AUTONOMOUS HOME COFFEE MAKER USING FUZZY LOGIC

A thesis submitted in partial fulfillment of the requirements for the degree of
Maстер of Science in Engineering

By

NICHOLAS JASON VANDERPOOL
B.S. Wright State University 2003

2007
Wright State University
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Nicholas Vanderpool ENTITLED Development of an Intelligent Autonomous Coffee Maker BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering.

_______________________
Kuldip S. Rattan, Ph.D.
Thesis Director

_______________________
Fred D. Garber, Ph.D.
Department Chair

Committee on
Final Examination

__________________________
Kuldip S. Rattan, Ph.D.

__________________________
Marian K. Kazimierczuk, Ph.D.

__________________________
Raymond E. Siferd, Ph.D.

__________________________
Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies
Abstract

Vanderpool, Nicholas. M.S. Egr., Department of Electrical Engineering, Wright State University, 2007. Autonomous Home or Office Coffee Maker Using Intelligent Control Methodology

Home coffee machines have improved over time with the addition of features such as self-activating timers and crude temperature controlled heating elements. However, home coffee brewer development has stagnated and has yet to achieve full automation status. Present coffee machines leave the user burdened with the mundane and messy task of preparing the coffee maker every time. Also, should the user decide to prepare less than an entire pot, he/she must estimate the correct amount of water and coffee to add. This can lead to a weak or strong brew, giving undesirable results. In order to make another pot, the user is then forced to clean the used coffee grounds and remove any unused coffee.

The objective of this thesis is to design and construct a functional, fully automated coffee maker. The coffee maker, with its need for user input, varying grounds and water volume necessity, and sequential method of operation, is a prime candidate for fuzzy control implementation. The main focus of this project is a controller, which is implemented on a standard 486 computer running Linux Red Hat 7. The system is included in this project, but only as a means to test the functionality and efficiency of the controller. User input includes the amount of coffee desired, the strength of that coffee, and the ratio of regular to decaffeinated. These input variables can be effectively and
easily defined by fuzzy membership functions. Also, when the coffee maker is not making coffee, the processor monitors water temperature for the next time it is needed and performs any necessary cleaning.

Results demonstrate that the water and coffee temperatures can be kept within a tolerance of three degrees Fahrenheit. The open-loop coffee-dispensing controller is validated by expert evaluation since taste is the deciding factor. The efficiency of the water controller is determined by the accuracy of the volume of water obtained. Overshoot and steady-state-error are kept within strict tolerances for this specific application.
ACKNOWLEDGMENTS

I would like to extend my special thanks to my thesis advisor, Dr. Kuldip Rattan, for his continual guidance and understanding of my intense desire to gloss over mathematical details in engineering practices, and therefore directed me to fuzzy control, an area of study tailored to people like myself. His help, along with his ability to let me run with the project as I saw fit gave me the controlled freedom necessary for the successful completion of this thesis. Many thanks also go to Dr. Marian Kazimierczuk and Dr. Raymond Siferd for taking the time to review my thesis and take part in my final examination committee.

My parents deserve a special mention. Thanks to my Mom, who always believed that I could cobble together anything that I wanted. Also, thanks to my Dad, who ensured that I learn the basics of wiring, plumbing, carpentry, and construction and showed me the technical side not covered in college.

I would also like to thank several people in the Electrical Engineering Department that were indispensable in a multitude of ways. To Barry Woods, who provided me with numerous great conversations about appliance innovation and helped me locate spare parts on occasion. To Vickie Slone, who assisted me in the nightmarish world of college paperwork procedures and shares my distaste for ranch style homes. To Dr. Garber, who was kind enough to give me a chance in the graduate program. To Mike Meyers, a co-worker, advisor, instructor, assistant, and good friend who was willing to listen to me endlessly complain about the complications of this thesis when most others would have
given up. Finally, I’d like to thank Sara Hohne, possibly the best employer I’ll ever have and the person who taught me that the workplace can actually be enjoyable. For nearly a decade, Sara helped me, looked after me, and when needed, played the role of advisor, creditor, teacher, mom, nurse during a painful head injury, and always a good friend.

Every person named above made my adult academic career bearable, and even pleasurable as time went on. They were friendly, tolerant of me when I hung around a bit too long, and understanding. To each of them I offer a profound debt of gratitude for their help and hospitality. Without them, I’d be nothing but an angry, anti-social technician still suffering from math anxiety. A special thanks to everyone.

Nicholas J. Vanderpool

Wright State University

Dayton, Ohio

July 1, 2007
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Background</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Coffee Machines – A Brief History</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Fuzzy Logic</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Fuzzy Logic Controller (FLC)</td>
<td>7</td>
</tr>
<tr>
<td>3. Automated Coffee System</td>
<td>10</td>
</tr>
<tr>
<td>3.1 Control System</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Fuzzy Logic Rule Base</td>
<td>10</td>
</tr>
<tr>
<td>3.3 Electrical System</td>
<td>16</td>
</tr>
<tr>
<td>3.3.1 Overview of the Computer Control System</td>
<td>17</td>
</tr>
<tr>
<td>3.3.2 Overview of the Plant</td>
<td>24</td>
</tr>
<tr>
<td>3.3.2.1 Ground Coffee Dispenser</td>
<td>27</td>
</tr>
<tr>
<td>3.3.2.2 The Central Brewing Chamber</td>
<td>30</td>
</tr>
<tr>
<td>3.3.2.3 The Main Heating Tank Assembly</td>
<td>33</td>
</tr>
<tr>
<td>3.3.2.4 Carafe Heater</td>
<td>35</td>
</tr>
<tr>
<td>3.3.2.5 Carafe and Pump</td>
<td>36</td>
</tr>
<tr>
<td>3.3.3 Temperature Sensors</td>
<td>38</td>
</tr>
<tr>
<td>3.3.4 Pressure Sensors</td>
<td>42</td>
</tr>
<tr>
<td>3.3.5 User Input</td>
<td>44</td>
</tr>
<tr>
<td>4. Results/Discussion</td>
<td>46</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A general fuzzy logic control diagram</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Input Fuzzy Logic Membership Function</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Triangular memberships for the number of cups selected</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>Triangular memberships for the strength of the coffee</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>Triangular memberships for the ratio of regular to decaf coffee</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>Block diagram of the user input inference system</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>The control tables for both heating systems</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>A block diagram of the entire system</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>Interface circuitry between the control computer and the plant</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>The I/O readout box</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>The I/O box used for interfacing between the computer plant</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>The schematic of the digital control board</td>
<td>20</td>
</tr>
<tr>
<td>13</td>
<td>The digital control board</td>
<td>21</td>
</tr>
<tr>
<td>14</td>
<td>The flowchart of the control program main loop</td>
<td>23</td>
</tr>
<tr>
<td>15</td>
<td>Overview of the coffee maker plant, (front view)</td>
<td>25</td>
</tr>
<tr>
<td>16</td>
<td>Plant schematics</td>
<td>26</td>
</tr>
<tr>
<td>17</td>
<td>The heating element power solenoids</td>
<td>27</td>
</tr>
<tr>
<td>18</td>
<td>The mechanical layout of the coffee dispensers</td>
<td>28</td>
</tr>
<tr>
<td>19</td>
<td>The Regular Coffee Dispenser</td>
<td>29</td>
</tr>
<tr>
<td>20</td>
<td>The mechanical layout of the central brewing chamber hardware</td>
<td>30</td>
</tr>
</tbody>
</table>
**LIST OF TABLES**

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Coffee brewing control chart</td>
<td>3</td>
</tr>
<tr>
<td>2: The output rule matrix for the coffee ground dispensing system</td>
<td>14</td>
</tr>
</tbody>
</table>
1. Introduction

Intelligent control has existed for several decades and proven to be an effective and reasonably simple way to control electromechanical devices. Japan has increased its use of fuzzy logic for the past two decades. However, intelligent control is still somewhat of a rarity in American home appliance design. One of the most recent candidates for this method of control is the washing machine, using standard membership functions to measure variables such as the soil consistency of the water. Other appliances include heating devices such as the rice cooker, varying the temperature as needed by means of intelligent control.

Coffee preparation is a mundane, daily task that many people perform multiple times a day. Though the steps to make coffee are relatively easy to perform, they are frequently messy and unnecessary. The technological level needed to automate this process effectively and in a cost efficient manner is already well established. However, the work of design and testing must still be performed to alleviate society of another menial task.

The home and office coffee maker is an ideal candidate for intelligent control since it requires both user input as well as continuous state monitoring. This, combined with minimum hardware can easily handle the mundane steps needed to make coffee. This exercise in automation requires the timing and control of a motor, electromechanical solenoids, heating elements, and an electric liquid pump. These components are activated in a sequence with varied time durations or power levels based on selected input
from a user as well as feedback sensors. The coffee machine then performs the necessary actions to clean itself and prepare for another usage.

