# CONTROL AND AUTOMATION OF A HEAT SHRINK TUBING PROCESS

Shahrokh Yousefi Darani

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2014

APPROVED:

Enrique Barbieri, Major Professor and Chair of the Department of Engineering Technology
Robert G. Hayes, Committee Member
Huseyin Bostanci, Committee Member
Costas Tsatsoulis, Dean of the College of Engineering
Mark Wardell, Dean of the Toulouse Graduate School

Yousefi Darani, Shahrokh. Control and Automation of a Heat Shrink Tubing Process. Master of Science (Engineering Systems-Electrical Systems), August 2014, 59 pp., 2 tables, 25 figures, references, 26 titles.

Heat shrink tubing is used to insulate wire conductors, protect wires, and to create cable entry seals in wire harnessing industries. Performing this sensitive process manually is time consuming, the results are strongly dependent on the operator's expertise, and the process presents safety concerns. Alternatively, automating the process minimizes the operators' direct interaction, decreases the production cost over the long term, and improves quantitative and qualitative production indicators dramatically. This thesis introduces the automation of a heat shrink tubing prototype machine that benefits the wire harnessing industry. The prototype consists of an instrumented heat chamber on a linear positioning system, and is fitted with two heat guns. The chamber design allows for the directing of hot air from the heat guns onto the wire harness uniformly through radially-distributed channels. The linear positioning system is designed to move the heat chamber along the wire harness as the proper shrinkage temperature level is reached. Heat exposure time as a major factor in the heat shrink tubing process can be governed by controlling the linear speed of the heat chamber. A control unit manages the actuator position continuously by measuring the chamber's speed and temperature. A model-based design approach is followed to design and test the controller, and MATLAB/Simulink is used as the simulation environment. A programmable logic controller is selected as the controller implementation platform. The control unit performance is examined and its responses follow the simulation results with adequate accuracy.

## ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Enrique Barbieri, for his support and guidance during my study at UNT. I also acknowledge my appreciation to thesis committee members, Dr. Robert G. Hayes and Dr. Huseyin Bostanci, for their feedback.

Particular thanks goes to my beloved parents, and my brothers for being a constant source of moral support and guidance in my whole life.

Zohreh, your friendship and love have given me a new meaning of life. You stood beside me and encouraged me constantly during these years. I dedicate this work to you.

Lastly, I thank God for giving me the opportunity of coming to this country and completing my study at UNT. I present my regards to all friends and nice people I have met in this country.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

The aim of this thesis is to provide a practical solution for a problem of direct industrial relevance. For this goal potential resources in academia are utilized to direct us in addressing the existing challenge in the real world. Although wire harnessing industry is targeted for achievements of this project, the results could be expanded for a variety of other control and automation applications in industrial environments. Moreover, the research will provide a solid framework to be used for any other similar studies.

## 1.1. Wire Harnessing Process

The process of interest in this thesis is wire harnessing, assembling of wires that are bound together to transmit signals or electric power. The wires are typically bound by straps, cable lacing, sleeves, electrical tape, or a weave of extruded string. Wire harnesses provide several benefits over loose wires. Many aircraft, automobiles and spacecraft contain huge bundles of wires that would stretch over several kilometers if fully extended. By putting this multitude of wires into a wire harness shape, they can be better secured against the adverse effects of vibrations, abrasions, and moisture. Moreover, usage of space will be optimized, and the risk of an electrical short will be decreased. Since the installer has only one harness to install, installation time is decreased and the process can be easily standardized [1].

The wire harnessing process consists of several steps needed to produce wire harnesses. The process involves engineering design and development efforts, labor work, and machinery operation. Wire harnesses are usually designed according to geometric and electrical requirements so the process sequences and its specifications are determined to suit a particular application.

## 1.2. Heat Shrink Tubing in Wire Harnessing Process

A heat shrink tube is ordinarily made of nylon or polyolefin, which has the capability of shrinking about its diameter axis when heated. This feature helps the tube to be utilized in a wide range of applications, from near microscopically-thin-wall tubing to rigid, heavy-wall tubing. Among all existing applications, heat shrink tubes could be used to insulate wire conductors in wire harnessing processes. Heat shrink tubes can also be used to repair the insulation on wires or to bundle them together, to protect wires, and to create cable entry seals. Besides serving as an electrical insulator, the heat shrink tube provides environmental protection against dust, solvents, and other foreign materials, and is mechanically held in place by its tight fit [2].

## 1.3. Motivations

To perform heat shrinking of the tubes in a wire harness process, the unshrunk tube is fitted on the wire before making the connection. The tubing is then shrunk to wrap tightly around the joint by heating with a hot air gun or other heating sources. Uncontrolled heating can cause uneven shrinkage, physical damage, and insulation failure. If overheated, heat shrink tubing can melt, scorch, or catch fire like any other plastic [3]. Thus, the heating process should be controlled precisely to result in the desired tube profile.

All the above mentioned reasons make the heat shrink tubing process very tricky and sensitive. That’s why very skillful operators are needed to do the job, but performing this kind of sensitive process manually is time consuming and risky. The production performance is strongly dependent on the operators' expertise, and as operators interact with the process directly, this procedure is insufficiently safe. Therefore, figuring out a way to control the process automatically will minimize the operators' direct interaction, decline the production cost over the long term, and improve quantitative and qualitative production indexes dramatically.

Automatic control or automation is the use of various control systems for operating equipment such as machinery, processes in factories, and other applications with minimal human intervention. Some processes could be completely automated while the others might be semi-automated. The most important benefit of automation is that it saves labor; however, it is also used to save energy and used materials and to improve quality, accuracy and precision. Automation is achieved by various means including mechanical, hydraulic, pneumatic, electrical, electronic, and computers, usually in combination [19].

## 1.4. Existing Solutions in the Market

Although the heat shrink tubing process is done manually in many industries, there are some manufacturers in the market that provide automated solutions for this application. For some products, the work pieces are loaded into the machine manually, and passed through a heating tunnel automatically. In Figure 1.1, for instance, the machine developed by SLE

Electronics USA Inc.[1] has a belt conveyor that carries the wires with the sleeves inside an infrared oven for the shrinking process. Next, the work pieces are passed to a cooling zone and are deposited in a collection bin. This feature makes this solution ideal for large production requirements, because the throughput is determined by the feeding rate at which the operator loads the assemblies. These kinds of machines also can address the precise shrinkage specifications due to automatic movement of work pieces, but they could not process long wire harnesses as the heating tunnel dimensions are limited.

Figure 1.2 exhibits another heating tool named Triple Element Bench Glo-Ring® developed by Eraser[2] and could be used for tubing purposes. This product incorporates quartz encapsulated heating elements to radiate heat at temperatures up to 1500°F (815°C). The Glo-Ring could serve as an alternative to heat guns; however, this equipment does not solve the problem associated with heat guns in manual processes. In other words, if a long wire harness needs to be heated, the operator has to move the work piece manually. This results in the same problem experienced when using a heat gun manually and the only change is the heating source and of course moving the work piece instead of the heating source.

[1] www.sle-usa.com
[2] www.eraser.com

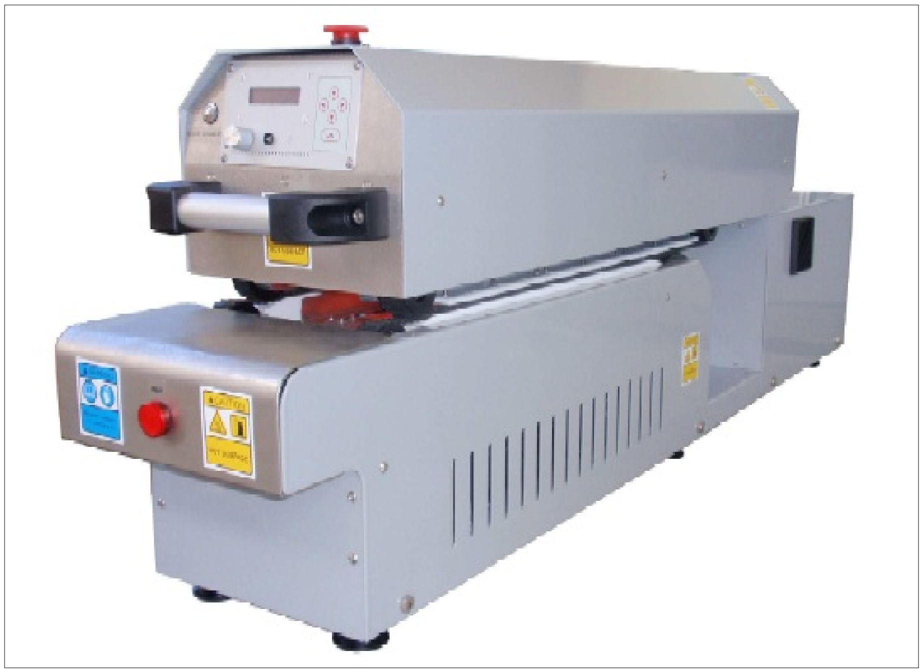


Figure 1.1: Shrinking Tube Conveyor System
(Picture captured from www.sle-usa.com)

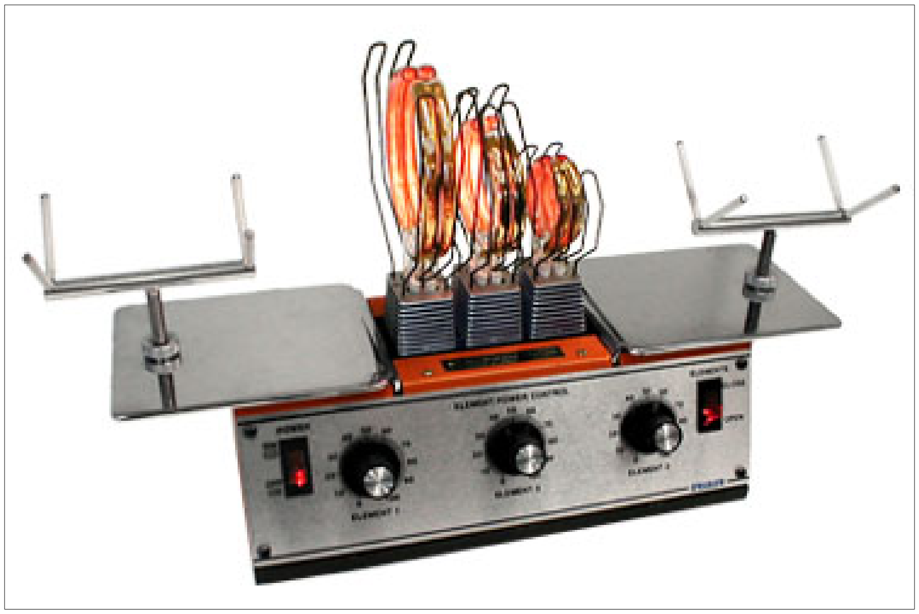


Figure 1.2: Triple Element Bench Glo-Ring® Heating Tool
(Picture captured from www.eraser.com)

## 1.5. Planned Objectives and Scope Definition

Based on the discussion presented above, a gap exists between the wire harnessing industry’s needs concerning the heat shrink tubing process, and existing solutions in the market. So, design and development of an automatic heat shrink tubing machine that benefits the wire harnessing industry is the ultimate purpose of a parent project, which breaks down into sub-projects including the current work. Similar to any new product development project, identifying user needs, interpreting them to the tangible engineering parameters, and performing a conceptual design based on cost and manufacturing constraints are the step-by-step phases that need to be carried out.

A small-scale prototype is developed first for preliminary testing and to direct us to the final product (Fig. 1.3). This prototype consists of an instrumented heat chamber (Fig. 1.4) on a linear positioning system, and fitted with one/two heat gun(s). The chamber design allows for the directing of hot air from the heat gun onto the wire harness uniformly through radially-distributed channels. Figure 1.5 illustrates the heat chamber parts in detail.

The current research proposes a practical controller unit for the mentioned heat shrink tubing machine. Taking advantage of recent achievements in the area of industrial automation, along with fundamental control theory, will be the bases of this study. As stated earlier, this research could be generalized to formulate and solve a wide range of any other control and automation problems. The aim of this study specifically is to design, simulate, and develop a control unit for the named heat shrink tubing machine to manage the heat exposure automatically. The control part of the work will collaborate with other sub-projects within the parent project. Therefore, all mechanical and manufacturing engineering parts are excluded from this work and this study will only look at the control and automation aspects of the machine.

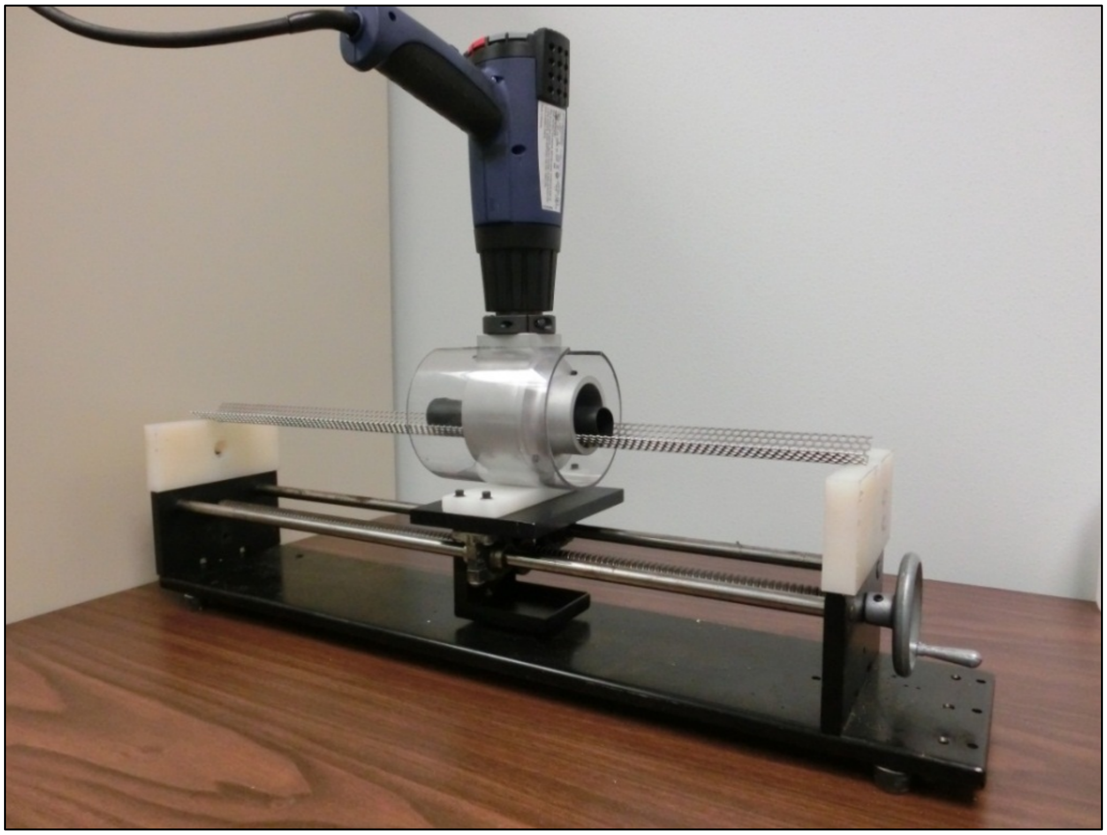


Figure 1.3: Prototype Heat Shrink Tubing Machine (Single Gun)

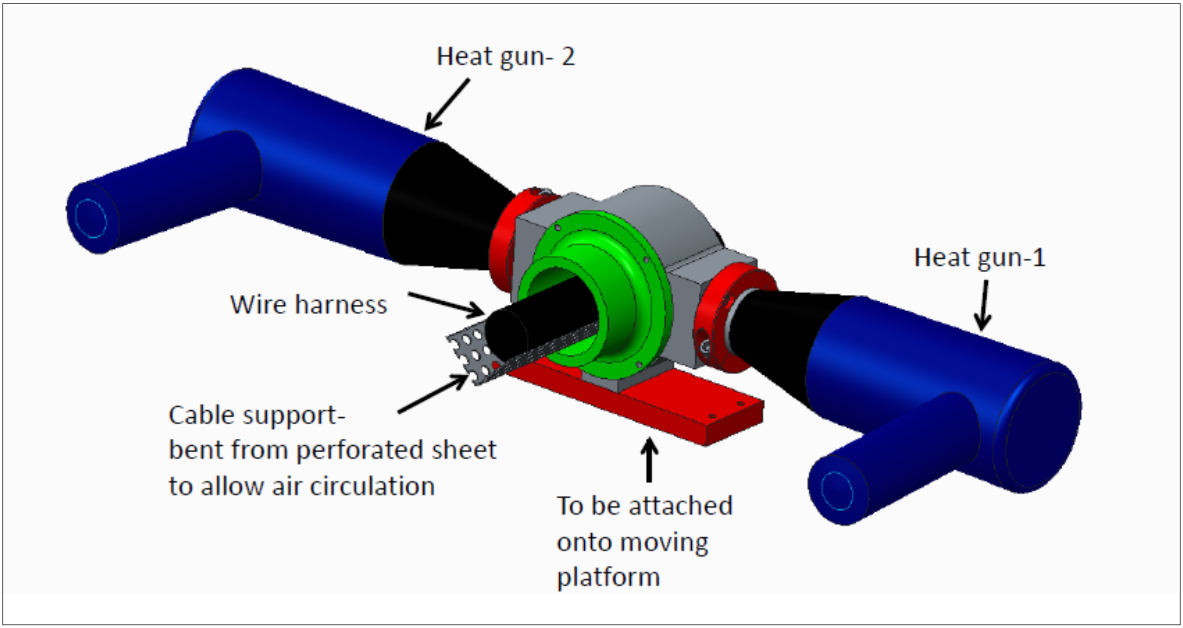


Figure 1.4: Heat Chamber Design (Double Gun)

