slowly and maintaining boiler pressure around 80psi until the system warms up. During this time the turbine-generator assembly will leak underneath the joint between the two. It is highly recommended you mop up this excess water. Be careful not to touch the pipes, however.

11. When boiler temp reaches about 300º and turbine temp above 200º, open the steam valve to 70. Put on the oven mitt and flick the generator spinner toward you. It should start up and spin on its own. If it doesn't, close the steam valve to get more pressure and try again after the *RC* heats up more.

12. Once the generator is spinning the *RC* is producing power. While in operation, don't let the pressure exceed 115psi, the temperature exceed 550º, or the voltage 15V.

13. In case of emergency: burner switch off, steam valve closed, Load at 100%. Close the Propane regulator after you change the others.

D.2 MAS Startup

1. Open MATLAB. Change the director to the base folder with all the MAS files and run "maspath.m" in the utility folder. Return the main director to the agents folder.

2. Open the parallel command window by typing "pmode start local 6". You may have to modify the pmode config file to allow 6.

3. Make sure the IP and port configurations are correct to reach the laptop. The settings to modify are in the feedback agent m-file. The AC should have the port and ip of the feedback agent. Default: 10002 and 129.62.151.131. The MC VI should have the correct receiving port open. Default: 42001.

4. In the pmode window run "masagent". After a couple seconds the agents should boot up in standby mode.

5. In the normal command window, run the interfaceAgentGui. The GUI should appear.

6. Click "Start interface agent" to begin. Click "generate agent table" to find the other agents.

7. Now browse for your demand curve and click "apply" to start the controller. Be careful starting the system at a power level too different from the *RC* current output.

REFERENCES

[1] U.S. Energy Information Administration, "US Generating Units," Available: http://www.eia.doe.gov/cneaf/electricity/epa/epates.html. [Accessed: December 2011]

[2] Ned Mohan, "Synchronous Generators", in *First Course on Power Systems*, 2006 ed. Minneapolis, MN: MNPERE, 2006, ch. 9

[3] Raul Garduno-Ramirez, "Overall Intelligent Hybrid Control System for a Fossil-fuel Power Unit," Ph. D dissertation, Dept. of Electrical Eng., Pennsylvania State Univ., 2000.

[4] Jin S. Heo, "A Multi-Agent System-based Intelligent Control for Large-scale Power Plants," Ph. D dissertation, Dept. of Electrical Eng., Pennsylvania State Univ., 2006.

[5] Turbine Technologies, Ltd., "Rankine Cycler Procedure," in *Steam Turbine Power System Operator's Manual*, rev. 9.05.

[6] Turbine Technologies, Ltd., "Steam Generation Boiler," in *RankineCycler Steam Turbine Power System Sample Lab Experiment Procedure*, 2011.

[7] Turbine Technologies, Ltd., "Steam Turbine CAD Cutaway," in *RankineCycler Steam Turbine Power System Sample Lab Experiment Procedure*, 2011.

[8] Probotix HT23-280-3 Datasheet. Probotix. [Online]. Available: https://probotix.com/manuals/ProboStepVX.pdf

[9] Heo and Lee, "A Mulitagent-System-Based Intelligent Reference Governor for Multobjective Optimal Power Plant Operation," in *IEEE Transactions on Energy Conversion*, *Vol.23, No. 4*, December 2008. pp 1082-1092.

[10] Jason D. Head et al, "Implementation of a Multi-Agent System for Optimized Multiobjective Power Plant Control," *North American Power Symposium*, 2010.

[11] The Foundation for Intelligent Physical Agents, "FIPA ACL Message Structure Specification," *The Foundation for Intelligent Physical Agents*, 2001. [Online]. Available: http://www.fipa.org/specs/fipa00061. [Accessed: May 17, 2010].

[12] Kennedy, J.; Eberhart, R. "Particle Swarm Optimization". *Proceedings of IEEE International Conference on Neural Networks*. 1995. pp. 1942–1948.

[13] Jin S. Heo, Kwang Y. Lee, Raul Garduno-Ramirez, "Multiobjective Control of Power Plants Using Particle Swarm Optimization Techniques," *IEEE Transactions on Energy Converstion, Vol. 21, No. 2*, 2006. pp. 552-561.

[14] IEEE RTS Task Force of APM Subcommittee, "The IEEE Reliability Test System", *IEEE Transaction on Power Systems, Vol. 13, No.3,* 1999.

[15] MATLAB Help Files, "Layer-Recurrent Network," *MATLAB R2009a*, The Mathworks, Inc., Cambridge MA. 2008.

[16] MATLAB Help Files, "Feedforward Network," *MATLAB R2009a*, The Mathworks, Inc., Cambridge MA. 2008.

[17] Jason D. Head, "Development and Implementation of a Multi-Agent System for Intelligent Optimal Power Plant Control," M.S. thesis, Dept. of Electrical and Computer Eng., Baylor Univ.,Waco, TX, 2012.

[18] K.J. Astrom and R.D. Bell, "Drum-boiler dynamics," presented at IFAC 13th World Congress, San Francisco, CA, 1996.

[19] Farah Hani Nordin and Farrukh Hafiz Nagi, "Layer-Recurrent Network in Identifying a Nonlinear System," presented at International Conference on Control, Automation and Systems, Seoul, Korea, 2008.

[20] Stephen D. J. McArthur et al., "Multi-Agent Systems for Power Engineering Applications—Part I: Concepts, Approaches, and Technical Challenges," *IEEE Transactions on Power Systems, Vol. 22, No. 4*, 2007. pp. 1723-1752.

[21] Stephen D. J. McArthur et al., " Multi-Agent Systems for Power Engineering Applications—Part II: Technologies, Standards, and Tools for Building Multi-agent Systems," *IEEE Transactions on Power Systems, Vol. 22, No. 4*, 2007. pp. 1753-1759.

[22] Craig S. Williams, Jr., "Design and Implementation of a Multi-Agent Optimized Control System for a Large-Scale Fossil-Fuel Electrical Power Unit," M.S. thesis, Dept. of Electrical and Computer Eng., Baylor Univ.,Waco, TX, 2011.