Many existing coffee makers on the market today already possess a certain level of automation. A common feature is the timer, a simple circuit designed to activate the heating element and ultimately prepare the coffee. The heater then remains on for a certain amount of time, or indefinitely, depending on the model. While this is technically automation, it still requires a great deal of work from the user to set up and clean after use. Bunn, a better-known manufacturer of coffee machines, has an automatic water-refilling feature, which consists of an inlet attached to a water supply line. This alleviates the user of one introductory step, namely, filling the tank with water. However, no current home coffee maker handles the problem of coffee grounds. Grounds must be loaded into the brewing chamber before using and requires a filter to hold them. The coffee grounds are fairly small and lightweight, so they can be difficult to transport and measure for accuracy by mechanical means. Also, the volume of grounds used depends on the amount of coffee and strength desired. These two factors can be easily mapped into fuzzy rules based on expert information. Criteria to base the correct amount of coffee on comes from research already performed by the Bunn Company. In the development of their brand of coffee makers, they’ve created a chart seen in Table 1. This chart explains the relationship between the strength of brewed coffee versus the extraction, the amount of coffee material removed from the grounds during brewing. This is dictated mainly by the amount of coffee versus the amount of water passed through the grounds.
There are several steps to automating simple drip-style coffee. First, if the carafe isn’t empty from the last usage, this unwanted coffee must be drained. Then, a new filter must be loaded into the brewing chamber. This is followed by fresh coffee grounds loaded to the filter assembly in the correct amount. The approximate water volume then must be added to the coffee machines reserve tank. These events can be performed one by one or simultaneously since they are merely loading procedures and are independent of one another. The quickness of these steps being performed hinges on efficient programming and use of limited I/O lines.
When the coffee maker is not in operation, it must still perform the constant task of keeping the water temperature at the desired level. The water temperature must be read in constant intervals to maintain the desired temperature. Therefore, the computer devotes all of its processing time to reading this digital input and performing a continuous temperature calculation and control system judgment. This calculation controls the amount of heat affecting the water in the reserve tank and keeps it in a perpetual state of readiness for the next brewing cycle, an important feature for quick use.
2. Background

2.1 Coffee Machines – A Brief History

More than one third of the world’s population drinks coffee today. Coffee’s popularity is not confined to any one country, continent, or culture. It is found in the Middle East, where the taste for coffee originated and where the way in which the drink is enjoyed has remained unchanged for hundreds of years. It is drunk in Africa, where modern coffee cultivation and commerce coexist with ancient customs. In Europe, the coffeehouse still reigns as a social hub. In the United States, one of the highest forms of praise one can bestow upon a dining establishment or a host or hostess is that they serve “a good cup of coffee.” In Latin America, this prime commodity has not surprisingly become an integral part of daily life. Even in parts of Asia, a region of the world generally characterized as tea totaling, coffee’s popularity is firmly entrenched or rapidly growing [2].

Coffee makers were first introduced around 500 A.D. and have evolved greatly. Around 1800, the first drip style coffee pot was invented, relying on a fire or stove to heat the device. Since then, several advances such as electric heat and electronic start timers have been introduced, but the underlying method and chemistry has stayed the same. While effective, the control has been limited to electrical safety practices and basic functionality. Accuracy in the coffee making in terms of flavor and amount rested completely on the experience and skill of the user.

Eventually, fully automated machines were introduced in the form of vending machines. These delivered a cup at a time and gave a large selection of coffee, but these
devices aren’t economical or appropriate for home use. Therefore, a more suitable design is required.

2.2 Fuzzy Logic

In the 1960’s, Zadeh published his work on a multi-value rule based system, differing from traditional logic, which contains only two possible elements, true and false. Fuzzy logic allows for varying degrees of truth. The new system can have any number of elements within the set, as well as have partial membership to that set. The most fruitful application of fuzzy logic has been in the field of control. Ever since its inception, fuzzy logic control (FLC) has been applied to a variety of complex, non-linear systems. Japan has been a leader in the implementation of FLC in industry with a number of FLC home appliances, such as washing machines, and rice cookers. Although capable of controlling such systems, it can be applied to any system requiring of varying complexity and specifications. An advantage FLC has over conventional controllers is its ability to assimilate human expert knowledge in the form of language. The FLC has the ability to describe a system linguistically, based on a set of rules with language-like descriptors such as ‘if’ and ‘then’, creating a more intuitive design approach.

FLC can be viewed as a real-time expert system and non-linear controller. As a real-time expert system, it is capable of accepting linguistic rules from an expert, typically a person or an operator, and evaluates these rules in real time. As a non-linear controller, FLC is capable of creating a piece-wise linear control curve that can closely follow a non-linear control curve.
2.3 Fuzzy Logic Controller (FLC)

Three stages make up fuzzy logic: fuzzification, inference, and defuzzification, shown in Figure 1. Before a signal can be controlled by a fuzzy logic controller, it must first be fuzzified. This portion consists of determining what memberships apply to the input in question which is then passed to the inference engine. The inference engine evaluates an established rule-base and applies the appropriate rule to the inputs before passing the result on to the final stage, the defuzzifier. The defuzzifier converts the fuzzy value back into a crisp, finalizing the process.

![Figure 1. A general fuzzy logic control diagram.](image)

Fuzzification is based on the original theory proposed by Zadeh, where a crisp value is converted into a fuzzy value via the input fuzzy set (Zadeh 1992). The input fuzzy set is based on a membership function, which can have various shapes. The membership function used in this thesis is in a triangular shape because of its simple implementation and adequacy for the control needs of this project.
A typical two set triangular membership function is illustrated in Figure 2. A crisp input is normalized and then fuzzified using a two set input membership function, shown on the triangular function as A and B. The fuzzification produces the degree of membership to each of the fuzzy sets. For this example, the crisp value of 0.4 is fuzzified into two fuzzy sets: set 1 at a 0.7 degree of membership and set 2 at a 0.3 degree of membership.

![Figure 2. Input Fuzzy Logic Membership Function.](image)

This next stage in the process is where rule-base is applied to the fuzzy input. Expert information is stored in the form of rules and can be created in a variety of ways. A typical rule in linguistic form would appear as follow:

If the **temperature** is *low* and the **temperature rate** is *low*, then *medium voltage* occurs.
The rule-base contains a list of all of the necessary rules and the inference engine applies the appropriate rules to the fuzzy inputs to produce a fuzzy output. The rules can also be represented as a matrix of values such as in Table 2.

The third and final stage of the FLC is defuzzification. The opposite of fuzzification, the fuzzy value is converted back to a crisp value. The method used in this thesis is the weighted average method.
3. Automated Coffee System

3.1 Control System

The coffeemaker control system is based on several levels of fuzzy logic methodology. In this application, fuzzy logic is best viewed as a decision-making controller based on rules determined by an expert. Since it is a transfer function of the coffee making process derived from the inherent physical properties of the system, which can be extremely difficult or impossible to calculate in some instances, it is easier to simply create a table of rules for the controller to follow. This approach is superior because the rules can be expressed in simple English language rather than an equation, making the control scheme easier to understand. This table of rules forms the rule base for the coffee machine controller both during coffee preparation and standby.

3.2 Fuzzy Logic Rule Base

The rule base for the coffee maker is accomplished first by deciding the level of accuracy desired and then filling in the needed scale values to meet the criteria. It was decided that the user can select anywhere from two to twelve cups of coffee. More specifically, values for two, four, six, eight, ten, and twelve cups were selected as the fuzzy sets. Also, the user can select a coffee strength from light, to medium, to dark. A third input was later added to mix a combination of regular coffee grounds to decaffeinated. A system expert who has fifteen years of experience of coffee drinking provided the data needed to fill out the rule matrix. Values for two, six and ten cups were
tested for and based on these values, the rest of the rule matrix was completed using logical evaluation and trial and error.

As stated above, there are three inputs needed from the user before the machine can operate. The number of cups desired, strength of that coffee, and the ratio of regular to decaffeinated must be selected via three potentiometers, mounted on the front face panel. To handle this input, three fuzzy sets were developed. The memberships representing the number of cups can be seen in Figure 3.

![Figure 3. Triangular memberships for the number of cups selected.](image)

As shown, the triangular membership functions are used for simplicity and ease of computation. For the triangular membership function, only the center points are needed and any value between the center points is calculated in the fuzzification code. The same applies to the fuzzy set for the strength of the coffee. Shown in Figure 4, the fuzzy set also adheres to the triangular membership form, but only uses three center points, opposed to the six for the number of cups of coffee.
A third, simpler fuzzy set deals with the ratio input. Since a relationship between regular and decaffeinated coffee is all that must be calculated, this set consists of only two half triangular membership, as shown in Figure 5. This final stage generates output in the form of two time values, $t_1$ for the regular coffee ground hopper activation time, and $t_2$ for the decaffeinated hopper activation time.
These three inference systems, combined with a water pressure system make up the user input fuzzy logic system component of the coffee machine, seen as a block diagram in Figure 6. Three output values are calculated, all in the form of time necessary to operate electro-mechanical solenoids in the plant.

![Figure 6. Block diagram of the user input inference system.](image)

This means that the system in question has four inputs and three outputs. The coffee ground dispensing system leads to a rule matrix that can be seen in Table 2. Notice the slight variation from the standard rule matrix. Here, the values follow a certain pattern, growing larger as more cups of coffee are to be prepared.
Both heater control systems follow a simple look-up table approach. A simplistic form of fuzzy control, the input of the system, the number of cups desired and the water pressure are combined. The look-up table creates a flat-topped control response with no overlapping. No overlapping is allowed because the physical hardware used generates several voltages levels and combining two together would result in an electrical short. The control tables can be seen in Figure 7.