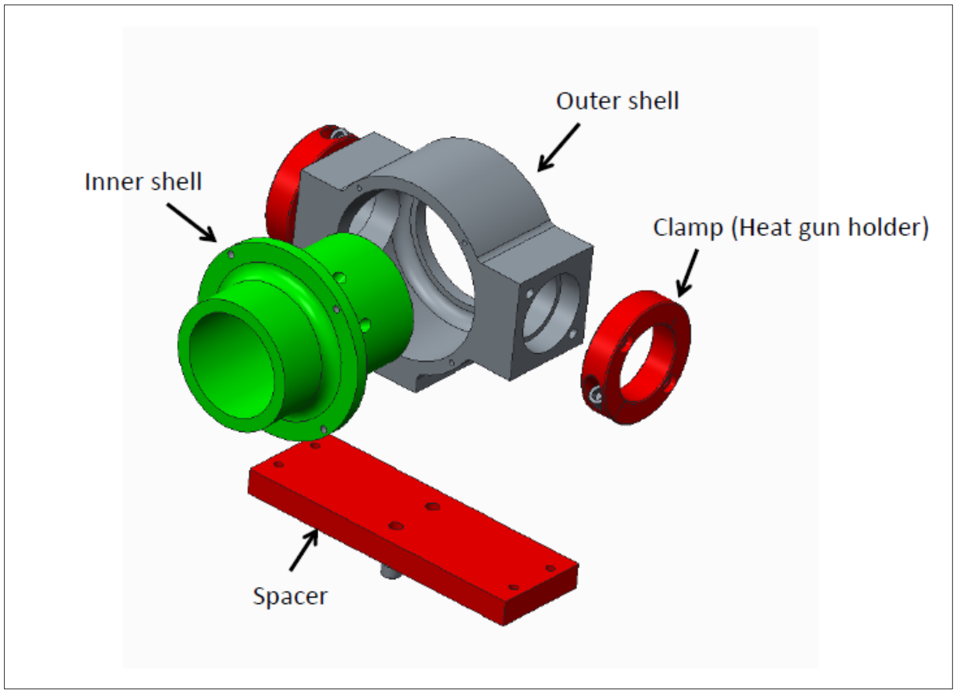


Figure 1.5: Heat Chamber Exploded View (Double Gun)

## 1.6. Proposed Solution

Heat exposure time is a major factor that can affect the shrinking process. Thus, with regulating that, one can control the shrinking process. In the new heat shrink tubing machine, when the heat guns (installed on the heat chamber) move over the cable's tube, with controlling the linear speed of the chamber, the heat exposure time can be adjusted and consequently the desired tube shrinking could be achieved.

For the above mentioned goal, a linear positioning system is designed to move the heat chamber along the wire harness as the proper shrinkage temperature level is reached. The other requisite is a controller unit to manage the actuator position and speed; nowadays, building control units with the help of embedded controller devices is common. To develop such a control unit, alongside the selection and configuration of an appropriate hardware, the needed control algorithm should be developed and implemented in a software environment. Then the produced software will be deployed into the hardware platform.

Based on the controlled process, plant environment characteristics, and used control philosophy, different platforms might be used for different applications. Programmable logic controller (PLC) is selected to be used for this application (versus other kinds of embedded controllers such as ASIC, FPGA, DSP). The main reason for this selection is that a heat shrink tubing machine will work in an industrial rough environment, and in these kinds of situations, PLCs are the best choices. The other reason is that since a PLC is an integrated control system (not a solitary chip), it will provide isolation, signal conditioning, and current/voltage amplification needed for interfacing with sensor and actuator layers. Eventually, PLCs will communicate easily with other devices like HMI panels and personal computers via predefined standard protocols.

## 1.7. Methodology: Model-Based Control Design

Because the current work is developing concurrently with the bigger parent project (development of a new heat shrink tubing machine), many parameters and factors are subjected to change which might delay development of the control unit. In contrary, with the traditional method of product development (all immediate preceding activities must be complete before the next phase) the following activities can begin sooner and not delay the work (laddering approach). So, modeling the system could be started considerably before completion of the machine development. The undefined parameters of the system will be estimated and as the project progresses, these approximations will be tuned progressively. In this way it would be needless to design the whole controller unit, by first waiting for the fulfillment of the machine development phase. This approach will save the development time significantly.

Model-based design is a generic method to address a defined problem associated with designing control systems. Model-based design is a methodology that provides an efficient approach for establishing a common framework for communication throughout the design process. In model-based design approach, development of a control system includes the following steps: modeling the plant (system to be controlled), designing a controller for the plant, simulating the plant and the controller and evaluating the controller performance, and finally, implementing the controller and integrating it with the real plant. The model-based design paradigm is different from traditional design methodology. Here, instead of using complex structures, designers can define models with advanced functional characteristics using prefabricated building blocks. These models along with simulation engines can lead to rapid prototyping, software-in-the-loop testing, and model and controller verification. In some cases, hardware-in-the-loop test also can be used to check the performance of the real controller with simulated plant [4].

As discussed earlier, the PLC is selected for controller implementation purposes. These days, when controller implementation transpires at the industry level, PLC programming is performed by expert programmers using one of the IEC-61131-3 standard languages. These programmers rarely come equipped with knowledge about modern software design methods. In other words, PLC logic is still implemented by conventional trial-and-error practices. On the other side, modern software design concepts have been considerably developed in recent years, thanks to the object-oriented methods, and this may lead to novel approaches in logic code design and generation for PLCs [8]. Moreover, fast changes in customer requirements during control and automation projects demand high levels of flexibility in control systems. To address these rapid changes in control philosophy, it is required that control logic code be generated automatically from the design stage outputs [9]. During the PLC code generation phase in the current work, the goal is to automatically generate some portion of the controller code using the provided utility in MATLAB®/Simulink®[1].

[1] MathWorks Inc., www.mathworks.com

## 1.8. Contents of the Following Chapters

After a short review on different types of system modeling, in Chapter 2, a physical model based on MATLAB/Simulink is presented for the heat shrink tubing machine. Simulation models mostly are developed for a specific range of applications because it is costly and time consuming to develop a valid model for a wide range of applications. That is why the developed model only looks at the electro-mechanical sections of the whole system and thermal behavior has not been modeled. Next, in the same chapter, the control challenges are introduced and an appropriate control system is proposed. A series of simulations are performed to ensure the proposed controller performance follows desired specifications. Chapter 3 includes the step-by-step phases for implementation of the designed controller, and covers both hardware and software aspects of the work. Finally, a test bench is suggested and the implemented controller performance is examined. A summary of the complete work is reviewed in the conclusion section in Chapter 4. Additionally, some potential ideas as follow-up works are presented in the future work section.

# CHAPTER 2

# SYSTEM MODELING AND CONTROL DESIGN

With regard to the model-based design approach, the development of a controller includes modeling the system to be controlled, designing an appropriate controller, simulating and evaluating the controller performance, and finally, implementing the controller and integrating it with the real plant. This chapter covers the first three steps, and the last step, controller implementation, will be covered in the next chapter.

## 2.1. Different Types of System Modeling

Modeling is the process to come up with a model to reproduce the system behavior, based on some knowledge of the original system [11]. Modeling methods are divided into two major groups: data-driven models and analytical models. Data-driven modeling uses techniques like system identification whereas analytical modeling creates a block diagram model that realizes mathematical equations (differential or algebraic equations) governing system dynamics. A type of analytical modeling is physical modeling, where a model is created by connecting blocks that represent the physical elements that the actual plant consists of. This project takes the benefit from this kind of modeling. In other words, the heat shrink tubing machine (plant) is broken down to its physical building blocks and each block will be modeled separately. Next, all modeled building blocks connect together to model the whole system.

## 2.2. Model Development, Assumptions, and Parameters Estimation

MATLAB®/Simulink®[1] has been selected as the simulation environment. It is a multi-domain package that enables this software to be a perfect choice for developing control systems and testing system-level performance. Simscape™ library in MATLAB/Simulink provides building blocks from mechanical, electrical, thermal, and other physical domains that make it possible to model a complete physical system without dealing with mathematical equations directly.

The designer needs to set up some attributes for each building block. The created model in Simscape automatically generates the differential equations that represent the system's behavior. These equations are integrated with the rest of the system model, and are solved directly. Simscape elements connect together with physically modeled connections and that is why each parameter and variable has its own physical unit, with all unit conversions handled automatically [13].

The heat chamber in the developed prototype machine is attached onto a linear stage that is driven manually by rotating the screw rod with a hand wheel (Fig. 1.3) but now a DC motor is selected to perform the job automatically. In the developed system model which is shown in Figure 2.1, the DC motor block represents the equivalent electric circuit of the selected DC motor. It includes the electrical and torque characteristics of the motor. This model is based on the assumption of no electromagnetic energy being lost, and that is why the back-emf and torque constants will have the same values. The friction block next to the DC motor shows rotational friction between rotating parts that come into physical contact with each

[1] MathWorks Inc., www.mathworks.com

other. The friction value is calculated as a function of relative velocity. The worm gear and lead screw blocks represent the needed mechanisms for converting the rotational movement to linear displacement. The load force block model is an ideal source of force that is controlled based on the input signal. The word ideal means it is powerful enough to maintain constant force regardless of the velocity at the source terminals. The total 7.49 pound (3.4 Kg) load weight of the chamber (including clamps, shell, and base) and two heat guns will give us a 33.32 N load force for simulation purposes. Eventually, the position sensor block simulates a translational motion sensor, and its outputs are linear speed and position.

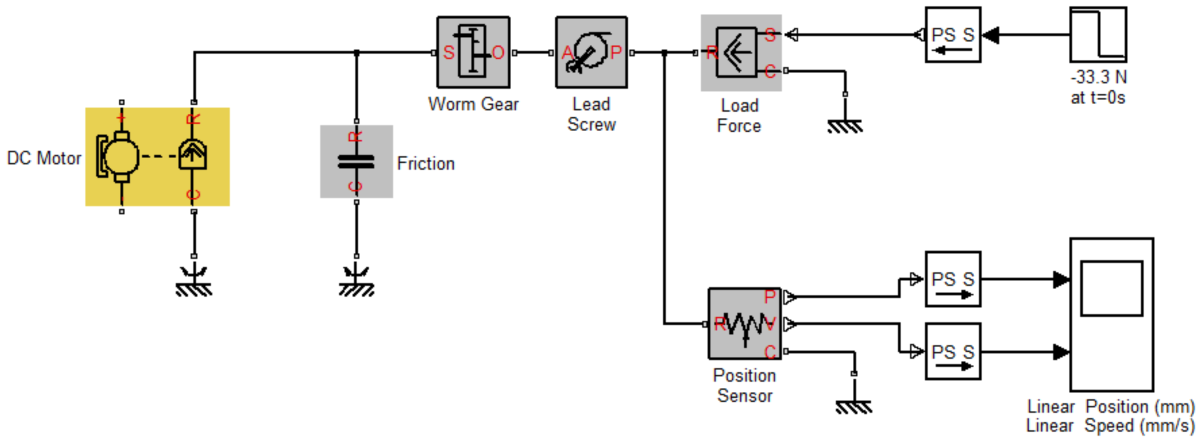


Fig. 2.1: Linear Positioning System MATLAB/Simulink Model

## 2.3. Model Validation

Model validation guarantees that the simulation results match with the observation from the physical system [11]. Mostly, simulation models are developed for a specific application and their validity measured for that purpose because it would be too costly and time consuming to develop a valid model for a wide range of applications. Because of that, the model developed for the heat shrink tubing machine only looks at the electro-mechanical parts while the thermal behavior has not been considered. Tests are performed until acceptable confidence is achieved concerning model validity in its intended application [12].

## 2.4. Controller Design

The plant model resulting from the former step is used to design the control unit. In this phase, simulating the developed controller in conjunction with the system under control behavior will help us monitor the results, detect the modeling errors, and modify the controller parameters.

### 2.4.1. System Level Logic Control

The automatic heat shrink tubing machine will work while interacting with no other manufacturing facility except for the human operators. Observing the current process that is being performed manually by the operators, it has been understood that the proposed machine should work in different modes. The operator will start up the machine, place the wire harnesses into the machine, and control and supervise its operation mode.

The operation modes will be changed by event signals. Events are supposed to be triggered by devices, such as push buttons, sensors, and internal signals, which are defined in the control logic and rely on internal variables. Such control scheme could be perfectly modeled with a discrete event system (DES) that has discrete state space and an event-driven dynamic, i.e., the state can only change as a result of instantaneous events occurring asynchronously over a time interval [8]. In this context, state-charts have been traditionally used to describe these kinds of systems (although, there are also other methods such as Petri-Net models). MATLAB/Simulink possesses the capability to develop and simulate a controller in a state-chart diagram. Figure 2.2 illustrates the corresponding idea for supervisory control purposes. According to this diagram, after the operator powers on the machine (PWR=1), the system status is transferred from its initial off state to the stand-by state. In stand-by mode the heat chamber is pre-heated and placed in the park position to be ready for the shrinking process. Once the operator issues the appropriate command via the user interface (RUN=1), the system state will be changed to the operation state and it means the shrinking process starts. It is always possible that the machine’s status moves to the designed emergency state and there is a bunch of reasons for that. Emergency situation could be raised up by the operators or the internal signals. Ultimately, after observing the alarms list and appropriate reaction, the operator can restart the system from the initial off status.

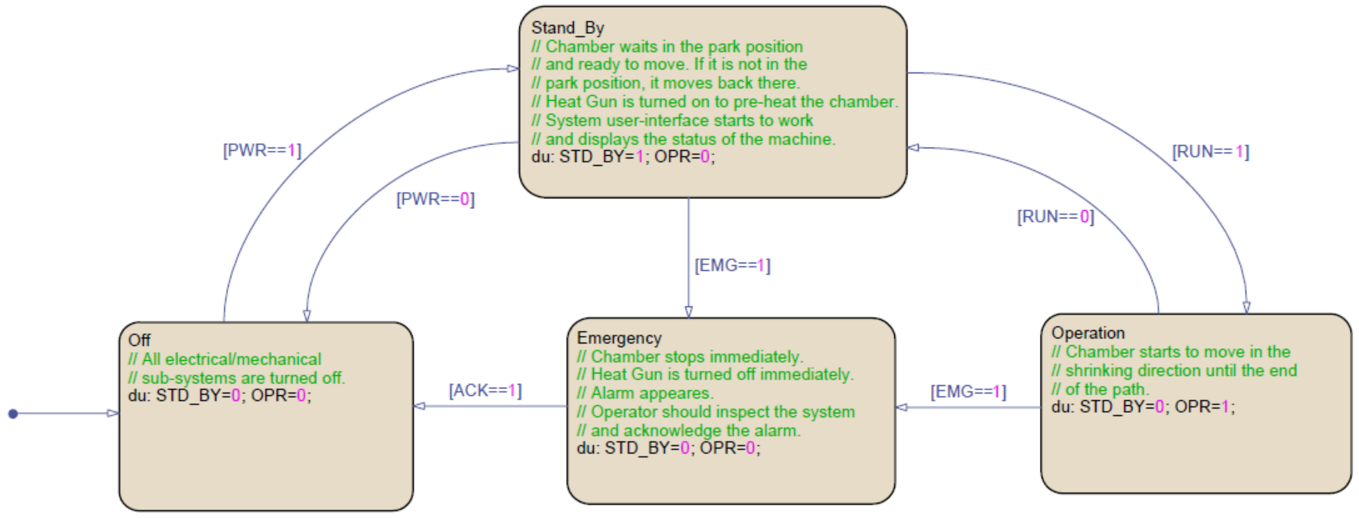

Stand_By
// Chamber waits in the park position
// and ready to move. If it is not in the
// park position, it moves back there.
// Heat Gun is turned on to pre-heat the chamber.
// System user-interface starts to work
// and displays the status of the machine.
du: STD_BY=1; OPR=0;
[PWR==1]
[PWR==0]
[EMG==1]
[RUN==0]
[RUN==1]
Off
// All electrical/mechanical
// sub-systems are turned off.
du: STD_BY=0; OPR=0;
[ACK==1]
Emergency
// Chamber stops immediately.
// Heat Gun is turned off immediately.
// Alarm appeares.
// Operator should inspect the system
// and acknowledge the alarm.
du: STD_BY=0; OPR=0;
[EMG==1]
Operation
// Chamber starts to move in the
// shrinking direction until the end
// of the path.
du: STD_BY=0; OPR=1;

Fig. 2.2: State-Chart Diagram to Manage the Operation Mode

### 2.4.2. Chamber Position Controller

As described in Chapter 1, the heating process will be managed by controlling the position and velocity of the heat chamber. Because the chamber will be driven by a nut on a screw rod that is fixed to a DC motor shaft, the chamber linear speed is proportional to the DC motor angular speed; hence, our efforts are focused on controlling the motor's speed.

Among all existing approaches, one of the most commonly used control methods in industry is the PID control. Wide availability and simplicity of use are the major advantages of the PID control method. Even complex control systems may use controller units whose main control building blocks are PID control modules. The PID controller has a long history and the change of control technology from the analog systems to the digital computers has not retired it [18].

PID control is an abbreviation commonly given to a three-term controller where P stands for the proportional term, I for the integral term, and D for the derivative term. It can be seen in a typical feedback control system, Figure 2.3, that the PID block-set is placed right after the calculation of the error signal, in order to feed in the control signal to the actuator. An ordinary parallel structure PID control can be given the following equivalent mathematical representations in time and Laplace domain [18]:

Time domain: $$u_c(t) = K_P\, e(t) + K_I \int^t e(\tau)d\tau + K_D \, {}^{de}/_{dt}$$

Laplace domain: $$U_c(s) = \left[K_P + {}^{K_I}/_{s} + K_D s\right] E(s)$$

The proportional term is used when the controller action is to be proportional to the amount of the error signal by a factor of $K_P$. Increasing $K_P$ speeds up the response and reduces (but usually does not eliminate) the steady state offset. The integral term is used when it is required that the controller correct any steady offset from a constant reference signal value. The integral term decreases the steady state error without the use of excessively large controller gain. Finally, the derivative term uses the rate of change of the error signal by the $K_D$

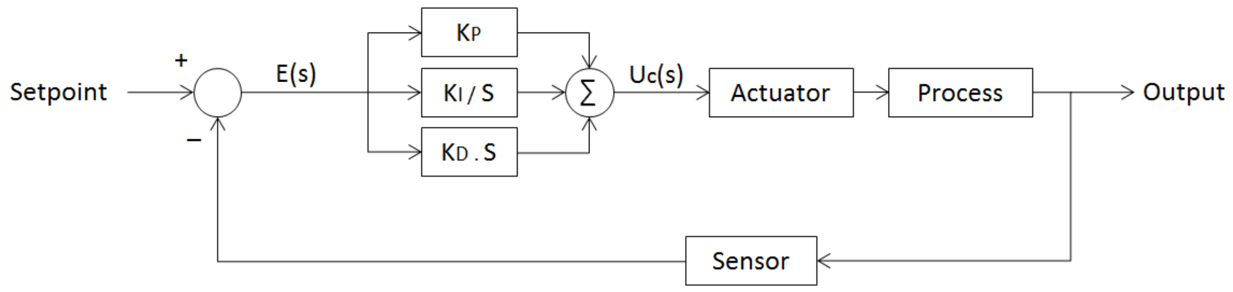


Fig. 2.3: Typical Feedback Control System with Parallel Structure PID Controller

factor, and it introduces an element of prediction into the control action. Using the derivative control demands more care than using proportional or integral control due to possible noise amplification. This reason along with simplicity incentive and satisfactory performance motivates using the PI, not the PID, for the current specific application of motion control. As described earlier, the integral term in the PI controller still has the capability of making the steady state error negligible for a step change in the reference signal which is why engineers often tend to use this type of controller in motion control applications.

One beneficial characteristic of the PID controller is that two blocks can be used in a series structure to result in a better closed-loop dynamic behavior. This architecture is called cascaded PID control [15]. Similarly, in a typical cascade PI controller, there are two PI blocks arranged in a way that one controller provides the set point for the other. One block acts as the outer loop controller (master), which controls the primary physical parameter, here the angular speed of the motor. The other block acts as the inner loop controller (slave), which reads the output of the outer loop controller as a set point, usually controlling a more rapid changing parameter, here the motor’s current. The idea of the second loop is to secure the armature current and so on govern the target angular speed.

It can be mathematically proven that the working frequency of the controller in cascade style is increased and the time constant of the whole system is reduced [16]. The other benefit of the cascade control approach is that it provides limits on the secondary variable which is the armature current. However, cascade control might result in a more complex system, increasing the control cost due to more instruments, and requiring more difficult tuning.

To design the controller structure and tune its parameters, the performance objectives of the system must be defined first. Tuning of a PI controller involves choosing the $K_P$ and $K_I$ parameters that provide the required system dynamics, including response speed, settling time, and proper overshoot rate, all of which guarantee the system stability and acceptable steady state error. The most common method for tuning is based on trial and error. There are also other analytical and practical methods to tune up a typical PI controller: classical control methods in the frequency domain, and Ziegler-Nichols, to name but two. These methods all provide a first approximation and the result usually needs further manual adjustment by the designer [14].

In the current project, the tuner utility in the PID controller building block is used to tune up the controller parameters. In the case of the cascade control scenario, the inner loop controller must be tuned first while the outer loop is not applied. Then, the inner loop needs to be in tracking mode when the outer loop is being tuned. The following values result for both master and slave controllers in the simulated system with reference tracking as the main objective:

| | | |
|---|---|---|
| Slave Controller (Motor Current): | $K_P = 1$ | $K_I = 30$ |
| Master Controller (Motor Speed): | $K_P = 0.3$ | $K_I = 0.3$ |

Figure 2.4.A and Figure 2.4.B show the schematic block diagrams of the designed controller in MATLAB/Simulink. The angular speed set point in the master controller will be selected after running a series of experiments to find what motor speed most appropriately corresponds to the chamber linear speed, in order to have a desirable heat exposure performance. Because of the temperature disturbance, the required exposure time and,

consequently, the desired motor speed are subject to change. Therefore, the motor speed set point should be compensated with respect to the chamber temperature variations.

In order to achieve this goal, the actual chamber temperature could be measured and after calculation of its deviation from the nominal shrinkage temperature, the motor speed set point could be regulated accordingly. A linear lookup table is used to translate the temperature fluctuations to the reference speed variations. Figure 2.5 depicts how the linear position and speed are regulated in response to variations in the chamber temperature that are in vicinity of its nominal value.

In Figure 2.6, the armature current and chamber speed are measured when there is a pulse in the chamber temperature. Needless to say due to the thermal transfer function of the double-gun heat chamber, it is almost impossible for the temperature to have an abrupt change and this case only could be useful for controller performance evaluation in simulation world. The graphs illustrate how the chamber linear speed follows the changes in the chamber temperature or outer loop reference signal, while there is not any specific change in the armature current or inner loop output signal.

According to the wire harness characteristics, the total travel length of the heat chamber should be almost 26 inches (or approximately 660 mm). The experiments performed on the manual prototype determined that the needed linear speed for the chamber with two heat guns is 0.15 inches/second (3.8 mm/sec) and, as Figure 2.5 reveals, the designed control system could address this specification very well.