These two tables for the carafe and the tank follow the if-else method. The input is evaluated against a set of known rules, finds the rule that applies, and follows that particular rule. Below is an excerpt from the control software showing these rules.

```plaintext
// boundary definitions
CA = 130; // temperature barrier between 120 and 60 volt setting
CB = 140; // temperature barrier between 60 and 24 volt setting
CC = 155; // temperature barrier between 24 and 12 volt setting
CD = 165; // temperature barrier between 12 volt and OFF setting
TA = 180; // temperature barrier between 120 and 60 volt setting
TB = 190; // temperature barrier between 60 and 24 volt setting
TC = 197; // temperature barrier between 24 and 12 volt setting
TD = 203; // temperature barrier between 12 volt and OFF setting
```
if ( FUZZY.carafe_heat <= CA )
{ carafe_new = 120; }
else if ( ( FUZZY.carafe_heat > CA ) || ( FUZZY.carafe_heat <= CB ) )
{ carafe_new = 60; }
else if ( ( FUZZY.carafe_heat > CB ) || ( FUZZY.carafe_heat <= CC ) )
{ carafe_new = 24; }
else if ( ( FUZZY.carafe_heat > CC ) || ( FUZZY.carafe_heat <= CD ) )
{ carafe_new = 12; }
else if ( FUZZY.carafe_heat > CD )
{ carafe_new = 0; }

// Decipher what membership the tank output should be in.
if ( FUZZY.tank_heat <= TA )
{ tank_new = 120; }
else if ( ( FUZZY.tank_heat > TA ) || ( FUZZY.tank_heat <= TB ) )
{ tank_new = 60; }
else if ( ( FUZZY.tank_heat > TB ) || ( FUZZY.tank_heat <= TC ) )
{ tank_new = 24; }
else if ( ( FUZZY.tank_heat > TC ) || ( FUZZY.tank_heat <= TD ) )
{ tank_new = 12; }
else if ( FUZZY.tank_heat > TD )
{ tank_new = 0; }

These rules equate to the control system table shown in Figure 7

<table>
<thead>
<tr>
<th>Carafe Temperature Control</th>
<th>120 VAC</th>
<th>60 VAC</th>
<th>24 VAC</th>
<th>12 VAC</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>130</td>
<td>140</td>
<td>155</td>
<td>165</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heater Temperature Control</th>
<th>120 VAC</th>
<th>60 VAC</th>
<th>24 VAC</th>
<th>12 VAC</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>180</td>
<td>190</td>
<td>197</td>
<td>203</td>
</tr>
</tbody>
</table>

**Figure 7.** The control tables for both heating systems.
3.3 Electrical System

The design of the electrical system can be effectively broken down into two main categories. The first deals with the computer hardware, section 3.3.1, and covers all schematics up the data input and output lines, just before interfacing with electromechanical components. From then on, everything else is covered in section 3.3.2, the electromechanical system that explains the hardware being controlled. A basic block diagram can be seen in Figure 8.

Figure 8. A block diagram of the entire system.
3.3.1 Overview of the Computer Control System

The code, (see Appendix B) for the fuzzy logic controller was written in C mainly due to the ease of using the parallel port. The interface circuitry to control the plant from the computers two parallel ports can be seen in Figure 9. For safety of the computer hardware, the 74HCT367 hex buffer chip is used, buffering the parallel ports from any damage that can be caused by faulty wiring or circuitry. Each data line can be used to control a device, or input information in decimal form. Also, each data port, input or output, is connected to an LED with a 550 Ohm resistor wired in series to limit current. This is incorporated in an interface box for easy viewing during operation.

Figure 9. Interface circuitry between the control computer and the plant.

The parallel port interface circuitry above is contained in a project case with LED display mounted in the front. This is currently mounted on the front left of the plant for
ease of monitoring what devices are currently on. This component shows the activity at all data, control, and status lines for both P1 and P2. Though a temporary setup, the box provides ease of testing and troubleshooting during construction of the plant, as well as any changes made to the system afterwards. The diagram of this is shown in Figure 10. The two columns of 8-bit data lines are displayed using standard green LED’s, while the two 4-bit control columns are shown using red. Finally, the two columns of input lines are shown in yellow.

![Address Table](image)

<table>
<thead>
<tr>
<th></th>
<th>Data Output</th>
<th>Status</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P1</strong></td>
<td>956D</td>
<td>957D</td>
<td>958D</td>
</tr>
<tr>
<td><strong>P2</strong></td>
<td>888D</td>
<td>889D</td>
<td>890D</td>
</tr>
</tbody>
</table>

**Figure 10.** The I/O readout box.
As can be seen, the various input and output lines are displayed from each parallel port used in the project. The address table above shows the address of each parallel port's respective I/O. These addresses are the decimal numbers used in the software to control the different lines. As standard in computer addressing, the 8-bits of each data port must be controlled simultaneously. For example, if it was desirable to turn on the first, third, and fourth bits only, \( (10110000) \) the decimal value of thirteen would have to be assigned, since this corresponds to the binary value of 10110000. The hardware constructed for the prototype can be seen in Figure 11.

Figure 11. The I/O box used for interfacing between the computer and plant.
The I/O readout box is connected to the plant by way of the digital control board. This board is responsible for switching sensors to the input lines of the I/O box. The digital control board schematic can be seen in Figure 12.

![Schematic of the digital control board](image)

**Figure 12.** The schematic of the digital control board.

Output lines are labeled P1D0, P1D1, P1D2, P1D3, P1D4, P1D5, P1D6, P1D7, P1C2, and P2C0. They connect to the plant by means of a +5 volt relay bank, which can be seen on the left of Figure 13. Input sensors are connected to P1 and P2, two four bit ports that are combined together in software to create an 8-bit port. During machine operation, these ports are frequently switched to multiple sensors including two LM335 temperature sensors, three 10K potentiometers, and a pressure transducer.
Figure 13. The digital control board.

The first part of the code consists of a while loop that maintains the main program to allow the option of running the program or exiting the system. The second part of the code deals with inputting the binary values representing how many cups of coffee are desired and how strong the coffee should be. The information is then read from the parallel port and appropriately scaled. Then a small ‘while’ loop allows time for the power relays and the multiplexer to switch. After the while loop, the input for the strength of the coffee is taken and appropriately scaled.

Following that, the code assigns values to both the input and output matrices. The main body of the code deals with the calculations needed to determine which fuzzy set
memberships apply. Testing the input to see if it falls within the fuzzy set and, if so, calculating the membership performs this. Also, this portion of code determines the position holders that make calculating the reduced rule matrix easier. There are also two ‘if’ statements that cover when the input is greater than or less than the maximum and minimum values, respectively. However, this should never occur if the hardware and input scaling are set up appropriately.

After both input memberships have been located, the output is calculated. This is done by first determining the reduced rule matrix. The position holders from the input fuzzy sets are used to reduce the rule matrix to a two-by-two matrix. Taking a fuzzy set and multiplying it with the remaining fuzzy set along with the reduced rule matrix produces the output in terms of tablespoons. This product must be converted to a variable representing the amount of time needed for the hopper to be open. This is achieved by simply multiplying the output by a scaling factor. This scaling factor aids in reducing any existing overshoot. Then, in a small ‘while’ loop the command to open the hopper is set high until the incrementing counter in the loop reaches the scaled value.

A flow chart of the entire operational cycle can be seen in Figure 14. This chart displays the fundamental steps taken in the programming.
Figure 14. The flow chart of the control program main loop.
3.3.2 Overview of the Plant

The plant in this case consists of the intricate assembly electrical and mechanical devices, shown in Figure 15. These incorporate two heating elements traditionally designed for 120 volt operation, two 24 volt, alternating current operated water solenoids, a small 120 volt pump, a bi-directional dc drive motor, two 24 volt dc actuators, two LM335 temperature sensors, and a pressure sensor. The temperature sensors monitor and alter the input voltage, and subsequently change the heat output, controlling each heating element. The two actuators are built into the mechanical assembly, which opens and closes the two coffee hoppers. The dc motor is responsible for actuating the filter assembly. It only possesses two states, referred to simply as the brew state and the clean state. In brew state, the coffee filter is kept vertical, allowing hot water to flow into it, through the grounds, and out. After running the coffee brewing cycle, the dc motor rotates the filter assembly 90 degrees to the clean state, where it is sprayed with water to wash out the used coffee grounds. Upon completion, the motor then rotates the assembly back to the brew state making it ready for another cycle. Since this state only requires two states, the positions can be achieved by incorporating limit switches, which is good practice since they perform the positioning and also provide protection from the assembly over rotating and potentially damaging the hardware.
These individual devices are operated in sequence, creating the automated actions needed to brew coffee correctly as well as perform the necessary tasks needed to clean the system and prepare the plant for another run through, effectively completing a loop of functionality. The technical specifics can be seen in the schematic drawing in Figure 16.
The plant schematics show eight relays used to control loads H1 and H2, the heating elements of the plant. These relays are solenoid switches incorporating mechanical state memory. By using these, the control computer is only required to provide a momentary pulse to the solenoid, allowing it to change state before the digital control line is set low and the solenoid is de-energized. The eight solenoids can be seen in Figure 17.
3.3.2.1 Ground Coffee Dispenser

There are two coffee dispensers installed in the plant. One contains regular coffee grounds and the other holds decaffeinated. The user makes the choice of which to use, or if the caffeine elimination subroutine is active, a mixture of the two. Both are identical in mechanical and electrical respects. As seen in Figure 18, a system of linkages attaches a 24 volt dc solenoid to a stopper mounted in the inside of the dispensers. Included in the two dispenser systems are indicator lights, as well as a low voltage agitator to prevent dispenser blockage. Each activator is activated at the same time as its respective solenoid.
Figure 18. The mechanical layout of the coffee dispensers.

From a controls perspective, it becomes apparent that the two dispensers each form an open control loop. That is, there is no sensor that detects granual flow from each dispenser. This means that the accuracy of the system is completely dependent on proper calculation of the ‘on’ time as well as the mechanical efficiency of the dispenser. Put
another way, the flow rate from both of the dispensers must remain a reasonably stable constant.

The coffee dispenser system requires meticulous adjustment on the controller gains due to delay introduced in the mechanical linkages. The hardware constructed for the prototype can be seen in Figure 19.

Figure 19. The Regular Coffee Dispenser.
3.3.2.2 The Central Brewing Chamber

The actual act of coffee brewing occurs in the central brewing chamber. This component is essentially the keystone of the plant, as most of the other components meet at this junction. The central brewing chamber houses the filter assembly, a unit similar to other coffee filter cups. It is mounted on a horizontal axle that passes through to the outside of the chamber where it is connected to a gearbox through a push-rod linkage. The filter has two states: brew mode and clean mode. In brew mode, the filter is positioned upright, allowing hot water and coffee grounds to flow into it, where extraction occurs. The brewed coffee then leaks out through a hole in the bottom, out of the brewing chamber, and into the waiting carafe below. A diagram of the central brewing chamber can be seen in Figure 20.

![Diagram of the central brewing chamber hardware.](image)

Figure 20. The mechanical layout of the central brewing chamber hardware.
In clean mode, which occurs after coffee brewing has completed, the filter assembly is rotated ninety degrees by the gearbox push-rod linkage and solenoid 2, (S2 in Figure 20 above) is activated. Water from the main line enter the filter, cleaning out the used grounds, and all existing residue is washed away into the main drainage. The gearbox can be seen in Figure 21.

Figure 21. The filter rotation gearbox.
Since the gearbox is connected to the brewing chamber by means of a prismatic control rod, the location of it relational to the chamber isn’t important. Since there are only two states the gearbox needs to move the filter assembly into, there is no need for elaborate motor control. Therefore, control of this device can be limited to simple relay and switch control. A digital input line, when set high, energizes the relay in the gearbox, driving the DC motor. The motor continues until a range limit switch is contacted, deactivating the motor. When the digital line is set low, the relay is deactivated and power forced though the motor in the opposite direction until the second range limit switch is contacted. The schematic for this sub circuit can be seen in Figure 22.

Figure 22. The filter assembly gearbox used for the filter washing cycle.

Figure 23 shows the hardware constructed for the prototype.
3.3.2.3 The Main Heating Tank Assembly

The main heating tank is responsible for keeping a supply of water at the optimum brewing temperature at all times. This piece of hardware requires continuous monitoring by use of a temperature sensor emersed within the lower tank. By receiving feedback from this sensor, the computer can re-evaluate the necessary power switch and change the voltage supplied to the heating element also contained within the lower tank. Figure 24 shows the schematic of the water heating apparatus.

Figure 23. The Central Brewing Chamber.
In operation, when the coffee making routine is activated by pressing the Start button, the coffee making routine makes a determination of the amount of cups of coffee that are to be made. This results is a time for the supply water solenoid to be turned on, which releases water into the upper tank. The upper tank provides acts as a holding reservoir which slowly drains into the lower tank, forcing heated water out through the ¼” copper line that feeds into the filter assembly. The temperature sensor controlling the electric heater makes the closed loop controller of the main water heater, one of four control sub-systems. The hardware can be seen in Figure 25.
3.3.2.4 Carafe Heater

The carafe heater is a 100 watt element heater installed below the central brewing chamber. It provides a base for the carafe to sit while heating the coffee contained
within. The carafe heater is controlled much in the same way as the main heating tank. It’s controlled by four power solenoids with mechanical memory state. The carafe heater is shown in Figure 26.

![Figure 26. The carafe heater.](image)

### 3.3.2.5 Carafe and Pump

The carafe is a standard BUNN coffee carafe modified to contain an LM335 temperature responsive transistor in series with a 1K Ohm resistor, which interfaces the rest of the plant by means of a four pin telephone connector port. Also installed within the carafe is a ¼” I.D. tube connected to an external pump capable of pumping out excess coffee. The carafe can be seen in Figure 27.
Figure 27. The carafe modified for temperature sensing and vacuum coffee removal.

Working in conjunction with the carafe is the pump. This standard, non-priming pump is responsible for pumping out excess coffee from the carafe at the beginning of the coffee making routine. There is no sensor to detect whether excess coffee exists. Therefore, the pump is always activated for several seconds at the beginning of the cycle to ensure that old coffee has been removed from the system and drained into the waste receptacle. Figure 28 shows the 120 V pump.
3.3.3 Temperature Sensors

Two temperature sensors are used in the plant. Coffee is best brewed at one temperature and served at a slightly lower one. For this reason, one sensor is required at the main water-heating tank, monitoring the direct temperature of the water as it waits to be used. The second is directly coupled to the serving carafe, monitoring the brewed coffee temperature. Each is centered around the LM335 precision temperature sensor.
Operating as a two-terminal zener diode, the LM335 has a breakdown voltage directly proportional to the absolute temperature at +19 mV/Kelvin degree. With less than 1 Ohm dynamic impedance the device operates over current range of 400 microamperes to 5 mA with virtually no change in performance. The LM335 has typically less than one-degree error over a 100-degree temperature range. With a near linear output, this is more accuracy than needed for water and coffee temperature stabilization. However, with the low cost, availability, and ease of use, the LM335 is a good choice.

Each sensor is used in conjunction with an analog to digital converter, which converts the analog signal to an 8-bit digital one, allowing the data to be read by the computer acting as the controller for this project. The 8-bit word is converted to a decimal value within the control program, which is then used in the weighted fuzzy calculation to select which voltage is fed to each heating element, in turn controlling the temperature. This creates a classic feedback system. The computers sampled data can be correlated to actual temperature by means of taking a data set to solve for the relational curve. This is done by taking sample values from the computer while simultaneously recording the water temperature with an external thermometer. The relational data can be seen in Figure 29.
The same technique can be used for calibrating the main water heater tank temperature probe. Another set of data was recorded using an external temperature-monitoring device submerged into the tank. This data and correlation can be seen in Figure 30.
Figure 30. The correlation between tank temperature and sampled computer data.

The main water-heating tank maintains continuous monitoring both during coffee preparation and standby, keeping the water at brewing temperature at all times. Coffee is traditionally best brewed at 118 degree Fahrenheit. After reading the voltage given from the LM335 and converted into an 8-bit word, the controller program uses the fuzzy rule matrix to select which of four control lines to connect to the heating element. Each control is connected to an alternating current, varying in voltage. Should the heater exceed the desired temperature, no control lines are connected. After taking several samples, the rate of heat loss can be calculated, and a decision of the voltage setting to feed the heater can be determined.
3.3.4 Pressure Sensors

The pressure sensor measures water pressure entering the system. This is accomplished by use of the Model SA Honeywell pressure transducer. The diaphragm of this device is directly coupled to the main water intake and takes absolute pressure measurements. The SA produces a high-level voltage output of 1 to 6 volt output from an unregulated supply and requires no external calibration. The output of the pressure transducer is connected to an analog to digital converter, which converts the analog signal output to a usable 8-bit digital signal allowing the computer acting as the controller to take measurements of absolute pressure. This is necessary during the water filling cycle. The pressure transducer hardware used for the prototype can be seen in Figure 31.

![The SA Honeywell Pressure Transducer](image)

Figure 31. The SA Honeywell Pressure Transducer
Water pressure in home and office plumbing isn’t reliable due to the unpredictable and continuous usage at other outlets. A water grid is a contained system traditionally with one inlet and an unknown quantity of outlets. These outlets consist of faucets, toilets, taps, and any other appliances. As each of these individual outlets is operated, the total pressure in the entire grid is decreased, in turn decreasing the potential water flow at other available outlets. Since the water solenoid, which operates the water filling procedure relies on a time value calculated within the fuzzy rule matrix based on the user input, a fluctuating water flow would produce undesirable results, most likely giving less coffee than wanted, in turn destroying the balance between water and coffee volume, giving strong coffee. For this reason, water pressure must be monitored to ensure correct results. When an 8-bit signal is received from the pressure transducer, it is read as an integer between 0 and 256. This can be related to pressure by means of calibration from a data set, seen in Figure 32. This data was gained by physically altering the water pressure outside of the plant, recording the raw data, followed by solving for the relation curve.
Figure 32. The relation between pressure and sampled data.

3.3.5 User Input

Before activating, several decisions are required by the user. The front control panel contains three potentiometers and two buttons. One potentiometer decides the number of cups of coffee. The second potentiometer decides the strength of the coffee. The third potentiometer is used to input the ratio of regular coffee grounds and decaffeinated coffee. The low lower left button begins the software program, and the right button is a wash button to perform a filter washing cycle assuming the brewing stage has not been activated.

User input for system can be input from the interface panel located on the front of the plant. This is shown in Figure 33. However, the controls can also be accessed by means of a remote console driven by a computer networked to the coffee control system.
The remote system is driven and supported by the coffee control program, using the Linux graphics library GTK. This creates a virtual user interface that mimics the layout of the physical, local interface panel. All features and controls are available on both and can be used interchangeably. Additionally, some sensor feedback is available on the remote system, showing temperature and pressure readouts. This was for testing and monitoring purposes only.
4. Results and Discussion

In this chapter, the results of the automated coffee maker are presented. Producing proper amounts of coffee, water, and regulating temperature are the important aspects of this thesis. Four individual load controllers are incorporated within the main control system of the coffee machine: the water volume controller, the primary water heater temperature controller, the carafe temperature controller, and the coffee ground-dispensing controller. The responses of these controllers are analyzed independently based on the rise time, steady-state error and overshoot.

4.1 Water Volume Controller

The water pressure-controlled flow regulator can be evaluated by the efficiency of the water volume obtained. Volume is calculated from the time the water supply solenoid is opened and subsequently closed. In this controller, the rise time depends on the water supply pressure and the finite dimension of the plumbing used to transfer the water. In a typical home, any device that uses water, such as washing machines, showers, or faucets, affects the water supply pressure. Also, it is important that the water volume controller has no overshoot. Overshoot in this controller equates to excess water. Excess water would have the dual effect of creating more coffee than necessary as well as disturbing the relationship between the amount of coffee grounds and water, an important ratio which determines the strength of the coffee created.
Before filling the sub tank, the pressure is sampled and used to determine the flow rate in gallons per minute. It is assumed that the water pressure remains constant during the actual filling cycle. Figure 34 shows that the area under the control curve equates to the amount of the water dispensed.

![Figure 34. The water volume response.](image)

4.2 Primary Water Heater and Carafe Temperature Controllers

The primary water heater and secondary carafe heater can be evaluated based on the rise time of the temperature, the overshoot characteristic, and the steady-state response of the two systems. Figure 35 shows the plot of the temperature vs. time for the water heater response curve. Due to the relatively slow response time of the water
temperature, it’s not difficult to meet reasonable design criteria, such as a three degree Fahrenheit steady-state error. With the power completely cut off at 203 degrees Fahrenheit, the overshoot problem is easily avoidable. The initial rise time is roughly 62 seconds. To improve this time, an electric heater with a higher wattage must replace the existing element. However, this is the worst-case scenario since the water within the tank starts at room temperature.