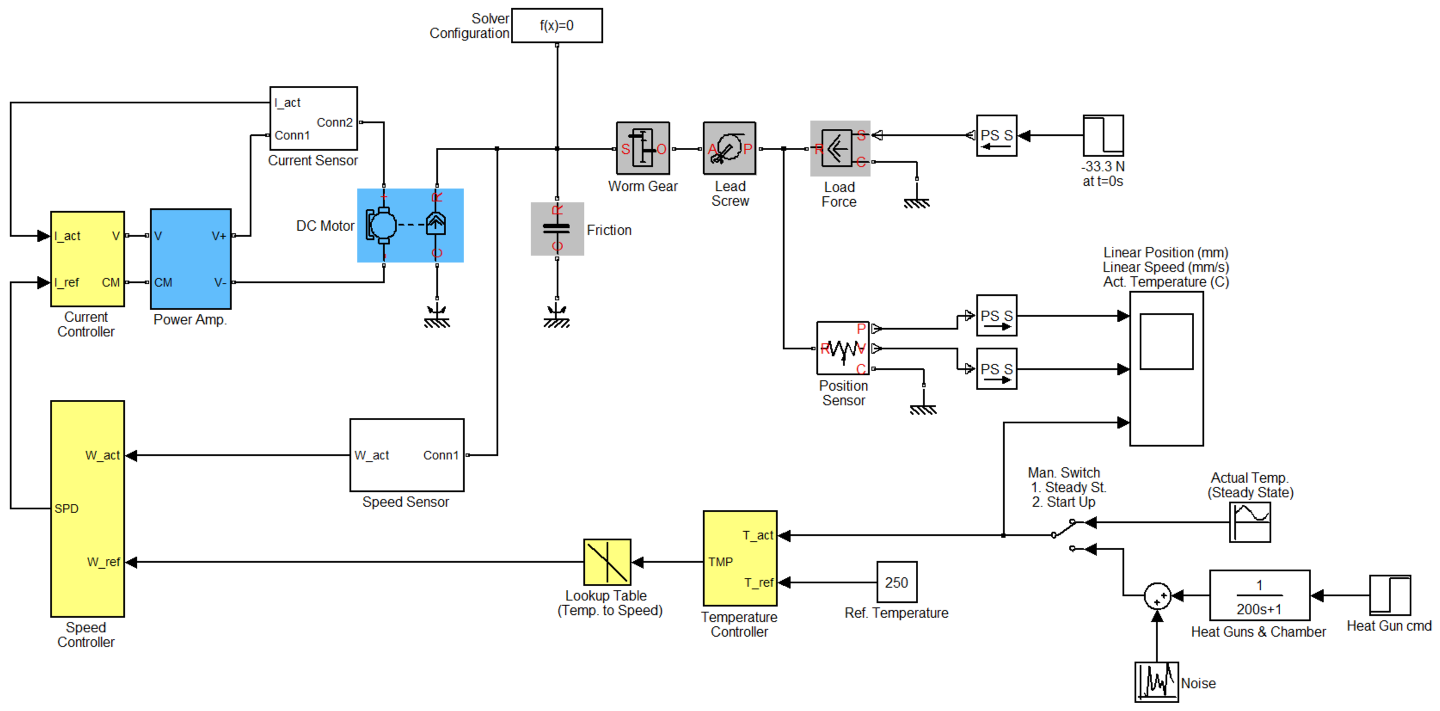


Fig. 2.4.A: System Model Along with the Designed Controller Block Diagram

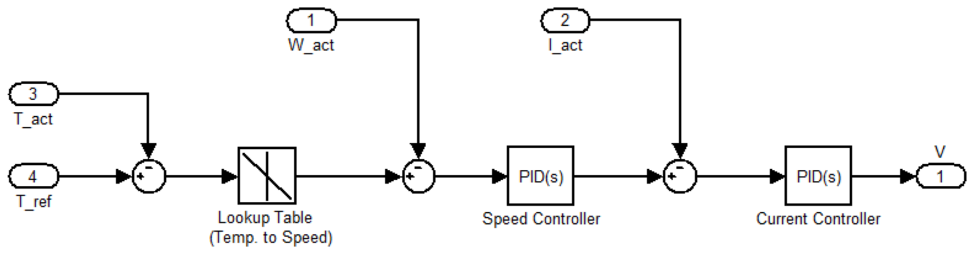


Fig. 2.4.B: Controller Block Diagram in Detail

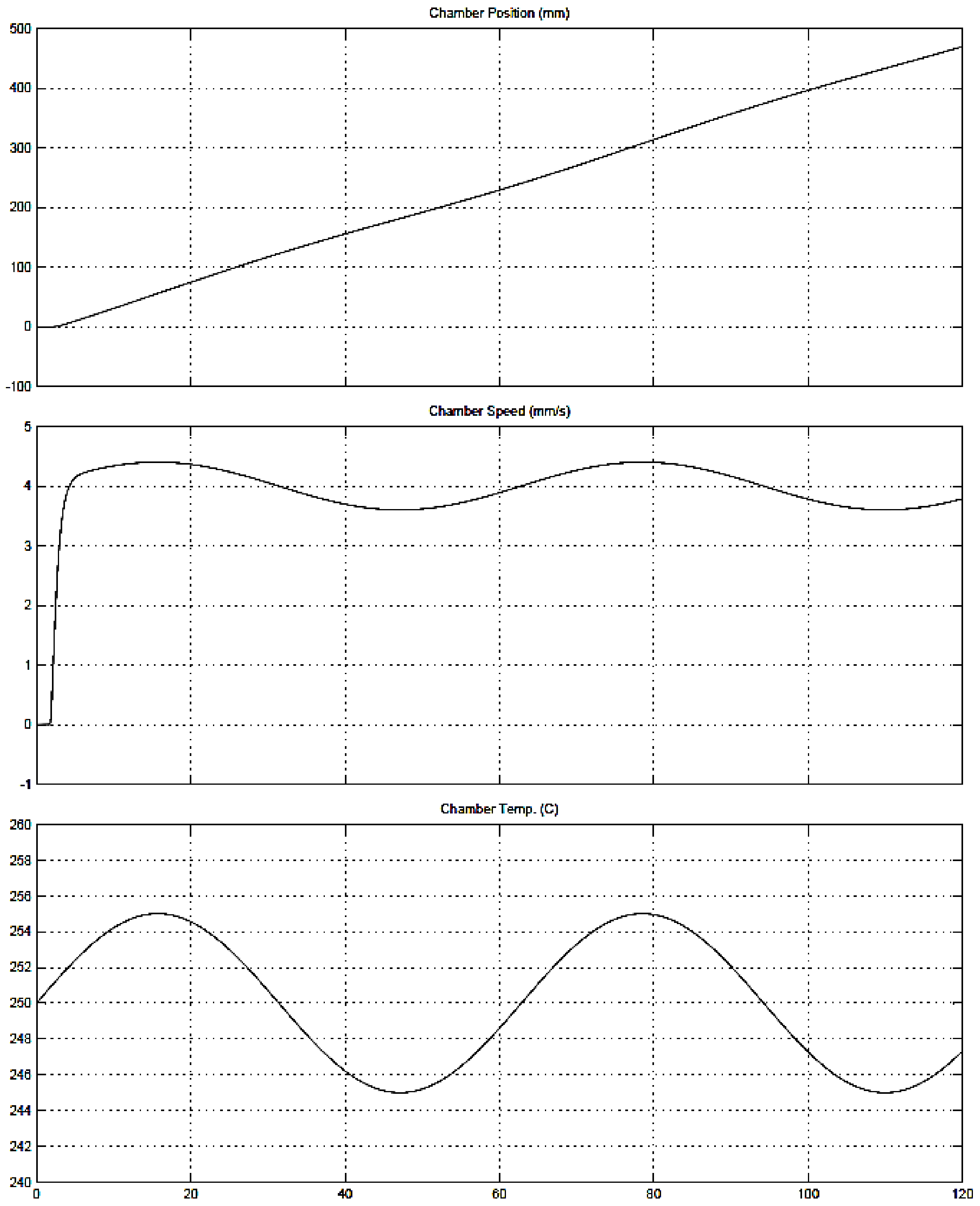


Fig. 2.5: Chamber Position and Speed Respect to the Temperature Variations

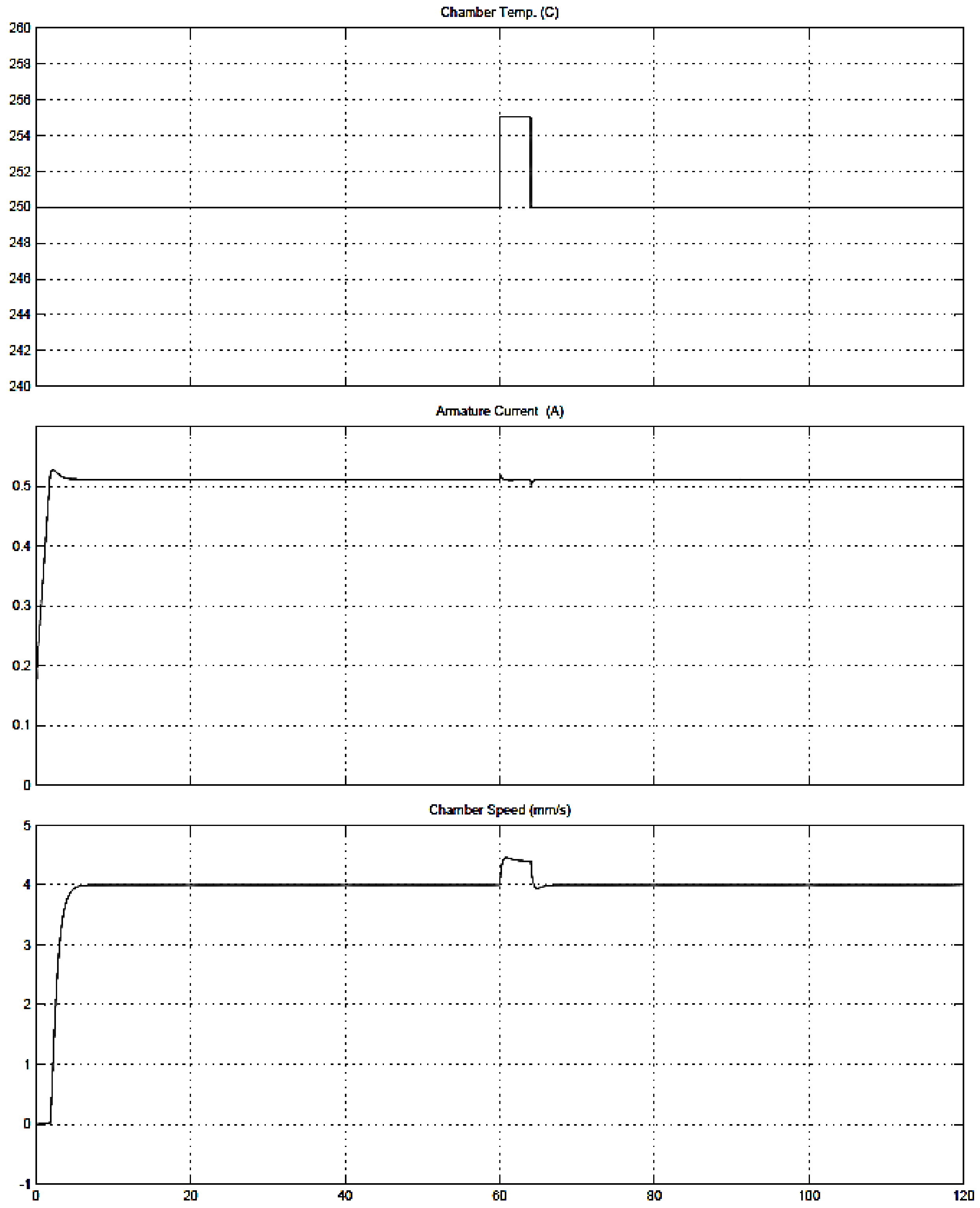


Fig. 2.6: Armature Current and Chamber Speed in Response to the Rapid Changes in Temperature

# CHAPTER 3

# CONTROLLER IMPLEMENTATION

The next phase, after designing the controller and testing it in the simulation environment, is the implementation of the control system based on the results of the simulation. This phase includes implementation of the hardware and software, and the examination of the control unit performance according to the model-based design guidelines.

## 3.1. Hardware Implementation

With respect to the reasoning presented in Chapter 1, PLCs stand to be a good choice for implementing the controller in the current application. They sufficiently meet the computation needs, either arithmetically or logically. Their effectively shielded packaging lets them work well in industrial environments having electromagnetic noises and dust. Because power consumption is not as much of a concern in typical industrial controller design as it is a concern in portable devices design, this factor does not play a determinant role in hardware selection process. Other benefits of PLCs are ease of upgrade to higher performance versions, availability of technical support by third-parties, possibility of performing minor logic modifications by trained technicians, and existing standard communication protocols for HMI systems.

Another justification in using PLCs as compared to using other kinds of embedded controllers is that a PLC is designed to work in a real- time manner. The inputs are read at one time and saved. The logic then is processed sequentially and, at the end, the outputs are updated. This allows precise timing of execution and minimizes endless loops. This is an important concept in industrial automation systems where an undesired delay could result in a costly consequence. Always, the cycle time that it takes to execute the logic is measured and if it exceeds a predefined value, the developer knows that there is a real problem and the PLC needs to execute the timeout sequence. Although, most of the mentioned features are feasible in many other kinds of embedded controllers, because PLCs come as pre-configured structures, any allocated time and cost could be spent on control algorithm instead of implementation techniques.

Among existing PLCs, the CompactLogix®[1] controller family is a good candidate for the intended application. The process capability, speed, and supported I/O size are always important factors in the selection of the right hardware. CompactLogix family controllers are designed for medium range control applications. The 1769-L3X series offers a modular configuration which is suited for flexible architectures. Table 3.1 lists the modules provided and the equipment for building an integrated CompactLogix controller in the current project [20, 21, 22, and 23], while Figure 3.1 shows the controller used in the experiments.

[1] Rockwell Automation Inc., www.rockwellautomation.com

Table 3.1: Modules and Equipment Used in the Controller Hardware

| **Catalog Number** | **Description** |
|---|---|
| 1769-L32E | CompactLogix EtherNet/IP Controller |
| 1769-PA2 | Compact Expansion Power Supply 120/240V AC Input 2 A @ 5V DC Output Module |
| 1769-IQ16 | Compact 16 Point 24V DC Sinking/Sourcing Input Module |
| 1769-OW8 | Compact 8 Point AC/DC Relay Output Module |
| 1769-IF4FXOF2F | Compact Combination Fast 4 In/2 Out Analog Module |
| 1769-ECR | Compact I/O end cap |
| 1747-CP3 | RS-232 Cable |
|  | 4-Port Ethernet Switch and Standard Ethernet Cable with RJ-45 Connector |

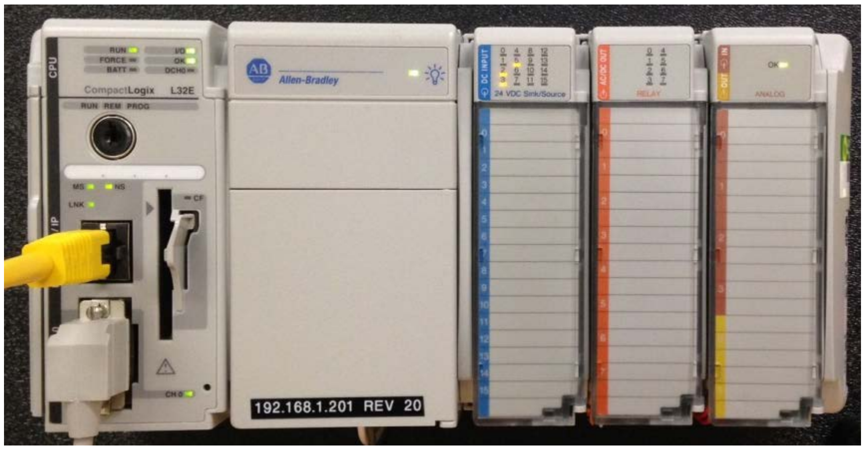


Figure 3.1: Modular Based CompactLogix Controller

## 3.2. Software Implementation

After selecting the appropriate hardware platform, software implementation is carried out according to the existing programming standards. As far as PLCs are concerned, a variety of programming languages are based on the IEC-61131-3 standard, with each one fit for a specific application. For example, Structure Text is known as a high level PLC programming language which is used for complicated algorithms while graphical languages like Ladder Diagram or Function Block are more suitable for simple logics. The latter group is not as flexible as the Structure Text but it is easier to be traced and debugged, and that is why engineers tend to use this kind of language most often. In the existing project, Structure Text programming language is selected to implement the part of the controller software that corresponds to the system level logic (supervisory control), while Ladder Diagram is used for the chamber position controller.

The Structure Text code for system level logic is created by automatic code generation from the controller developed in MATLAB®/Simulink®[1]. This automatic code generation will dramatically decrease the possible errors and development time. Using the automatic coder utilities, control system designers can spend more time to fine tune the algorithm through rapid prototyping and experimentation, and less time on coding effort. The design and test processes are completely iterative, meaning that at any phase, the designer can return to the original model, modify the parameters, and regenerate the code. The MATLAB/Simulink simulation environment provides the following automatic code generation capabilities:

[1] MathWorks Inc., www.mathworks.com

- Simulink Embedded Coder™: C/C++
- Simulink HDL Coder™: Verilog/VHDL
- Simulink PLC Coder™: Standard Structure Text

Among the above utilities, Simulink PLC Coder makes it possible to convert a designed MATLAB/Simulink control model into a Structure Text language program. The code generated is imported into the relevant IDE (Integrated Development Environment). As a result, the application code will be compiled and deployed to the PLC (Fig. 3.2).

Although Simulink PLC Coder module can generate hardware-independent Structured Text code from Simulink models, developers need to identify the ultimate hardware model for implementation because of the existing differences in data types and instructions syntax among PLCs coming from different vendors. IDEs supported by PLC coder include B&R Automation Studio®, PLCopen, Rockwell Automation® RSLogix™ 5000, Siemens® SIMATIC® STEP® 7, and Smart Software Solutions CoDeSys [17]. Rockwell Automation RSLogix 5000 is used as the IDE for numerous types of Allen-Bradley controllers including CompactLogix. Appendix A contains the PLC code generated with PLC Coder utility pertinent to the system level logic control designed in Chapter 2.

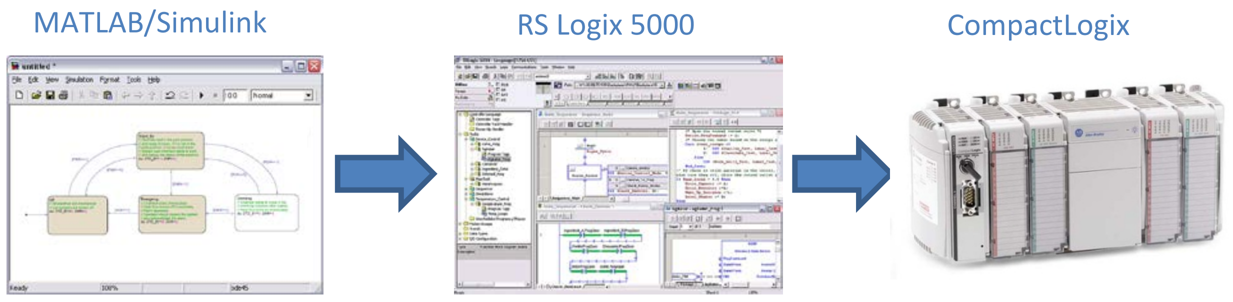


Figure 3.2: PLC Code Generation and Deployment Sequence for System Level Logic

Figure 3.3 represents the Ladder Diagram program developed for the chamber position controller. In this figure, the second rung performs the operations needed to calculate the motor speed set point with respect to the chamber temperature variations, as described earlier in Chapter 2. Then, two PID blocks are placed in the third rung to create the cascade loops. The internal operation of the two PID blocks connected in master and slave manner in RSLogix 5000 is represented in Appendix B in the form of a block diagram [24].

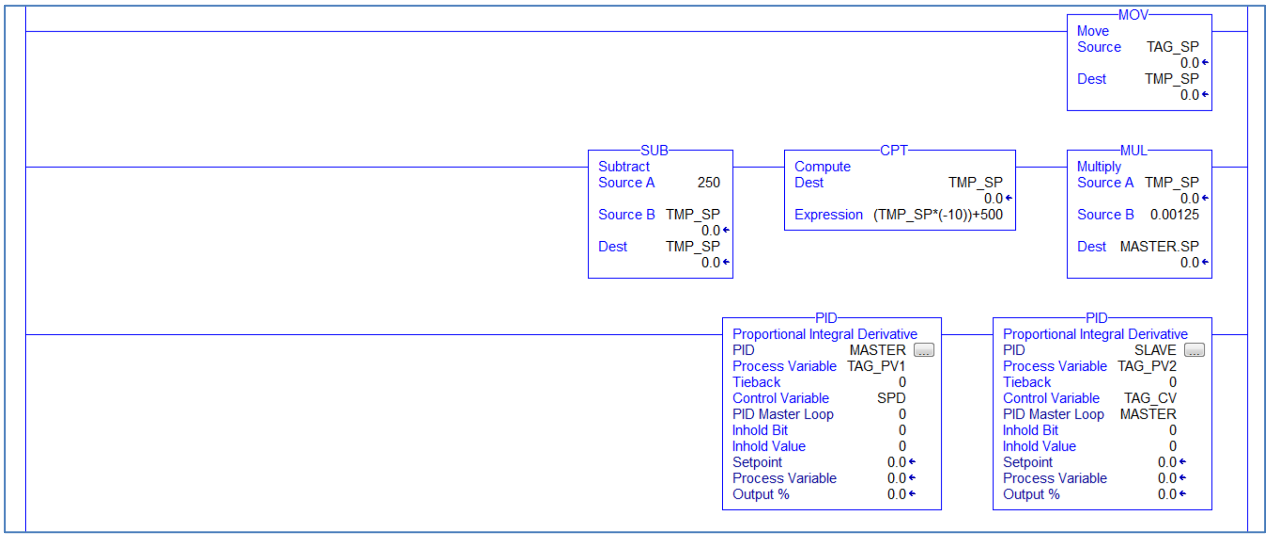


Figure 3.3: Chamber Position Control Ladder Diagram

## 3.3. Position Controller Test

Before integrating the controller implemented with the real heat shrink tubing machine, a round of tests should be performed to check the implemented controller performance. These tests will help to detect any possible problem and fix it in the right time and before driving the real instruments.

Due to the computational and graphical capabilities of MATLAB/Simulink, it makes sense to keep this software package in the controller test process, even after the preliminary design and simulation phase. In Chapter 2, both controller and system under control (heat shrink tubing machine) were modeled and examined in the MATLAB/Simulink environment, but now what is subject to test here is the real control unit. So, in the next test step, the real controller (PLC) is connected to the simulated machine model in MATLAB/Simulink, and the implemented controller performance is examined. Generally, this approach is addressed as Hardware-In-the-Loop (HIL) test; however, in a typical HIL test the simulated object under control should be run on a hardware platform and an operating system, with a real-time kernel. It is clear that MATLAB/Simulink cannot present a real-time behavior while running on an ordinary operating system like Microsoft Windows®[1], but this test still could be helpful. Figure 3.4 pictures the developed experimental setup.

In the experimental setup, the first step is to provide a solution for data exchange between the controller and the heat shrink tubing machine model in MATLAB/Simulink. The next paragraphs detail this step.

### 3.3.1 Data Exchange between MATLAB and the Controller

One solution for feeding the data needed from MATLAB to the controller unit and vice versa is the use of special I/O modules that are installed on personal computers and supported by MATLAB, for example, those by National Instruments[2] or Quanser[3]. Such systems may be

[1] Microsoft Corporation, www.microsoft.com
[2] www.ni.com
[3] www.quanser.com

suitable for usual laboratory tests but are rarely used in industrial applications because not only they increase the test cost considerably, but also create many integration problems [5].

Another solution could be the construction of an API (Application Programming Interface) in MATLAB which listens to the traffic on the PLC network and, if necessary, returns some data. In comparison with the former solution, the main advantage of this approach is that MATLAB does not have to be integrated with the peripheral cards and its main disadvantages are that the building up of such an interface is time consuming and the result is not standard [5].

Eventually, a common solution would be to use the OPC (OLE for process control) standard, a solid and efficient method to establish a communication between MATLAB/Simulink and the PLC unit. In addition to the HIL test, this feature can also be used to perform a real-time parallel optimization procedure. In this case, the process under control would continue running independently and MATLAB/Simulink would be executing all the necessary mathematical operations in parallel and adjusting the controller parameters accordingly [5]. OPC technology utilizes a software interface with a client and server mode based on COM/DCOM (Component Object Model/Distributed Component Object Model). COM/DCOM offers a general standard mechanism for client's and server's communication. OPC technology makes it possible for software and hardware from different brands to integrate, and presents an easy and effective solution for communication between PC based applications such as MATLAB/Simulink on one side, and process devices such as PLCs on the other side [6].