![Temperature vs. time response of the main water tank controller.](image.png)

Figure 35. The temperature vs. time response of the main water tank controller.

The carafe heater must regulate the temperature at a lower value than the main water tank. Note that 160 degrees Fahrenheit is the optimal value for serving. Figure 36 shows the temperature response of the carafe heater controller. Since the temperature of the coffee entering the carafe is typically just less than 200 degrees Fahrenheit, the
controller remains off until that heat is dissipated. It then regulates the temperature around the target value. The temperature sensor feedback has a resolution of +/-3 degrees. That resolution hurts the accuracy of the steady-state error, but by fine-tuning the control membership function close to the target value, the temperature can be maintained within an acceptable range.

![Figure 36. The temperature vs. time response of the carafe heater controller.](image)

### 4.3 Coffee Ground-Dispensing Controller

The coffee dispensing control system has three inputs and two outputs. It is required that this controller produces no overshoot. An overshoot would mean too much coffee has been dispensed, resulting in a stronger taste than desired. An overshoot must also be avoided because no mechanism exists to remove granules of ground coffee. The
best test for accuracy is to look for repeatability of the volume of coffee for each cup and strength set. The correct volume of regular and decaffeinated was chosen based on the taste of the coffee produced and refined by trial and error. Even though the values were originally chosen by a system expert, this still isn’t sufficient for a consensus viewpoint. For example, if the expert prefers coffee that is stronger than the average population of coffee drinkers, then average coffee drinkers will interpret ‘medium’ closer to ‘strong’.

To truly obtain a more universal result, a random sampling of coffee drinkers would be required. This can be rectified by taking a sample of the coffee drinking population and recording what’s interpreted as a ‘medium’, ‘dark’, and ‘light’ cup of coffee. By performing this task, there’s a better chance of tuning the rule base to what the average population of coffee drinkers would consider correct. Then, only slight variations at the input controls would be required to accommodate most coffee drinkers.

Though the four controllers mentioned above are critical for proper operation of the system, the main objective is to create coffee that appeals to the user in a suitable time frame. Concerning time frame, the more cups of coffee that are desired, the larger the rise time of the system is, though at maximum number of cups plus maximum strength, the operation time is under five seconds. This is an acceptable time frame for granular dispensing and still far quicker than the manual alternative. Brewing twelve cups of coffee in under one minute, is an acceptable time frame. Though the construction of the dispensers is relatively simple, the flow rate doesn’t deviate enough to seriously alter the outcome. The relationship between time and coffee volume can be seen in Figure 37.
Figure 37. The solenoid curve (top) and the coffee ground output (bottom).
5. Conclusions/Recommendations

5.1 Conclusions

The primary objective of this thesis is to design and create a control system for an autonomous coffee maker. This system must be capable of receiving user and physical environmental input capable of altering the outcome, and using this information to adapt itself so that it may produce a delectable amount of coffee. The second objective is to design and construct a system that is capable of working in conjunction with the controller, as making coffee is the only true way to test the efficiency of the controller. The research illustrates some of the issues associated with the development of a real-time, real-world automated system. That is, fine-tuning the coffee dispenser rule base, sensor placement, plumbing integration, and interaction between the different components are common and must be overcome. This research provides a sample of applied techniques to overcome these types of problems.

The system incorporates four controllers responsible for separate functions. The coffee granular dispenser system is an open-loop control system that assumes a constant flow rate. Dispenser activation is calculated, which is obtained using two input variables: the number of cups and the value of the coffee strength. The hardware designed and built gives a suitable flow rate that can be relied on to return accurate results.

The water pressure controller is an open-loop system that uses two inputs to obtain the correct water solenoid activation time. A PX209 pressure transducer is used to
sample the water pressure at the time of dispensing. The PSI value is determined and
checked against a table to determine the gallons/minute flow rate. The number of cups
value taken from user input is combined with the pressure information and the result is
used as input for a fuzzy table to obtain the final solenoid opening time. The system
provides accurate results in terms of water volume.

The water and coffee temperature controllers are closed-loop systems that feed
input from the LM335, a temperature responsive thermistor. Each sensor is scanned
continuously, converted to a digital representation, and used to interpret the temperature
of the water or coffee. This value is checked for placement within one of the flat-topped,
non-overlapping membership to interpret the correct course of action. The chosen level
of power is fed to the individual heating coils. The systems maintain an accuracy of at
least +/-three degrees Fahrenheit.

In developing the automated coffee maker, the developer must possess knowledge
in a wide array of areas including programming, computer hardware, electrical
engineering, electrical wiring, plumbing, physics, mechanical engineering, and most
importantly a refined palette worthy of testing coffee.
5.2 Future Development

The prototype machinery adequately maintains all required conditions and produces favorable coffee. The system developed in this research can be used as a base for future development. Of course, there is also room for improvement in the current design. Below are some recommended areas of future work:

- This experimental plant was designed and built with low cost, off the shelf parts in mind. Therefore, size isn’t optimized, resulting in a rather large plant. The size would prove to be a problem in many offices and homes, and is not feasible for mass marketing. However, a new design could be implemented to minimize space requirements.

- Load components should be picked with power specifications in mind. Components that require similar power requirements would simplify the power supply as well.

- The system incorporates only the minimum amount of sensors to operate the basic features of this prototype. With a few additional sensors, the system could be smarter in making judgments leading to better efficiency. For example, excess coffee is pumped out of the carafe regardless of whether excess coffee remains in
the carafe after use. This could be improved by incorporating a level sensor within the carafe.

- The water control function is an open-loop control system. There is no way for the plant to directly judge whether the proper volume of water has been dispensed. Only approximations can be made based on the pressure. This could be solved by adding a water flow sensor, tabulating the amount of water that has passed through the sensor, giving a more straightforward method of determining water volume.

- The coffee dispenser controller also fails to meet the requirements of a closed loop system. Grain measurement flow sensors are expensive, and weight sensors would have proven more difficult to implement in this design, so they weren’t incorporated. Therefore, the correct amount of coffee passing from the hopper in question to the filter relies on the grains ability to flow at a steady rate. This may be unreliable. An improvement to the system would facilitate a better way of measuring grain flow from the hoppers. The heater control solenoids could be simplified by replacing them with a power inverter circuit or a DC pulse width modulator design.
• While a standard desktop computer is currently being used to operate the plant, the software written could be transferred to a microcontroller or PLC. This would reduce cost and size of the controller.

• Though the machine is designed to allow variance of the strength, a broader range of test subjects could be taken to calibrate the machine. This would give a more accurate depiction of what the coffee drinking population regards as average strength.

• The main program loop speed could be increased if individual sensor sampling and holding responsibilities were transferred to secondary circuits. Each circuit could be calibrated to the speed of the sensor it’s connected to and sample as much as possible. The main program loop would then be free to cycle faster and check for user input more often, minimizing any observable delay in machine interactions.

• An algorithm can be designed to help health conscious people slowly reduce their daily caffeine intake. The plant allows a ratio of regular coffee grounds to be blended with decaffeinated. An algorithm could be developed to record the amount of caffeine a coffee drinker has, and decrease that ratio daily until the mixture is 100% decaffeinated.
This appendix contains the necessary information for the use and operation of the automated coffee maker system designed for partial completion of Master’s degree requirements.

Operation

1) Connect the electric power cord into a standard 120 VAC wall outlet.
2) Connect the water main to a standard ½” threaded water supply. Tighten as needed and open the flow valve.
3) Connect the drainage line to a standard 1 ¼” drain line.
4) Switch on the computer power switch.
5) After the computer has correctly booted and waiting at the prompt, open a terminal window by right clicking the desktop and selecting ‘terminal’.
6) At the terminal prompt, type ‘su’ and enter the root password to obtain root level status.
7) type ‘./thesis/coffee_interface’.
8) You will be prompted to select the coffee amount, strength, and type of coffee desired. Select these values on the front panel of the machine.
9) Allow at least ten minutes for the water in the system to warm to optimum brewing temperature, 200 degrees Fahrenheit.

10) Select the number of cups desired using the cups adjustment knob on the front interface panel.

11) Select the strength value desired using the strength adjustment knob on the front interface panel.

12) Select the ratio of regular to decaf desired using the ratio adjustment knob on the front interface panel.

13) Press the START button.
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
#include <gtk/gtk.h>
#include <sys/io.h>
#include "coffee.h"
#include "globdefs.h"
#include "interface.h"
#include "sound.h"
#include "utils.h"
#include "fuzzy.h"

/* Function prototypes */
void ACTIVATE_MOTOR ();
void ACTIVATE_PUMP ();
void ACTIVATE_WATER_FILL ();
void ACTIVATE_WATER_CLEAN ();
void ACTIVATE_DECAF_HOPPER ();
void ACTIVATE_REGULAR_HOPPER ();
void ACTIVATE_DECAF_AGITATOR ();
void ACTIVATE_REGULAR_AGITATOR ();
void ACTIVATE_INDICATOR ();
void ACTIVATE_OFF ();

/******************** SOUND BUFFER ALLOCATION *******************/
playing_t playing[MAX_PLAYING_SOUNDS];
SDL_AudioSpec desired, obtained;
sound_t sound1, sound2, sound3, sound4, sound5, sound6, sound7, sound8, sound9, sound10;