Figure 3.5 demonstrates the data communication architecture schematic in the test system developed. In MATLAB, the OPC Toolbox provides blocks in the Simulink environment

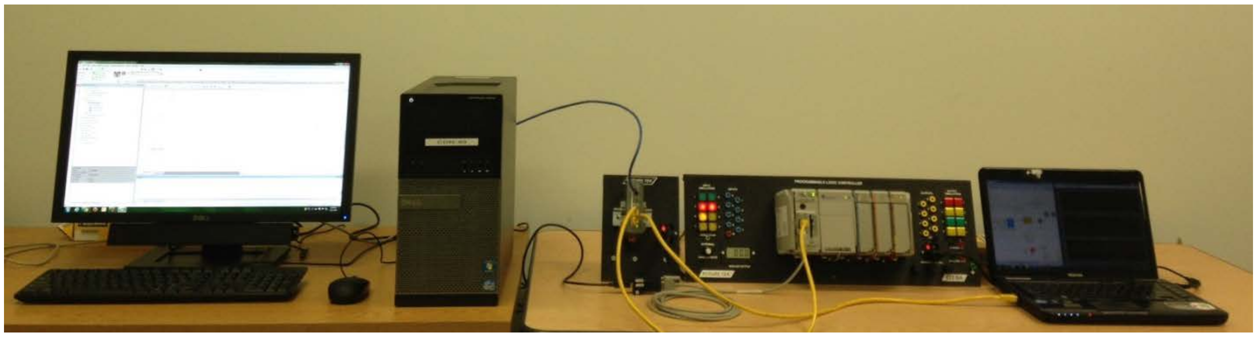


Figure 3.4: Experimental Setup

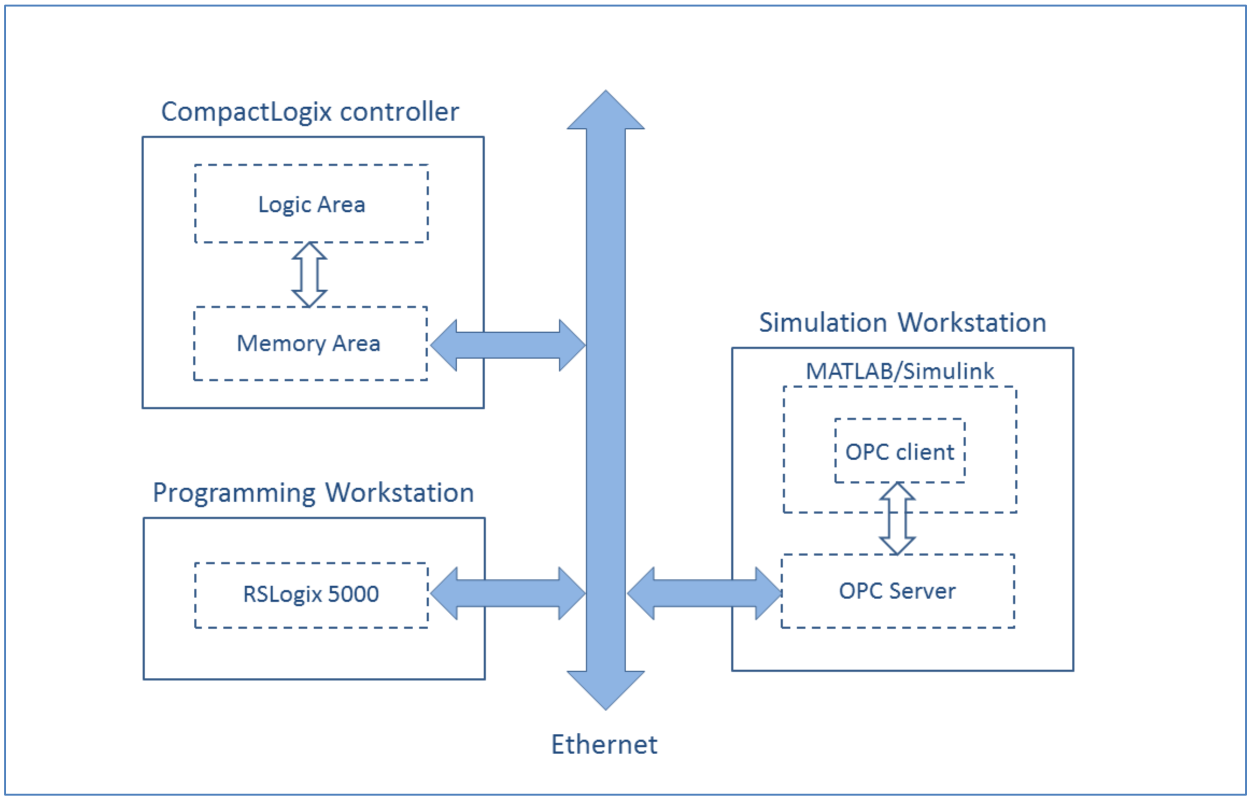


Figure 3.5: Data Communication Architecture in the Experimental Setup

for interacting with a typical OPC server. Figure 3.6 shows how OPC Read and OPC Write blocks are placed in a simple example in MATLAB/Simulink. This example is used to establish and test the communication link between the controller and MATLAB/Simulink in preliminary steps. The sourced signal is written to the OPC Server through the OPC Write block. Then, the controller passes this signal to another memory address and the OPC Read block gets it back, and sends to the Scope. Figure 3.7 illustrates the original source and received signals in the same plot. As shown, the data samples that are read from the server are delayed by 0.1 seconds from the original signal that equals two sample time intervals (0.05 seconds): one sample time to write and another sample time to read the data.

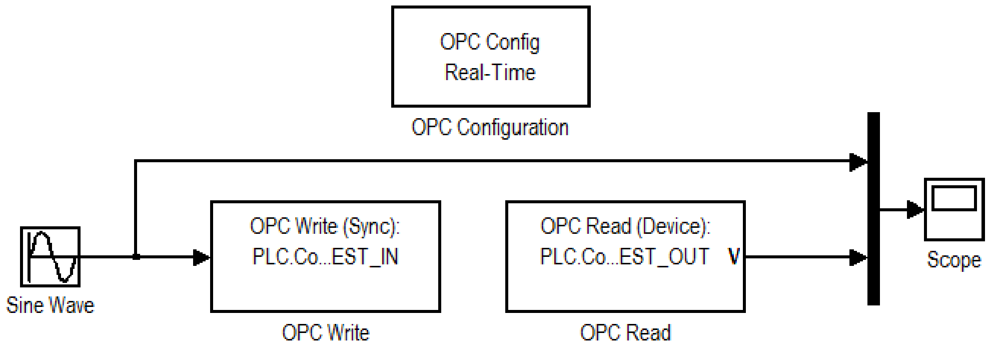


Figure 3.6: A Simple OPC Read/Write Example in MATLAB/Simulink (Block Diagram)

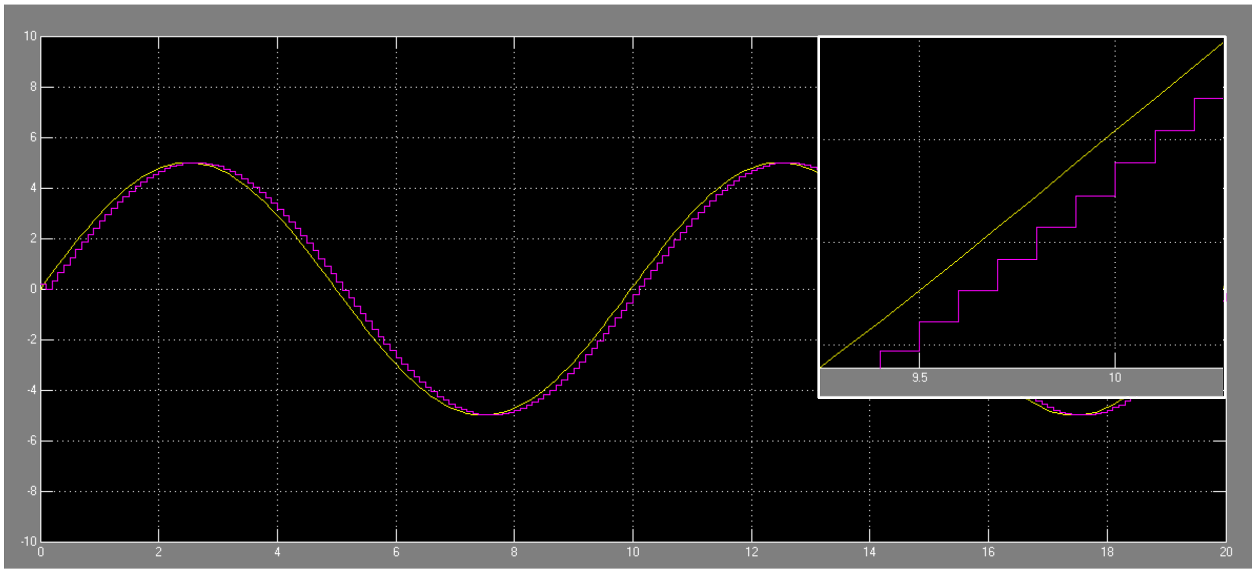


Figure 3.7: OPC Read/Write Example in MATLAB/Simulink (Plots)

### 3.3.2 Test Results

The test system hardware consists of a process simulation workstation, network switch, PLC, and PLC programming workstation (Fig. 3.4). The process simulation workstation has two features: MATLAB/Simulink that simulates the process, and the OPC server which is installed in the same computer so that the communication between the OPC server and the PLC is achieved via this computer’s network interface. Although in the final system, the controller signal transfer is done via I/O modules, in the test system based on OPC server, signal transfer is achieved temporarily by the PLC’s memory area. Table 3.2 lists the signals exchanged in the test system. The communication between the process simulation workstation and the controller is Ethernet based, and the network switch is utilized for this purpose.

The original MATLAB/Simulink model described in Chapter 2 is modified to delegate the control functions to the PLC (Fig. 3.8). Three process signals that are measured and sent to the PLC via the OPC Write block include chamber temperature (to determine the motor speed set point), motor speed (as the master PID loop variable), and motor current (as the slave PID loop variable). The process model receives the control signal (armature driving voltage) from the PLC side via the OPC Read block.

Table 3.2: Exchanged Data between MATLAB/Simulink and PLC

| **Signal Name** | **Description** | **From** | **To** |
|---|---|---|---|
| TAG_SP | Set point Manipulator (Chamber Temperature) | MATLAB | PLC |
| TAG_PV1 | Process Value 1 (Motor Speed) | MATLAB | PLC |
| TAG_PV2 | Process Value 2 (Motor Current) | MATLAB | PLC |
| TAG_CV | Control Value | PLC | MATLAB |

Figure 3.9 includes motor reference and actual speed (which are not in scale due to a sensor calibration factor) and also the motor actual current in the case that both controller and process are modeled in MATLAB/Simulink. Figure 3.10 exhibits the same signals when the real controller (with the same $K_P$ and $K_I$ parameters) is connected to the process model in MATLAB/Simulink. A comparison of the two figures shows that the real controller performance is sufficiently close to the modeled controller, although they are not completely identical. Figure 3.11 and Figure 3.12 also provide chamber position and chamber speed plots in response to the abrupt variation in chamber temperature for the modeled and real controller, respectively.

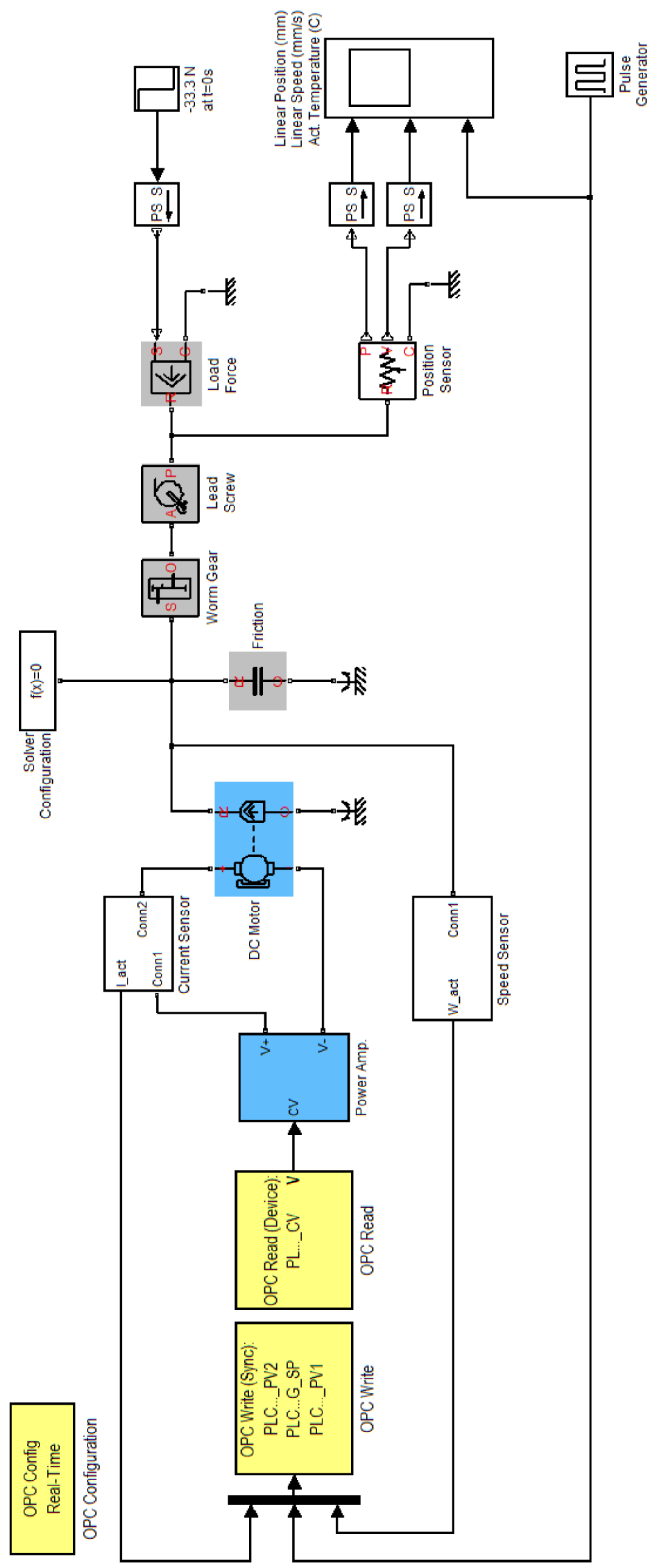


Figure 3.8: Modeled Process in MATLAB/Simulink Connected to the Real Controller (PLC)

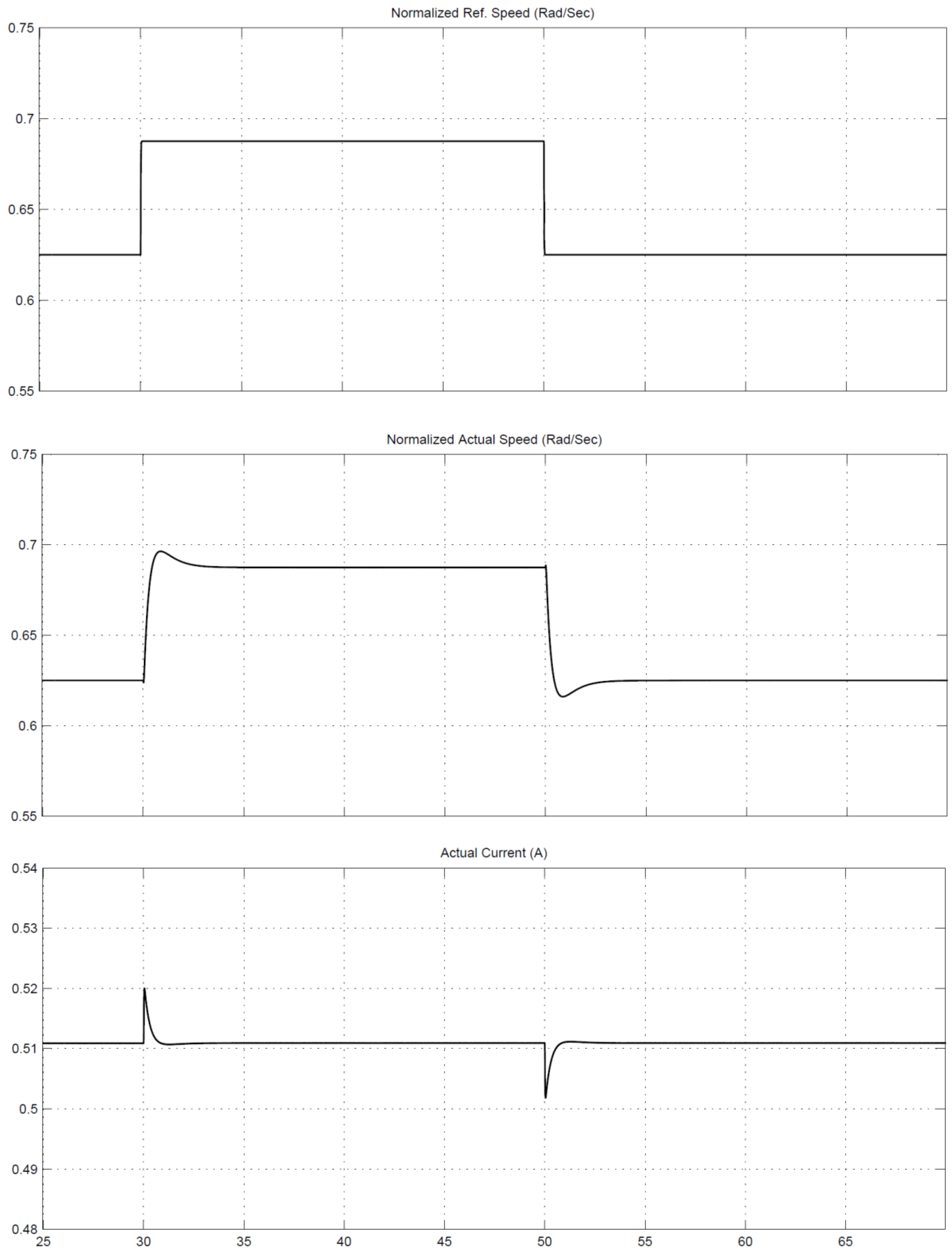


Figure 3.9: Motor Speed and Current (in MATLAB/Simulink) for the Modeled Controller

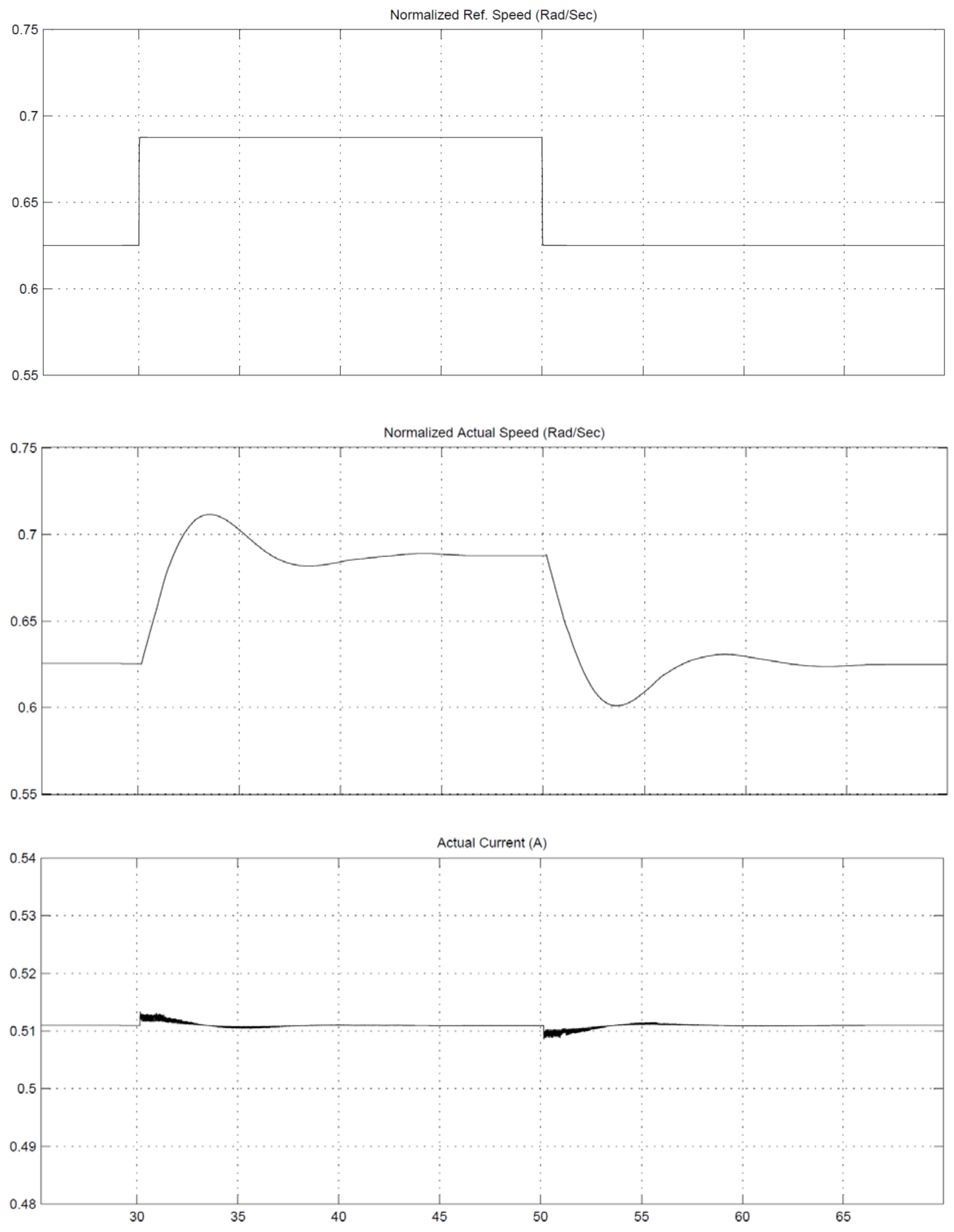


Figure 3.10: Motor Speed and Current (in MATLAB/Simulink) for the Real Controller

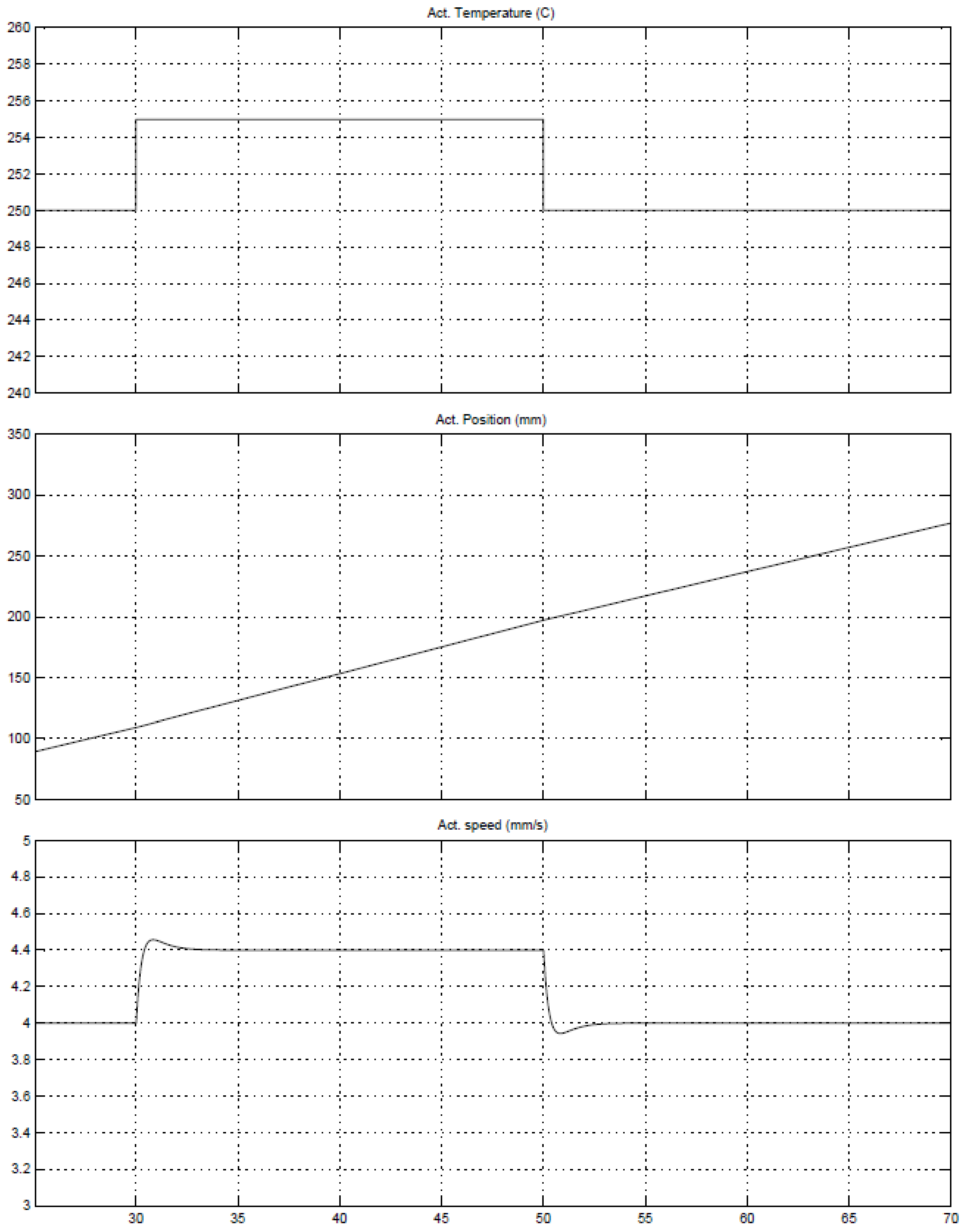


Figure 3.11: Chamber Position and Speed (in MATLAB/Simulink) for the Modeled Controller

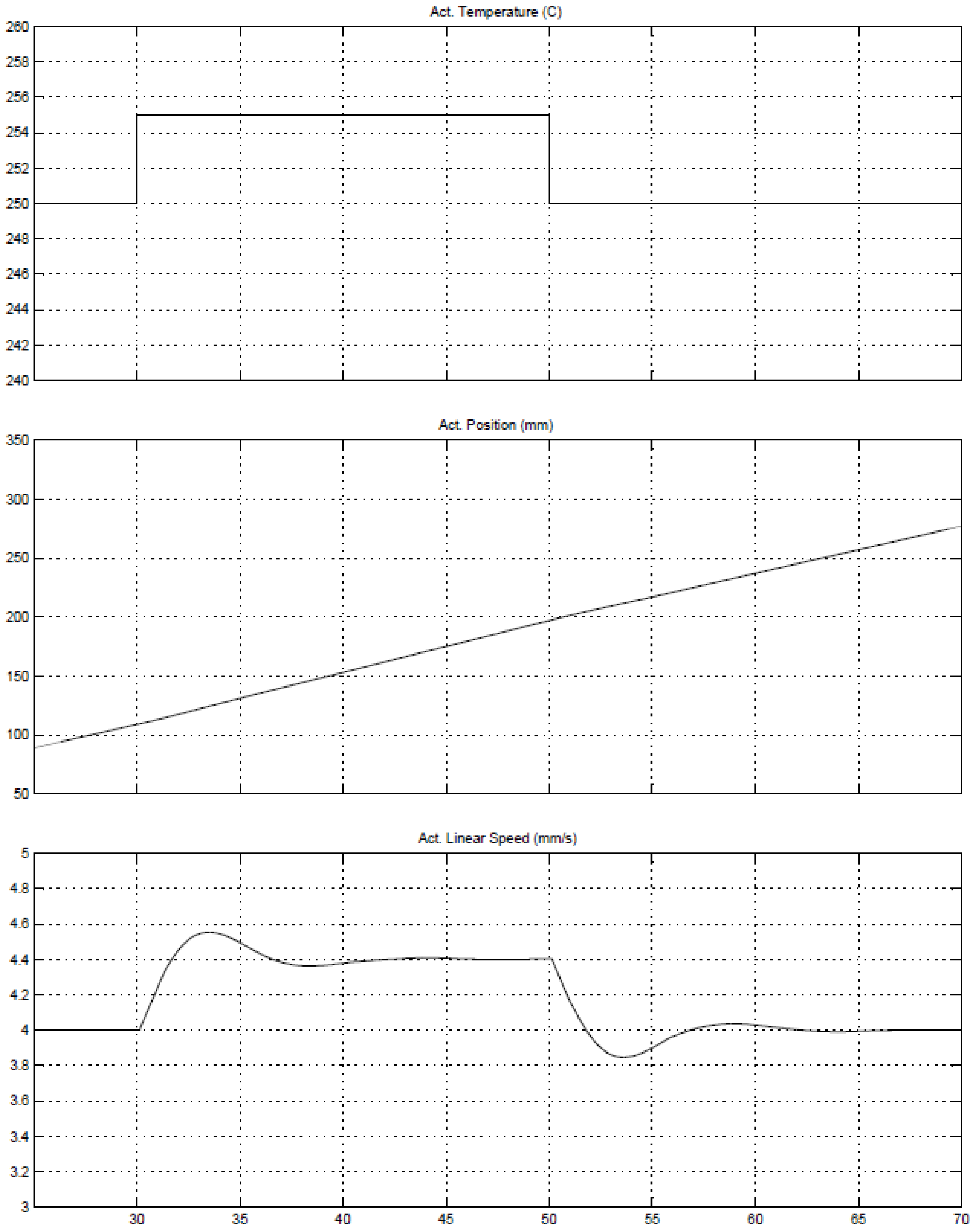


Figure 3.12: Chamber Position and Speed (in MATLAB/Simulink) for the Real Controller

# CHAPTER 4

# CONCLUSION AND FUTURE WORKS

This chapter summarizes the work described in the thesis body, discusses the attained results, and presents potential ideas to be developed in the future as follow-up works.

## 4.1. Conclusion

This thesis introduced a problem in the wire harnessing industry and proposed a practical solution based on the current technology in the field of industrial control and automation. Although the wire harnessing industry is the focus, as an application-specific example, the used method could be expanded for a wide range of control engineering problems in different industries.

In a typical wire harnessing process, the specific kinds of tubes are fitted on the wire before making the needed connection. The tubes then are heated by hot air blowers (or other kinds of heat sources) to be shrunk and tightened on the wire harnesses. The heating process demands an intensive control to result in the desired tube profile. Typically, skilled operators accomplish the job manually, which is time consuming and insufficiently safe. The main effort in this thesis was concentrated on developing a control system to perform the process automatically. An automated heat shrink tubing process minimizes the operators' direct interaction, lowers the production cost over the long term, and improves quantitative and qualitative production indexes. For experiment purposes, a small-scale prototype machine was used and the idea could be generalized on a full-scale machine as well. The prototype machine consists of a radial heat chamber with one or two heat guns that moves horizontally by a linear positioning system. The chamber is designed in a way to slide along the wire harness and direct the hot air from the heat gun(s) onto the fitted tube uniformly via distributed channels. So, the positioning system moves the heat chamber as the proper shrinkage temperature level is reached. Among all existing positioning systems, an electromechanical device is selected because of its accuracy, needed power, and cost. The heat chamber is driven by a nut on a screw rod which is coupled to a servo motor's shaft. This screw-driven positioning system is well-known for its accuracy and repeatability and these characteristics make this kind of actuator a good selection for sliding the heat chamber in the heat shrink tubing machine.

Heat exposure time as a major factor in the heat shrinking process can be managed by controlling the linear speed of the heat chamber. Thus, the main duty of the control unit is to adjust the chamber speed, which is done by measuring the actual speed and temperature continuously. Among all existing motion control methods, the PID control was selected. Its wide availability and simplicity of use are the major advantages of the PID control method but the PI controller was used in this specific motion control application due to the possible noise amplification in PIDs. To secure the armature current while governing the angular speed in the motor, the cascade PI controller was introduced. It includes two PI blocks arranged in a way that one controller feeds the reference value for another one. In a cascade PI controller, one block acts as the master controller and controls the primary physical parameter (the motor's speed in this case). The other block behaves as the slave controller, which reads the output of the master controller as the set point signal and usually controls a more rapid changing parameter (the motor's current in this case).

The other control challenge is the variable temperature inside the heat chamber due to heat losses and imperfect heat gun operation. This variation causes the required exposure time and, consequently, the desired motor speed to change continuously. In other words, the motor speed set point should be adjusted with respect to the chamber's actual temperature. So, the actual temperature is measured and, after calculation of its deviation from the nominal shrinkage temperature, the motor speed set point is regulated accordingly. Finally, the proposed control unit should manage the operation mode of the machine. This supervisory control function was designed and modeled with a discrete event system in the MATLAB®/Simulink®[1] software.

The specifications of the controlled process, plant environment, and used control method make the PLCs perfect choices for implementing the control unit in this application. The heat shrink tubing machine works in a harsh industrial environment, and that is why PLCs are the best candidate. Additionally, because a PLC is an integrated control system as a unit package, it provides the needed isolation circuits, signal conditioning, and current/voltage amplification for interfacing with sensors and actuators stage.

PLC programming languages like ladder diagram or function block are more suitable for simple logics and they are easy to be traced and debugged. Ladder diagram was used for the chamber position control. In contrast, structure text language is known as a high level PLC programming method which is used for complicated algorithms. This language was selected to implement the part of the controller software that corresponds to the system level logic (supervisory control). The structure text code for system level logic is created by automatic

[1] MathWorks Inc., www.mathworks.com

code generation from the developed control model in MATLAB/Simulink. This automatic code generation decreases the possible errors and development time dramatically.