/******************************************************************************
 *    CloseTheApp                *
 *    *
 *    A function to close down the system before terminating               *
 *    This frees up the memory allocated for sound buffer                  *
 *    as well as turning everything off, closing the data file             *
 *    and finally exiting the gtk graphics system                          *
 *    *
 ******************************************************************************/
void CloseTheApp ( GtkWidget * main_window, gpointer data )
{
    SDL_PauseAudio(1);
    SDL_LockAudio();
    free(sound1.samples);
    free(sound2.samples);
    free(sound3.samples);
    free(sound4.samples);
    free(sound5.samples);
    free(sound6.samples);
    free(sound7.samples);
    free(sound8.samples);
    free(sound9.samples);
    free(sound10.samples);
    SDL_UnlockAudio();
    ACTIVATE_OFF();
    TURN_OFF();
    fprintf(outfile, "---------------------- END OF CYCLE ---------------------\n");
    fclose(outfile);
    gtk_main_quit();
}

void AudioCallback(void *user_data, Uint8 *audio, int length)
{
    int i;
memset(audio, 0, length);

for (i=0; i < MAX_PLAYING_SOUNDS; i++) {
    if (playing[i].active) {
        Uint8 *sound_buf;
        Uint32 sound_len;

        sound_buf = playing[i].sound->samples;
        sound_buf += playing[i].position;

        if ((playing[i].position + length) >
            playing[i].sound->length) {
            sound_len = playing[i].sound->length -
            playing[i].position;
        } else {
            sound_len = length;
        }

        SDL_MixAudio(audio, sound_buf, sound_len, VOLUME_PER_SOUND);

        playing[i].position += length;

        if (playing[i].position >= playing[i].sound->length) {
            playing[i].active = 0; /* mark it inactive */
        }
    }
}

int LoadAndConvertSound(char *filename, SDL_AudioSpec *spec,
                        sound_p sound)
{
    SDL_AudioCVT cvt;     /* format conversion structure */
    SDL_AudioSpec loaded; /* format of the loaded data */
    Uint8 *new_buf;

    if (SDL_LoadWAV(filename,
if (SDL_BuildAudioCVT(&cvt, loaded.format,
        loaded.channels, loaded.freq,
        spec->format, spec->channels,
        spec->freq) < 0) {
    printf("Unable to convert sound: %s\n", SDL_GetError());
    return 1;
}

cvt.len = sound->length;
new_buf = (Uint8 *) malloc(cvt.len * cvt.len_mult);
if (new_buf == NULL) {
    printf("Memory allocation failed.\n");
    SDL_FreeWAV(sound->samples);
    return 1;
}

memcpy(new_buf, sound->samples, sound->length);

cvt.buf = new_buf;
if (SDL_ConvertAudio(&cvt) < 0) {
    printf("Audio conversion error: %s\n", SDL_GetError());
    free(new_buf);
    SDL_FreeWAV(sound->samples);
    return 1;
}

SDL_FreeWAV(sound->samples);
sound->samples = new_buf;
sound->length = sound->length * cvt.len_mult;

printf("%s' was loaded and converted successfully.\n", filename);

return 0;
}

/* Removes all currently playing sounds. */
void ClearPlayingSounds(void)
{

int i;

for (i = 0; i < MAX_PLAYING_SOUNDS; i++) {
  playing[i].active = 0;
}

int PlaySound(sound_p sound)
{
  int i;

  /* Find an empty slot for this sound. */
  for (i = 0; i < MAX_PLAYING_SOUNDS; i++) {
    if (playing[i].active == 0)
      break;
  }

  /* Report failure if there were no free slots. */
  if (i == MAX_PLAYING_SOUNDS)
    return 1;

  SDL_LockAudio();
  playing[i].active = 1;
  playing[i].sound = sound;
  playing[i].position = 0;

  SDL.UnlockAudio();

  return 0;
}
/* Sound file calling functions */

void ACTIVATE_SOUND1 ()
    { PlaySound(&sound1); }
void ACTIVATE_SOUND2 ()
    { PlaySound(&sound2); }
void ACTIVATE_SOUND3 ()
    { PlaySound(&sound3); }
void ACTIVATE_SOUND4 ()
    { PlaySound(&sound4); }
void ACTIVATE_SOUND5 ()
    { PlaySound(&sound5); }
void ACTIVATE_SOUND6 ()
    { PlaySound(&sound6); }
void ACTIVATE_SOUND7 ()
    { PlaySound(&sound7); }
void ACTIVATE_SOUND8 ()
    { PlaySound(&sound8); }
void ACTIVATE_SOUND9 ()
    { PlaySound(&sound9); }
void ACTIVATE_SOUND10 ()
    { PlaySound(&sound10); }

/*****************************/
* Function responsible for running the wash cycle routine.      *
* Notifies with a sound, activates the filter assembly to      *
* run in wash mode, activates the water solenoid for          *
* cleansing, waits, then resets the filter assembly.          *
*                                                            *
*************************************************************/
void WASH_ACTIVATE ()
{
    if ( command_location == 0 )
    {
        fprintf(outfile, "WASH ROUTINE ACTIVATED FROM LOCAL SOURCE\n");
    }
    else if ( command_location == 1 )
    {
        fprintf(outfile, "WASH ROUTINE ACTIVATED FROM REMOTE SOURCE\n");
    }
ACTIVATE_SOUND7();

ACTIVATE_MOTOR(); // Rotate filter assembly to wash position
sleep(3);          // Wait a few seconds
ACTIVATE_WATER_CLEAN(); // Turn on Water washing solenoid
sleep(4);          // Wash for four seconds
TURN_OFF_P1DATA();  // Turn off water solenoid
sleep(2);          // Wait another two seconds
TURN_OFF();        // Reset everything and let filter get back into place
sleep(3);          // Wait for filter assembly

} // COFFEE ACTIVATE

('').*/

void COFFEE_ACTIVATE()
{
    char cups[20];
    char strength[20];
    char ratio[20];
    char pressure[20];
    // Code for COFFEE ACTIVATION
}
char buffer[20];
double input_cups;
double input_strength;
double input_ratio;
int indicator_counter;

ACTIVATE_SOUND10();

for (indicator_counter = 0; indicator_counter < 3; ++indicator_counter)
{
    ACTIVATE_INDICATOR();
sleep(1);
    TURN_OFF_P1CTRL();
sleep(1);
}

Decipher if system was activated locally or remotely. We need to know which data set to use as input

if ( command_location == 0 )
{
    fprintf(outfile, "COFFEE MAKING ROUTINE ACTIVATED FROM LOCAL SOURCE\n");
    sprintf(buffer, "Cups = %3.2lf\n", MW.cup_local); fprintf(outfile, buffer);
    sprintf(buffer, "Strength = %3.2lf\n", MW.strength_local); fprintf(outfile, buffer);
    sprintf(buffer, "Ratio = %3.2lf\n", MW.ratio_local); fprintf(outfile, buffer);

    /* ASSIGN LOCAL INPUT VARIABLES TO FUZZY CALCULATION PROTOCOLS */
    input_cups = MW.cup_local;
    input_strength = MW.strength_local;
    input_ratio = MW.ratio_local;

    FUZZY.cups_input = input_cups;
    FUZZY.strength_input = input_strength;
    FUZZY.ratio_input = input_ratio;
}
else if ( command_location == 1 )
{
    fprintf(outfile, "COFFEE MAKING ROUTINE ACTIVATED FROM REMOTE SOURCE\n");
sprintf(buffer, "Cups = %3.2f\n", MW.cup_remote); fprintf(outfile, buffer);
sprintf(buffer, "Strength = %3.2f\n", MW.strength_remote); fprintf(outfile, buffer);
sprintf(buffer, "Ratio = %3.2f\n", MW.ratio_remote); fprintf(outfile, buffer);

/* ASSIGN REMOTE INPUT VARIABLES TO FUZZY CALCULATION PROTOCOLS */
input_cups = MW.cup_remote;
input_strength = MW.strength_remote;
input_ratio = MW.ratio_remote;

FUZZY.cups_input = input_cups;
FUZZY.strength_input = input_strength;
FUZZY.ratio_input = input_ratio;

} 
ACTIVATE_SOUND7 ();
sleep(2);
//Perform fuzzy calculations based on input of choice
// calculate_total_dispense_time ();
// calculate_ratio_times ();

// Activate the Regular Hopper for the allotted time and agitate
// ACTIVATE_REGULAR_AGITATOR ();
// sleep(1);
// ACTIVATE_REGULAR_HOPPER ();
/* pause for calculated time */
// TURN_OFF ();

// Activate the Decaf Hopper for the allotted time and agitate
// ACTIVATE_DECAF_AGITATOR ();
// sleep(1);
// ACTIVATE_DECAF_HOPPER ();
/* pause for calculated time */
// TURN_OFF ();

// Activate the water solenoid and let the sub-tank fill
calculate_water_dispense_time ();

}
gint main_loop ( gpointer data )
{
    int input_combined;
    char paster[20];

    fprintf(outfile, "Beginning main loop\n");
    // printf("Beginning main loop\n");

    char line[100];
    int value;

    if ( INV_VAR.initial_run < 3 )
    {
        outb( P2C0, port2_ctrl);
        WASTE_TIME();
        outb( 11, port2_ctrl);
        INV_VAR.initial_run++;
    }

    if ( MW.cup_remote < 16.7 )
    {
        sprintf(dial_value, "2");
        gtk_label_set_text ( GTK_LABEL ( MW.cup_dial_fuzzy_label ), dial_value);
    }
    else if ( 16.7 >= MW.cup_remote || MW.cup_remote < 33.36 )
    {
        sprintf(dial_value, "4");
        gtk_label_set_text ( GTK_LABEL ( MW.cup_dial_fuzzy_label ), dial_value);
    }
    else if ( 33.36 >= MW.cup_remote || MW.cup_remote < 50.06 )

else if ( 50.06 >= MW.cup_remote || MW.cup_remote < 66.7 )
{
    sprintf(dial_value, "8");
    gtk_label_set_text ( GTK_LABEL ( MW.cup_dial_fuzzy_label ), dial_value);
}
else if ( 66.7 >= MW.cup_remote || MW.cup_remote < 83.46 )
{
    sprintf(dial_value, "10");
    gtk_label_set_text ( GTK_LABEL ( MW.cup_dial_fuzzy_label ), dial_value);
}
else if ( 83.46 >= MW.cup_remote || MW.cup_remote < 100 )
{
    sprintf(dial_value, "12");
    gtk_label_set_text ( GTK_LABEL ( MW.cup_dial_fuzzy_label ), dial_value);
}

/* UPDATE THE STRENGTH LABEL ABOVE THE DIAL TO ALERT REMOTE USERS TO SETTING */
if ( MW.strength_remote < 25 )
{
    sprintf(dial_value, "LIGHT");
    gtk_label_set_text ( GTK_LABEL ( MW.strength_dial_fuzzy_label ), dial_value);
}
else if ( 25 >= MW.strength_remote || MW.strength_remote < 40 )
{
    sprintf(dial_value, "LIGHT MEDIUM");
    gtk_label_set_text ( GTK_LABEL ( MW.strength_dial_fuzzy_label ), dial_value);
}
else if ( 40 >= MW.strength_remote || MW.strength_remote < 60 )
{
    sprintf(dial_value, "MEDIUM");
    gtk_label_set_text ( GTK_LABEL ( MW.strength_dial_fuzzy_label ), dial_value);
}
else if ( 60 >= MW.strength_remote || MW.strength_remote < 75 )
{
    sprintf(dial_value, "MEDIUM DARK");
    gtk_label_set_text ( GTK_LABEL ( MW.strength_dial_fuzzy_label ), dial_value);
}
else if ( 75 >= MW.strength_remote || MW.strength_remote < 100 )
{
sprintf(dial_value, "DARK");
gtk_label_set_text ( GTK_LABEL ( MW.strength_dial_fuzzy_label ), dial_value);
}

/* UPDATE THE RATIO LABEL ABOVE THE DIAL TO ALERT REMOTE USERS TO SETTING */
double reg_per, dec_per;
reg_per = 100 - MW.ratio_remote;
dec_per = MW.ratio_remote;

sprintf(dial_value, "R: %3.0lf%%  D: %3.0lf%%", reg_per, dec_per);
gtk_label_set_text ( GTK_LABEL ( MW.ratio_dial_fuzzy_label ), dial_value);

/*
  1.) Read cups selector
  2.) Read strength selector
  3.) Read ratio selector
  4.) Read pressure sensor
  5.) Read carafe temperature sensor
  6.) Read tank temperature sensor
  7.) Read start button
  8.) Read wash button
  9.) Read heat switches
*/
ACTIVATE_OFF();
WASTE_TIME();

outb( 0, port2_data);
WASTE_TIME();
outb( 16, port2_data); // toggle write
WASTE_TIME();
outb( 0, port2_data); // clear
WASTE_TIME();
outb( 8, port2_data); // toggle read high
WASTE_TIME();
// printf("Press any key to sample cups value\n");
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
inputA = inb(port1_status); // reading cups A
inputB = inb(port2_status); // reading cups B
input_combined = INPUT_CONVERT(inputA, inputB);
/* # CUPS 0-100 CONVERSION FACTOR */
MW.cup_local = (input_combined/255)*100;

fprintf(outfile, "Cups A: %i  Cups B: %i Combine: %i\n", inputA, inputB, input_combined);
sprintf(paster, "CUPS: %i", input_combined);
gtk_label_set_text ( GTK_LABEL ( MW.cups_label ), paster);
WASTE_TIME();

outb( 128, port2_data);
WASTE_TIME();
outb( 144, port2_data);
WASTE_TIME();
outb( 128, port2_data);
WASTE_TIME();
outb( 136, port2_data);
WASTE_TIME();
// printf("Press any key to sample strength value\n");
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
inputA = inb(port1_status);  // reading strength A
inputB = inb(port2_status);  // reading strength B
input_combined = INPUT_CONVERT(inputA, inputB);
/* STRENGTH 0-100 CONVERSION FACTOR */
MW.strength_local = (input_combined/255)*100;
fprintf(outfile, "Strength A: %i  Strength B: %i Combine: %i\n", inputA, inputB, input_combined);
sprintf(paster, "STRENGTH: %i", input_combined);
gtk_label_set_text ( GTK_LABEL ( MW.strength_label ), paster);

outb( 64, port2_data);
WASTE_TIME();
outb( 80, port2_data);
WASTE_TIME();
outb( 64, port2_data);
WASTE_TIME();
outb( 72, port2_data);
WASTE_TIME();
// printf("Press any key to sample ratio value\n");
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
inputA = inb(port1_status);  // reading ratio A
inputB = inb(port2_status);  // reading ratio B
input_combined = INPUT_CONVERT(inputA, inputB);
/* RATIO 0-100 CONVERSION FACTOR */
MW.ratio_local = (input_combined/255)*100;
fprintf(outfile, "Ratio A: %i  Ratio B: %i  Combine: %i\n", inputA, inputB, input_combined);
sprintf(paster, "RATIO: %i", input_combined);
gtk_label_set_text ( GTK_LABEL ( MW.coffee_ratio_label ), paster);

outb( 192, port2_data);
WASTE_TIME();
outb( 208, port2_data);
WASTE_TIME();
outb( 192, port2_data);
WASTE_TIME();
outb( 200, port2_data);
WASTE_TIME();
// printf("Press any key to sample pressure value\n");
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
inputA = inb(port1_status); // reading pressure A
inputB = inb(port2_status); // reading pressure B
input_combined = INPUT_CONVERT(inputA, inputB);
fprintf(outfile, "Pressure A: %i  Pressure B: %i  Combine: %i\n", inputA, inputB, input_combined);
// sprintf(paster, "PRESSURE:
      %i", input_combined);
// gtk_label_set_text ( GTK_LABEL ( MW.pressure_label ), paster);
/* conversion */
double pressure_final;
pressure_final = 3.075 * input_combined - 7.575;
if ( pressure_final < 0 )
{ sprintf(paster, "PRESSURE:\n      0 psi", pressure_final);}
else
{
    sprintf(paster, "PRESSURE:\n      %.1lf psi", pressure_final);
    FUZZY.pressure = pressure_final;
}
gtk_label_set_text ( GTK_LABEL ( MW.pressure_label ), paster);

outb( 32, port2_data);
WASTE_TIME();
outb( 48, port2_data);
WASTE_TIME();
outb( 32, port2_data);
WASTE_TIME();
outb( 40, port2_data);
WASTE_TIME();
printf("Press any key to sample carafe temp value\n");
// fgets(line, sizeof(line), stdin);
// scanf(line, "%d", &value);
inputA = inb(port1_status);  // reading carafe temp A
inputB = inb(port2_status);  // reading carafe temp B
input_combined = INPUT_CONVERT(inputA, inputB);
fprintf(outfile, "Carafe A: %i  Carafe B: %i  Combine: %i\n", inputA, inputB, input_combined);
// sprintf(paster, "CARAFE HEAT: %i", input_combined);
// gtk_label_set_text ( GTK_LABEL ( MW.carafe_heat_label ), paster);
double carafe_heat_final = 3.21 * input_combined - 415.89;
FUZZY.carafe_heat = carafe_heat_final;
sprintf(paster, "CARAFE HEAT: %3.1lf deg. F", carafe_heat_final);
gtk_label_set_text ( GTK_LABEL ( MW.carafe_heat_label ), paster);
outb( 160, port2_data);
WASTE_TIME();
outb( 176, port2_data);
WASTE_TIME();
outb( 160, port2_data);
WASTE_TIME();
outb( 168, port2_data);
WASTE_TIME();
// printf("Press any key to sample tank value\n");
// fgets(line, sizeof(line), stdin);
// scanf(line, "%d", &value);
inputA = inb(port1_status);  // reading tank temp A
inputB = inb(port2_status);  // reading tank temp B
input_combined = INPUT_CONVERT(inputA, inputB);
fprintf(outfile, "Tank A: %i  Tank B: %i  Combine: %i\n", inputA, inputB, input_combined);
// sprintf(paster, "TANK HEAT: %i", input_combined);
// gtk_label_set_text ( GTK_LABEL ( MW.tank_heat_label ), paster);
double tank_heat_final = 3.66 * input_combined - 483.29;
FUZZY.tank_heat = tank_heat_final;
sprintf(paster, "TANK HEAT: %3.1lf deg. F", tank_heat_final);
gtk_label_set_text ( GTK_LABEL ( MW.tank_heat_label ), paster);
outb( 2, port2_data);
WASTE_TIME();
// printf("Press any key to sample start value\n");
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
inputA = inb(port1_status); // reading start A
inputB = inb(port2_status); // reading start B
input_combined = INPUT_CONVERT(inputA, inputB);
fprintf(outfile, "Start A: %i  Start B: %i  Combine: %i\n", inputA, inputB, input_combined);
sprintf(paster, "START: %i", input_combined);
gtk_label_set_text(GTK_LABEL(MW.start_label), paster);
INV_VAR.start = input_combined;
outb( 6, port2_data);
WASTE_TIME();
// printf("Press any key to sample wash value\n");
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
inputA = inb(port1_status); // reading wash A
inputB = inb(port2_status); // reading wash B
input_combined = INPUT_CONVERT(inputA, inputB);
fprintf(outfile, "Wash A: %i  Wash B: %i  Combine: %i\n", inputA, inputB, input_combined);
sprintf(paster, "WASH: %i", input_combined);
gtk_label_set_text(GTK_LABEL(MW.wash_label), paster);
INV_VAR.wash = input_combined;
WASTE_TIME();
outb( 4, port2_data);
WASTE_TIME();
// printf("Press any key to sample switch value\n");
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
inputA = inb(port1_status); // reading switch A
inputB = inb(port2_status); // reading switch B
input_combined = INPUT_CONVERT(inputA, inputB);
fprintf(outfile, "Switch A: %i  Switch B: %i  Combine: %i\n", inputA, inputB, input_combined);
FUZZY.switch_value = input_combined;
// sprintf(paster, "SWITCH: %i", input_combined);
// gtk_label_set_text(GTK_LABEL(MW.switch_heat_label), paster);

// printf("Please clear buttons\n");
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
outb( 0, port2_data);

// Heating coil management
calculate_heat_control_vector();