For test purposes, both controller and system under control (heat shrink tubing machine) were modeled and examined in the MATLAB/Simulink environment. Then, the implemented controller (PLC) was connected to the simulated machine model in MATLAB/Simulink, and the developed controller performance was examined. This approach is similar to the hardware-in-the-loop test; however, the simulated object under control was not running on a system with a real-time kernel. OPC technology was introduced as an easy and effective way for communication between PC-based applications such as MATLAB/Simulink, and process devices such as PLCs. Ultimately, the control unit performance was examined and the implemented controller responses followed the simulated results with adequate accuracy.

A characteristic that distinguishes this thesis from many other similar works is development of a PLC-based control system according to the model-based design guidelines. Although model-based control design is a known method, its standards and procedures have not been utilized enough in the PLC-based control projects. It does not necessarily mean that no effort has been done but traditional approach in developing PLC control systems is more common. The model-based approach used is an elegant way to generate the PLC code automatically, to simulate the system, and to save time and reduce the error contingency. In the mentioned approach, even after implementation of the controller, it is possible to switch back to the simulation phase, modifying the controller parameters, and after observing the simulated outputs, transferring the needed changes into the real controller. In other words, instead of traditional sequential-phases project accomplishment, the modern project execution methodology with iterative cycles was used. The following lines are the achievements of the current work in brief:

1. A real industrial need is addressed.
2. A solid framework for any similar control design project is developed.
3. MATLAB/Simulink automatic code generation is examined.
4. MATLAB OPC toolbox is examined.
5. Cascade PI speed control method in PLC is examined.
6. An experimental test bench is developed to be used for design and implementation of any other PLC based controller.

## 4.2. Future Works

While this thesis addressed the potential improvements in the heat shrink tubing process in the wire harnessing industry, still opportunities for extending the scope of the work exist. This section presents some of these potential topics.

### 4.2.1 Shrinkage Measurement

In the proposed control approach, the heat shrink process is controlled by governing the linear speed of the heat chamber and consequently by managing the heat exposure time. This approach may be translated to an indirect control method as we regulate the chamber speed for heat exposure time control. Another control approach may be to incorporate a feedback based on actual tube shrinking performance, which is the ultimate goal of the process. In other words, this is the tube shrinkage that is subject to control, not the heat exposure time. To measure the shrinkage performance, the control system should directly sense the tube thickness before and after heating (like the thing that the human operator does visually in a manual operation case). Any appropriate thickness measurement method can address this need. Laser-based diameter measuring could be one option [25, 26]. Another idea is to add a flexible strain-gauge instrument around the exit side of the chamber to detect the cable diameter. Figure 4.1 represents a typical conceptual design of this instrument. All these ideas may be considered as follow-up studies after the current project.

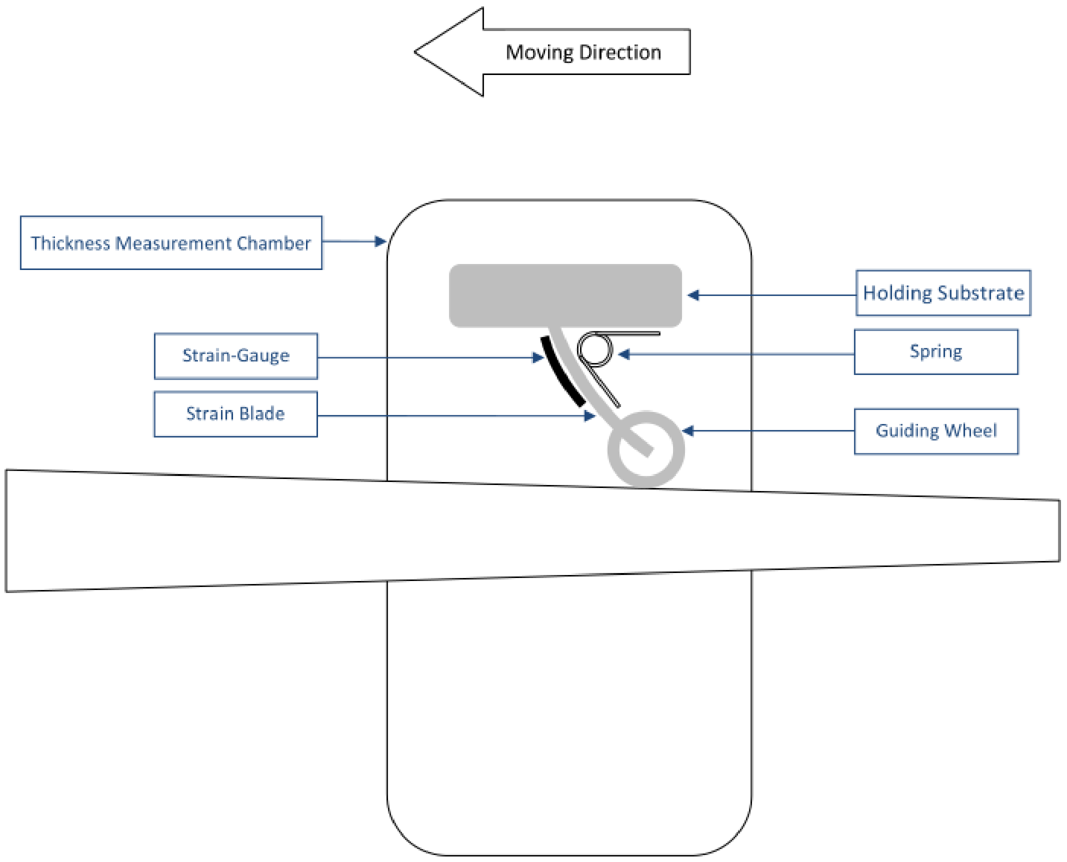


Figure 4.1: Cable Thickness Measurement Instrument (A Typical Conceptual Design)

### 4.2.2 Controller Emulation

To evaluate the implemented controller performance in the current work, it was connected to the process model in MATLAB/Simulink; however, sometimes it is also useful to emulate the controller operation in the simulation environment and check the control program alongside the modeled plant. It is totally clear that an emulated PLC block behaves more similarly to a real controller in comparison with the developed controller based on the MATLAB/Simulink building blocks described in Chapter 2. The benefit of this testing method, and having both emulated controller and simulated process in a single environment, causes the test bench to be simpler and eliminates possible integration errors. The drawback of this test bench is that the emulated controller will be limited to the host computer hardware and operating system performance. For instance, we may not expect hard real time features in an emulated PLC running on Microsoft Windows®[1].

There are some developed translation packages which automatically translate the PLC control program into MATLAB/Simulink software language. Most of these packages apply a set of translation rules that convert the PLC code program into m-files. The m-files then could be integrated with the MATLAB/Simulink process model [7].

### 4.2.3 Supervisory Control and User Interface

In this thesis, the main concern was managing the heat exposure time via chamber motion control. Although system level logic is developed and implemented by introducing PLC Coder™ utility in MATLAB/Simulink, the generated code is not tested effectively. This part of

[1] Microsoft Corporation, www.microsoft.com

the work could be the subject of another project in detail. Additionally, a user interface is needed in the final machine to handle operator communication with the control system. Thanks to the existing standard hardware and software packages in the market, it is easy to develop such a user interface system.

## APPENDIX A

## SYSTEM LEVEL LOGIC CONTROL: STRUCTURE TEXT CODE

```
<?xml version="1.0" encoding="utf-8"?>
<RSLogix5000Content ContainsContext="true" SchemaRevision="1.0" TargetName="Operation"
TargetType="AddOnInstructionDefinition">
<Controller Name="CCU_R2_0" Use="Context">
<AddOnInstructionDefinitions>
<AddOnInstructionDefinition Name="Operation" Use="Target">
<Parameters>
<Parameter DataType="SINT" Name="ssMethodType" Required="true" Usage="Input" Visible="true"/>
<Parameter DataType="BOOL" Name="PWR" Required="true" Usage="Input" Visible="true"/>
<Parameter DataType="BOOL" Name="RUN" Required="true" Usage="Input" Visible="true"/>
<Parameter DataType="BOOL" Name="EMG" Required="true" Usage="Input" Visible="true"/>
<Parameter DataType="BOOL" Name="ACK" Required="true" Usage="Input" Visible="true"/>
<Parameter DataType="BOOL" Name="STD_BY" Required="true" Usage="Output" Visible="true"/>
<Parameter DataType="BOOL" Name="b_OPR" Required="true" Usage="Output" Visible="true"/>
<Parameter DataType="BOOL" Name="b_STD_BY" Usage="Local" Visible="true"/>
<Parameter DataType="BOOL" Name="b_b_OPR" Usage="Local" Visible="true"/>
<Parameter DataType="SINT" Name="is_active_c2_Operation" Usage="Local" Visible="true"/>
<Parameter DataType="SINT" Name="is_c2_Operation" Usage="Local" Visible="true"/>
<Parameter DataType="DINT" Name="temp1" Usage="Local" Visible="true"/>
<Parameter DataType="DINT" Name="temp2" Usage="Local" Visible="true"/>
<Parameter DataType="DINT" Name="temp3" Usage="Local" Visible="true"/>
<Parameter DataType="DINT" Name="temp4" Usage="Local" Visible="true"/>
<Parameter DataType="DINT" Name="temp5" Usage="Local" Visible="true"/>
<Parameter DataType="DINT" Name="temp6" Usage="Local" Visible="true"/>
<Parameter DataType="DINT" Name="temp7" Usage="Local" Visible="true"/>
</Parameters>
<Routines>
<Routine Name="Logic" Type="ST">
<STContent>
<Line Number="1"><![CDATA[]]></Line>
<Line Number="2"><![CDATA[CASE ssMethodType OF]]></Line>
<Line Number="3"><![CDATA[    2: ]]></Line>
<Line Number="4"><![CDATA[        ]]></Line>
<Line Number="5"><![CDATA[        (* InitializeConditions for Stateflow: '<Root>/Operation Mode
Manager'  *)]]></Line>
<Line Number="6"><![CDATA[        is_active_c2_Operation := 0;]]></Line>
<Line Number="7"><![CDATA[        is_c2_Operation := 0;]]></Line>
<Line Number="8"><![CDATA[        b_STD_BY := 0;]]></Line>
<Line Number="9"><![CDATA[        b_b_OPR := 0;]]></Line>
<Line Number="10"><![CDATA[        ]]></Line>
<Line Number="11"><![CDATA[        ]]></Line>
<Line Number="12"><![CDATA[    3: ]]></Line>
<Line Number="13"><![CDATA[        ]]></Line>
<Line Number="14"><![CDATA[        (* Stateflow: '<Root>/Operation Mode Manager'
incorporates:]]></Line>
<Line Number="15"><![CDATA[         *  Inport: '<Root>/ACK']]></Line>
<Line Number="16"><![CDATA[         *  Inport: '<Root>/EMG']]></Line>
<Line Number="17"><![CDATA[         *  Inport: '<Root>/PWR']]></Line>
<Line Number="18"><![CDATA[         *  Inport: '<Root>/RUN']]></Line>
<Line Number="19"><![CDATA[          *)]]></Line>
<Line Number="20"><![CDATA[        (* Gateway: Operation Mode Manager *)]]></Line>
<Line Number="21"><![CDATA[        (* During: Operation Mode Manager *)]]></Line>
<Line Number="22"><![CDATA[        IF is_active_c2_Operation = 0 THEN ]]></Line>
<Line Number="23"><![CDATA[            (* Entry: Operation Mode Manager *)]]></Line>
<Line Number="24"><![CDATA[            is_active_c2_Operation := 1;]]></Line>
<Line Number="25"><![CDATA[            (* Transition: '<S1>:12' *)]]></Line>
<Line Number="26"><![CDATA[            (* Entry 'Off': '<S1>:4' *)]]></Line>
<Line Number="27"><![CDATA[            is_c2_Operation := 2;]]></Line>
<Line Number="28"><![CDATA[        ELSE ]]></Line>
<Line Number="29"><![CDATA[            CASE is_c2_Operation OF]]></Line>
<Line Number="30"><![CDATA[                1: ]]></Line>
<Line Number="31"><![CDATA[                    (* During 'Emergency': '<S1>:3' *)]]></Line>
<Line Number="32"><![CDATA[                    IF ACK THEN ]]></Line>
<Line Number="33"><![CDATA[                        temp1 := 1;]]></Line>
<Line Number="34"><![CDATA[                    ELSE ]]></Line>
<Line Number="35"><![CDATA[                        temp1 := 0;]]></Line>
<Line Number="36"><![CDATA[                    END_IF;]]></Line>
<Line Number="37"><![CDATA[                    IF temp1 = 1 THEN ]]></Line>
<Line Number="38"><![CDATA[                        (* Transition: '<S1>:10' *)]]></Line>
<Line Number="39"><![CDATA[                        (* Exit 'Emergency': '<S1>:3' *)]]></Line>
```

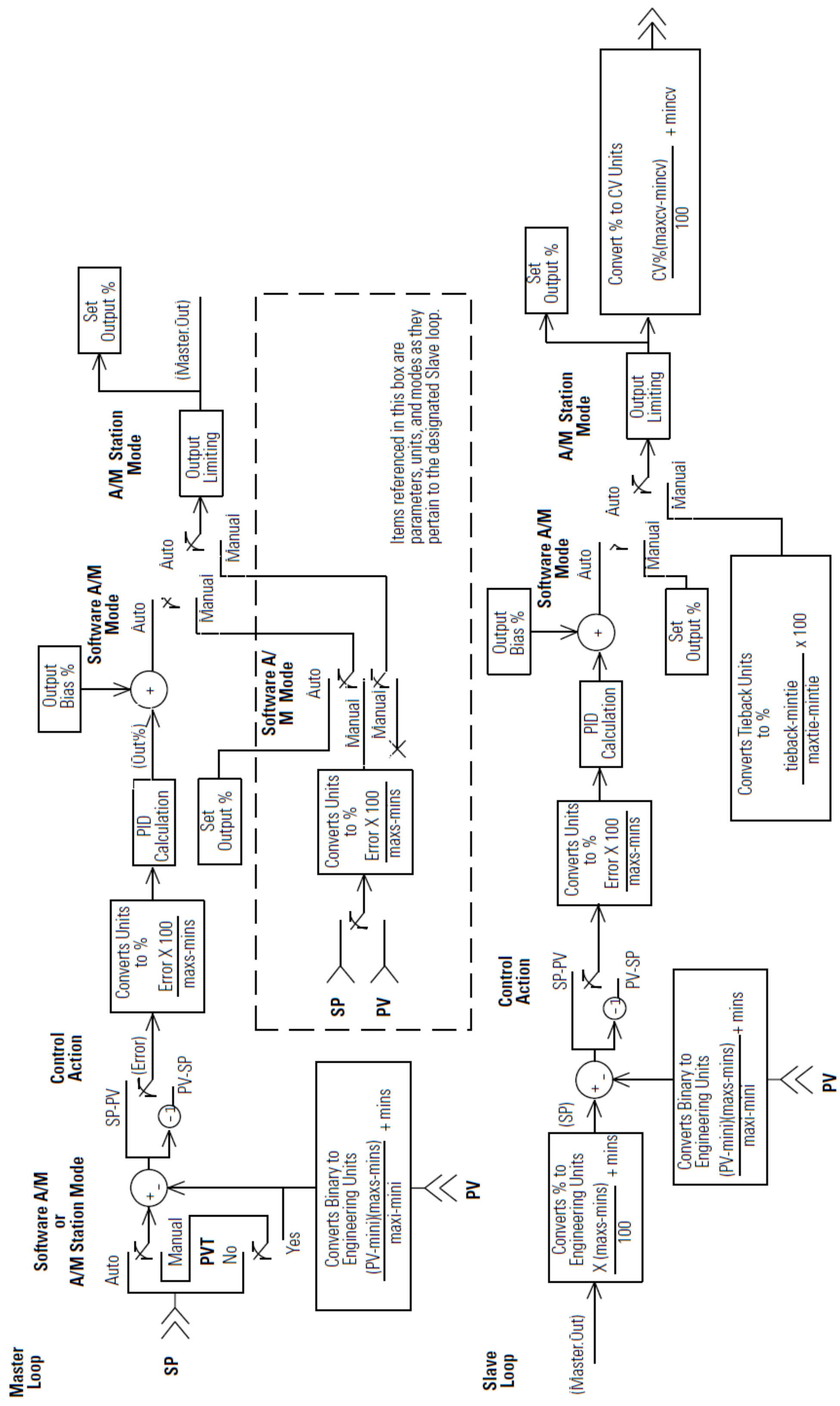
```
<Line Number="40"><![CDATA[                        (* Entry 'Off': '<S1>:4' *)]]></Line>
<Line Number="41"><![CDATA[                        is_c2_Operation := 2;]]></Line>
<Line Number="42"><![CDATA[                    ELSE ]]></Line>
<Line Number="43"><![CDATA[                        (* Chamber stops immediately.]]></Line>
<Line Number="44"><![CDATA[                         Heat Gun is turned off immediately.]]></Line>
<Line Number="45"><![CDATA[                         Alarm appeares.]]></Line>
<Line Number="46"><![CDATA[                         Operator should inspect the system]]></Line>
<Line Number="47"><![CDATA[                         and acknowledge the alarm. *)]]></Line>
<Line Number="48"><![CDATA[                        b_STD_BY := 0;]]></Line>
<Line Number="49"><![CDATA[                        b_b_OPR := 0;]]></Line>
<Line Number="50"><![CDATA[                    END_IF;]]></Line>
<Line Number="51"><![CDATA[                2: ]]></Line>
<Line Number="52"><![CDATA[                    (* During 'Off': '<S1>:4' *)]]></Line>
<Line Number="53"><![CDATA[                    IF PWR THEN ]]></Line>
<Line Number="54"><![CDATA[                        temp2 := 1;]]></Line>
<Line Number="55"><![CDATA[                    ELSE ]]></Line>
<Line Number="56"><![CDATA[                        temp2 := 0;]]></Line>
<Line Number="57"><![CDATA[                    END_IF;]]></Line>
<Line Number="58"><![CDATA[                    IF temp2 = 1 THEN ]]></Line>
<Line Number="59"><![CDATA[                        (* Transition: '<S1>:8' *)]]></Line>
<Line Number="60"><![CDATA[                        (* Exit 'Off': '<S1>:4' *)]]></Line>
<Line Number="61"><![CDATA[                        (* Entry 'Stand_By': '<S1>:1' *)]]></Line>
<Line Number="62"><![CDATA[                        is_c2_Operation := 4;]]></Line>
<Line Number="63"><![CDATA[                    ELSE ]]></Line>
<Line Number="64"><![CDATA[                        (* All electrical/mechanical]]></Line>
<Line Number="65"><![CDATA[                         sub-systems are turned off. *)]]></Line>
<Line Number="66"><![CDATA[                        b_STD_BY := 0;]]></Line>
<Line Number="67"><![CDATA[                        b_b_OPR := 0;]]></Line>
<Line Number="68"><![CDATA[                    END_IF;]]></Line>
<Line Number="69"><![CDATA[                3: ]]></Line>
<Line Number="70"><![CDATA[                    (* During 'Operation': '<S1>:2' *)]]></Line>
<Line Number="71"><![CDATA[                    IF RUN THEN ]]></Line>
<Line Number="72"><![CDATA[                        temp3 := 1;]]></Line>
<Line Number="73"><![CDATA[                    ELSE ]]></Line>
<Line Number="74"><![CDATA[                        temp3 := 0;]]></Line>
<Line Number="75"><![CDATA[                    END_IF;]]></Line>
<Line Number="76"><![CDATA[                    IF temp3 = 0 THEN ]]></Line>
<Line Number="77"><![CDATA[                        (* Transition: '<S1>:7' *)]]></Line>
<Line Number="78"><![CDATA[                        (* Exit 'Operation': '<S1>:2' *)]]></Line>
<Line Number="79"><![CDATA[                        (* Entry 'Stand_By': '<S1>:1' *)]]></Line>
<Line Number="80"><![CDATA[                        is_c2_Operation := 4;]]></Line>
<Line Number="81"><![CDATA[                    ELSE ]]></Line>
<Line Number="82"><![CDATA[                        IF EMG THEN ]]></Line>
<Line Number="83"><![CDATA[                            temp4 := 1;]]></Line>
<Line Number="84"><![CDATA[                        ELSE ]]></Line>
<Line Number="85"><![CDATA[                            temp4 := 0;]]></Line>
<Line Number="86"><![CDATA[                        END_IF;]]></Line>
<Line Number="87"><![CDATA[                        IF temp4 = 1 THEN ]]></Line>
<Line Number="88"><![CDATA[                            (* Transition: '<S1>:11' *)]]></Line>
<Line Number="89"><![CDATA[                            (* Exit 'Operation': '<S1>:2' *)]]></Line>
<Line Number="90"><![CDATA[                            (* Entry 'Emergency': '<S1>:3'
*)]]></Line>
<Line Number="91"><![CDATA[                            is_c2_Operation := 1;]]></Line>
<Line Number="92"><![CDATA[                        ELSE ]]></Line>
<Line Number="93"><![CDATA[                            (* Chamber starts to move in the]]></Line>
<Line Number="94"><![CDATA[                             shrinking direction until the
end]]></Line>
<Line Number="95"><![CDATA[                             of the path. *)]]></Line>
<Line Number="96"><![CDATA[                            b_STD_BY := 0;]]></Line>
<Line Number="97"><![CDATA[                            b_b_OPR := 1;]]></Line>
<Line Number="98"><![CDATA[                        END_IF;]]></Line>
<Line Number="99"><![CDATA[                    END_IF;]]></Line>
<Line Number="100"><![CDATA[                4: ]]></Line>
<Line Number="101"><![CDATA[                    (* During 'Stand_By': '<S1>:1' *)]]></Line>
<Line Number="102"><![CDATA[                    IF RUN THEN ]]></Line>
<Line Number="103"><![CDATA[                        temp5 := 1;]]></Line>
<Line Number="104"><![CDATA[                    ELSE ]]></Line>
<Line Number="105"><![CDATA[                        temp5 := 0;]]></Line>
<Line Number="106"><![CDATA[                    END_IF;]]></Line>
<Line Number="107"><![CDATA[                    IF temp5 = 1 THEN ]]></Line>
<Line Number="108"><![CDATA[                        (* Transition: '<S1>:6' *)]]></Line>
```

```
<Line Number="109"><![CDATA[                        (* Exit 'Stand_By': '<S1>:1' *)]]></Line>
<Line Number="110"><![CDATA[                        (* Entry 'Operation': '<S1>:2' *)]]></Line>
<Line Number="111"><![CDATA[                        is_c2_Operation := 3;]]></Line>
<Line Number="112"><![CDATA[                    ELSE ]]></Line>
<Line Number="113"><![CDATA[                        IF EMG THEN ]]></Line>
<Line Number="114"><![CDATA[                            temp6 := 1;]]></Line>
<Line Number="115"><![CDATA[                        ELSE ]]></Line>
<Line Number="116"><![CDATA[                            temp6 := 0;]]></Line>
<Line Number="117"><![CDATA[                        END_IF;]]></Line>
<Line Number="118"><![CDATA[                        IF temp6 = 1 THEN ]]></Line>
<Line Number="119"><![CDATA[                            (* Transition: '<S1>:9' *)]]></Line>
<Line Number="120"><![CDATA[                            (* Exit 'Stand_By': '<S1>:1' *)]]></Line>
<Line Number="121"><![CDATA[                            (* Entry 'Emergency': '<S1>:3'
*)]]></Line>
<Line Number="122"><![CDATA[                            is_c2_Operation := 1;]]></Line>
<Line Number="123"><![CDATA[                        ELSE ]]></Line>
<Line Number="124"><![CDATA[                            IF PWR THEN ]]></Line>
<Line Number="125"><![CDATA[                                temp7 := 1;]]></Line>
<Line Number="126"><![CDATA[                            ELSE ]]></Line>
<Line Number="127"><![CDATA[                                temp7 := 0;]]></Line>
<Line Number="128"><![CDATA[                            END_IF;]]></Line>
<Line Number="129"><![CDATA[                            IF temp7 = 0 THEN ]]></Line>
<Line Number="130"><![CDATA[                                (* Transition: '<S1>:5' *)]]></Line>
<Line Number="131"><![CDATA[                                (* Exit 'Stand_By': '<S1>:1'
*)]]></Line>
<Line Number="132"><![CDATA[                                (* Entry 'Off': '<S1>:4' *)]]></Line>
<Line Number="133"><![CDATA[                                is_c2_Operation := 2;]]></Line>
<Line Number="134"><![CDATA[                            ELSE ]]></Line>
<Line Number="135"><![CDATA[                                (* Chamber waits in the park
position]]></Line>
<Line Number="136"><![CDATA[                                 and ready to move. If it is not in
the]]></Line>
<Line Number="137"><![CDATA[                                 park position, it moves back
there.]]></Line>
<Line Number="138"><![CDATA[                                 Heat Gun is turned on to preheat the
chamber.]]></Line>
<Line Number="139"><![CDATA[                                 System user-interface starts to
work]]></Line>
<Line Number="140"><![CDATA[                                 and displays the status of the
machine. *)]]></Line>
<Line Number="141"><![CDATA[                                b_STD_BY := 1;]]></Line>
<Line Number="142"><![CDATA[                                b_b_OPR := 0;]]></Line>
<Line Number="143"><![CDATA[                            END_IF;]]></Line>
<Line Number="144"><![CDATA[                        END_IF;]]></Line>
<Line Number="145"><![CDATA[                    END_IF;]]></Line>
<Line Number="146"><![CDATA[                ELSE]]></Line>
<Line Number="147"><![CDATA[                    (* Transition: '<S1>:12' *)]]></Line>
<Line Number="148"><![CDATA[                    (* Entry 'Off': '<S1>:4' *)]]></Line>
<Line Number="149"><![CDATA[                    is_c2_Operation := 2;]]></Line>
<Line Number="150"><![CDATA[            END_CASE;]]></Line>
<Line Number="151"><![CDATA[        END_IF;]]></Line>
<Line Number="152"><![CDATA[        ]]></Line>
<Line Number="153"><![CDATA[        ]]></Line>
<Line Number="154"><![CDATA[        (* Outport: '<Root>/STD_BY'  *)]]></Line>
<Line Number="155"><![CDATA[        STD_BY := b_STD_BY;]]></Line>
<Line Number="156"><![CDATA[        ]]></Line>
<Line Number="157"><![CDATA[        (* Outport: '<Root>/OPR'  *)]]></Line>
<Line Number="158"><![CDATA[        b_OPR := b_b_OPR;]]></Line>
<Line Number="159"><![CDATA[        ]]></Line>
<Line Number="160"><![CDATA[END_CASE;]]></Line>
</STContent>
</Routine>
</Routines>
</AddOnInstructionDefinition>
</AddOnInstructionDefinitions>
</Controller>
</RSLogix5000Content>
```

## APPENDIX B

## MASTER/SLAVE PID LOOPS: RSLOGIX 5000 INSTRUCTION BLOCK DIAGRAM

(Captured from “Logix5000 Controllers General Instructions”, Rockwell Automation)



Master Loop
SP
Software A/M or A/M Station Mode
Auto
Manual
PVT
No
Yes
Control Action
SP-PV
PV-SP
(Error)
Converts Binary to Engineering Units
(PV-mini)(maxs-mins) / maxi-mini + mins
PV
Converts Units to %
Error X 100 / maxs-mins
PID Calculation
(Out%)
Output Bias %
Software A/M Mode
Set Output %
A/M Station Mode
Output Limiting
(Master.Out)
Software A/ M Mode
Converts Units to %
Error X 100 / maxs-mins
Items referenced in this box are parameters, units, and modes as they pertain to the designated Slave loop.
Slave Loop
(Master.Out)
Converts % to Engineering Units
X (maxs-mins) / 100 + mins
(SP)
Converts Binary to Engineering Units
(PV-mini)(maxs-mins) / maxi-mini + mins
Converts Tieback Units to %
tieback-mintie / maxtie-mintie x 100
Convert % to CV Units
CV%(maxcv-mincv) / 100 + mincv

## REFERENCES

[1] Wikipedia contributors. "Cable harness." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 23 Nov. 2013. Web. 1 Dec. 2013.

[2] Wikipedia contributors. "Heat-shrink tubing." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 27 Nov. 2013. Web. 1 Dec. 2013.

[3] "CableOrganizer.com's Learning Center - A Source of More Information." CableOrganizer.com - The Best Prices on Wire Management Solutions. Cableorganizer, 2002. Web. 1 Dec 2013. <http://www.cableorganizer.com/learning-center>.

[4] Wikipedia contributors. "Model-based design." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 15 Nov. 2013. Web. 1 Dec. 2013.

[5] Persin, Stojan, Tovornik, Boris, Muskinja, Nenad. "OPC-driven Data Exchange between MATLAB and PLC-controlled System." Int. J. Engng Ed. Vol. 19, No. 4 (2003): 586–592. PDF.

[6] Lieping, Zhang, Aiqun, Zeng, Yunsheng, Zhang. "On Remote Real-time Communication between MATLAB and PLC Based on OPC Technology." Proceedings of the 26th Chinese Control Conference, July 26-31, 2007:545-548. PDF.

[7] Martins, João, Lima, Celson, Martínez, Herminio, Grau, Antoni. "PLC Control and Matlab/Simulink Simulations – A Translation Approach, Matlab - Modelling, Programming and Simulations." Emilson Pereira Leite (Ed.), ISBN: 978-953-307-125-1, InTech, 2010. Web. <http://www.intechopen.com/books/matlab-modelling-programming-and-simulations/plc-control-and-matlab-simulink-simulations-a-translation-approach>

[8] Moura, Raimundo, Affonso Guedes, Luiz. "Control and Plant Modeling for Manufacturing Systems using Basic Statecharts, Programmable Logic Controller." Luiz Affonso Guedes (Ed.), ISBN: 978-953-7619-63-3, InTech, 2010. Web. <http://www.intechopen.com/books/programmable-logic-controller/control-and-plant-modeling-for-manufacturing-systems-using-basic-statecharts>

[9] Han, Kwan Hee. "Object-Oriented Modeling, Simulation and Automatic Generation of PLC Ladder Logic, Programmable Logic Controller." Luiz Affonso Guedes (Ed.), ISBN: 978-953-7619-63-3, InTech, 2010. Web. <http://www.intechopen.com/books/programmable-logic-controller/object-oriented-modeling-simulation-and-automatic-generation-of-plc-ladder-logic>

[10] Piedrafita, Ramón, Villarroel, José  Luis. "The Java Based Programmable Logic Controller. New Techniques in Control and Supervision of a Flexible Manufacturing Cell, Programmable Logic Controller." Luiz Affonso Guedes (Ed.), ISBN: 978-953-7619-63-3,

InTech, 2010. Web. <http://www.intechopen.com/books/programmable-logic-controller/the-java-based-programmable-logic-controller-new-techniques-in-control-and-supervision-of-a-flexible>

[11] Wan, Yan. "Systems Modeling and Simulation-Lecture Notes." Department of Electrical Engineering, University of North Texas, Fall 2013. PDF.

[12] Sargent, Robert. "Validation and Verification of Simulation Models." Proceedings of the 2004 Winter Simulation Conference (2004): 17-28. PDF.

[13] "Products & Services\Simscape." MathWorks. MathWorks, Web. 8 Feb. 2014. <http://www.mathworks.com/products/simscape/>.

[14] Abd Elhamid, Ahmed S. "Cascade Control System of Direct Current Motor." World Applied Sciences Journal 18-12 (2012): 1680-1688. PDF.

[15] Bhavina, Rathod, Jamliya, Nitesh, Vashishtha, Keerti. "Cascade Control of DC Motor with Advance Controller." Proceedings of SARC-IRAJ International Conference, 14th July (2013): 36-38. PDF.

[16] Wikipedia contributors. "PID controller." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 7 Feb. 2014. Web. 16 Feb. 2014.

[17] "Simulink® PLC Coder™ User's Guide." R2013b. MathWorks® Inc., Sep. 2013. Ebook.

[18] Johnson, Michael A., Moradi, Mohammad H. "PID Control New Identification and Design Methods", Springer, ISBN-10:1-85233-702-8, 2005. Ebook.

[19] Wikipedia contributors. "Automation." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 17 Mar. 2014. Web. 24 Mar. 2014.

[20] "1769-L32E, -L35E CompactLogix™ Controller", Rockwell Automation (March 2004), Publication 1769-IN020B-EN-P-March 2004.

[21] "Compact™ 24V dc Sink/Source Input Module", Rockwell Automation (June 2000), Publication 1769-IN007B-EN-P.

[22] "Compact™ 1769-OW8 AC/DC Relay Output Module", Rockwell Automation (January 1999), Publication 1769-5.2.

[23] "Compact I/O Combination Fast Analog I/O Module", Rockwell Automation (October 2008), Publication 1769-UM019A-EN-P-October 2008.

[24] "Logix5000 Controllers General Instructions", Rockwell Automation (October 2009), Publication 1756-RM003L-EN-P - October 2009.

[25] Groot, Peter de. "Optical thickness measurement of substrates using a transmitted wavefront test at two wavelengths to average out multiple reflection errors." Proceedings of SPIE Vol. 4777 (2002): 177–183. PDF.

[26] Sastikumar, D., Mohamad Jaffer, M.Jamal. "Measurement of Diameters of Wires Using Laser and Optical Fiber." IEEE Instrumentation and Measurement Technology Conference, St. Paul, Minnesota, May 18-20 (1998): 961–965. PDF.