// System finished.
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
ACTIVATE_OFF();  // clear all lines
fprintf(outfile, "Ending main loop\n");
return 1;
}

/*************************************************************/

int main( gint argc, gchar * argv[] )
{

int main_loop();

outfile = fopen("test_data.txt", "w");
fprintf(outfile, "-------------------- COFFEE FEEDBACK --------------------\n");

if ( INV_VAR.start > 200 )
{
    command_location = 0;
    COFFEE_ACTIVATE();
    outb( P2C0, port2_ctrl);
    WASTE_TIME();
}

else if ( INV_VAR.wash > 200 )
{
    command_location = 0;
    WASH_ACTIVATE();
    outb( P2C0, port2_ctrl);
    WASTE_TIME();
}

// printf("System finished.\n");
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);

if (SDL_Init(SDL_INIT_AUDIO) != 0) {
    printf("Unable to initialize SDL: %s\n", SDL_GetError());
    return 1;
}

desired.freq = 44100;       /* desired output sample rate */
desired.format = AUDIO_S16; /* request signed 16-bit samples */
desired.samples = 4096;     /* this is somewhat arbitrary */
desired.channels = 1;       /* ask for stereo */
desired.callback = AudioCallback;
desireduserdata = NULL;    /* I don't need this */
if (SDL_OpenAudio(&desired, &obtained) < 0) {
    printf("Unable to open audio device: %s\n", SDL_GetError());
    return 1;
}

/* LOAD SDL FORMAT WAV FILES INTO ALLOCATED MEMORY SPACE */
if (LoadAndConvertSound("sound1.wav",
                        &obtained, &sound1) != 0) {
    printf("Unable to load sound.\n\n");
    return 1;
}

if (LoadAndConvertSound("sound2.wav", &obtained,
                        &sound2) != 0) {
    printf("Unable to load sound.\n\n");
    return 1;
}

if (LoadAndConvertSound("sound3.wav",
                        &obtained, &sound3) != 0) {
    printf("Unable to load sound.\n\n");
    return 1;
}

if (LoadAndConvertSound("sound4.wav",
                        &obtained, &sound4) != 0) {
    printf("Unable to load sound.\n\n");
    return 1;
}

if (LoadAndConvertSound("sound5.wav",
                        &obtained, &sound5) != 0) {
    printf("Unable to load sound.\n\n");
}
return 1;
}

if (LoadAndConvertSound("sound6.wav",
    &obtained, &sound6) != 0) {
    printf("Unable to load sound\n");
    return 1;
}

if (LoadAndConvertSound("sound7.wav",
    &obtained, &sound7) != 0) {
    printf("Unable to load sound\n");
    return 1;
}

if (LoadAndConvertSound("sound8.wav",
    &obtained, &sound8) != 0) {
    printf("Unable to load sound\n");
    return 1;
}

if (LoadAndConvertSound("sound8.wav",
    &obtained, &sound8) != 0) {
    printf("Unable to load sound\n");
    return 1;
}

if (LoadAndConvertSound("sound10.wav",
    &obtained, &sound10) != 0) {
    printf("Unable to load sound\n");
    return 1;
}

ClearPlayingSounds();
SDL_PauseAudio(0);

gtk_init ( &argc, &argv );

if ((ioperm(port1_data,1,1))&(ioperm(port2_data,1,1))&(ioperm(port1_ctrl,1,1))&(ioperm(port2_ctrl,1,1))&(ioperm(port1_status,1,1))&(ioperm(port2_status,1,1)))
{
    printf("Failure on file descriptor initialization\n");
exit(1);
}

ACTIVATE_OFF ();
interface();

main_loop_tag = gtk_idle_add( main_loop, NULL );
gtk_main ();
return 0;
}

#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
#include <gtk/gtk.h>
#include <sys/io.h>
// #include "utils.h"

void COFFEE_ACTIVATE ();
void WASH_ACTIVATE ();

struct input_variables
{
    int start;
    int wash;
    int cups;
    int strength;
    int ratio;
    int pressure;
    int tank_temp;
    int carafe_temp;
    int heat;
int start_percentage;
int wash_percentage;
int cups_percentage;
int strength_percentage;
int ratio_percentage;
int pressure_percentage;
int tank_temp_percentage;
int carafe_temp_percentage;
int heat_percentage;

int initial_run;
} INV_VAR; // end of input_variables struct
```c
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
#include <gtk/gtk.h>
#include <sys/io.h>
#include "coffee.h"
#include "globdefs.h"
#include "interface.h"
#include "sound.h"
#include "utils.h"
#include "gtkdial.h"

void ACTIVATE_SOUND_A();
void ACTIVATE_SOUND_B();
void ACTIVATE_SOUND_C();
void CloseTheApp ( GtkWidget * main_window, gpointer data );

void cup_value_changed( GtkAdjustment *adjustment, GtkWidget *label )
{
    char buffer[16];

    MW.cup_remote = adjustment->value;
    sprintf(buffer, "%4.2f", adjustment->value);
    gtk_label_set_text (GTK_LABEL (label), buffer);
}

void strength_value_changed( GtkAdjustment *adjustment, GtkWidget *label )
{
    char buffer[16];

    MW.strength_remote = adjustment->value;
    sprintf(buffer, "%4.2f", adjustment->value);
    gtk_label_set_text (GTK_LABEL (label), buffer);
}

void ratio_value_changed( GtkAdjustment *adjustment, GtkWidget *label )
{
    char buffer[16];

    MW.ratio_remote = adjustment->value;
    sprintf(buffer, "%4.2f", adjustment->value);
    gtk_label_set_text (GTK_LABEL (label), buffer);
}
```
gtk_label_set_text ( GTK_LABEL (label), buffer);
}

interface()
{
  INV_VAR.initial_run = 0;
  MW.main_window = gtk_window_new ( GTK_WINDOW_TOLEVEL ); // POPUP
  gtk_window_set_title ( GTK_WINDOW ( MW.main_window ), "COFFEE INTERFACE");
  gtk_window_set_default_size ( GTK_WINDOW ( MW.main_window ), 780, 540 ); // 800, 600
  gtk_widget_set_uposition (MW.main_window, 0, 0 );

  gtk_signal_connect ( GTK_OBJECT ( MW.main_window ),
    "destroy", GTK_SIGNAL_FUNC ( CloseTheApp ), NULL );

  GdkColor background_color = {0, 0xffff, 0xffff, 0xffff};
  GtStyle *background;
  background = gtk_widget_get_style (MW.main_window);
  background->bg[GTK_STATE_NORMAL] = background_color;
  gtk_widget_set_style (MW.main_window, background);
  gtk_style_unref (background);

  MW.main_hold_box = gtk_hbox_new ( FALSE, 0 );
  MW.feedback_box = gtk_vbox_new ( FALSE, 0 );
  MW.feedback_box1 = gtk_hbox_new ( FALSE, 20 );
  MW.interface_box = gtk_vbox_new ( FALSE, 0 );
  MW.dial_box = gtk_hbox_new ( FALSE, 0 );
  MW.button_box = gtk_hbox_new ( TRUE, 0 );
  MW.button_box1 = gtk_vbox_new ( TRUE, 30 );

  // feedback box
  MW.pressure_label = gtk_label_new ( "PRESSURE:
");
  MW.tank_heat_label = gtk_label_new ( "TANK HEAT:
");
  MW.carafe_heat_label = gtk_label_new ( "CARAFE HEAT:
");
  MW.status_label = gtk_label_new ( "STATUS:
READY"");

  // gtk_misc_set_alignment (GTK_MISC (MW.pressure_label), 0, 0);
  // gtk_misc_set_alignment (GTK_MISC (MW.tank_heat_label), 0, 0);
  // gtk_misc_set_alignment (GTK_MISC (MW.carafe_heat_label), 0, 0);

  gtk_box_pack_start ( GTK_BOX ( MW.feedback_box ), MW.pressure_label, TRUE, TRUE, 5 );
gtk_box_pack_start (GTK_BOX (MW.feedback_box), MW.tank_heat_label, TRUE, TRUE, 5);
gtk_box_pack_start (GTK_BOX (MW.feedback_box), MW.carafe_heat_label, TRUE, TRUE, 5);
gtk_box_pack_start (GTK_BOX (MW.feedback_box), MW.status_label, TRUE, TRUE, 5);

// dial box
// cups
MW.cup_vbox = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (MW.dial_box), MW.cup_vbox);
MW.cup_dial_title_label = gtk_label_new ("CUPS");
gtk_box_pack_start (GTK_BOX (MW.cup_vbox), MW.cup_dial_title_label, 0, 0, 0);
MW.cup_dial_fuzzy_label = gtk_label_new ("0");
gtk_box_pack_start (GTK_BOX (MW.cup_vbox), MW.cup_dial_fuzzy_label, 0, 0, 0);
MW.cup_frame = gtk_frame_new (NULL);
gtk_container_add (GTK_CONTAINER (MW.cup_vbox), MW.cup_frame);
MW.cup_adjustment = GTK_ADJUSTMENT (gtk_adjustment_new (0, 0, 100, 0.01, 0.1, 0));
MW.cup_dial = gtk_dial_new (MW.cup_adjustment);

background = gtk_style_get_theme (MW.cup_dial);
background->bg[GTK_STATE_NORMAL] = background_color;
gtk_style_set_theme (MW.cup_dial, background);
gtk_style_unref (background);

MW.cup_dial_label = gtk_label_new ("0.00");
gtk_box_pack_start (GTK_BOX (MW.cup_vbox), MW.cup_dial_label, 0, 0, 0);
gtk_container_add (GTK_CONTAINER (MW.cup_frame), MW.cup_dial);
g_signal_connect (G_OBJECT (MW.cup_adjustment), "value_changed",
G_CALLBACK (cup_value_changed), (gpointer) MW.cup_dial_label);

// strength
MW.strength_vbox = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (MW.dial_box), MW.strength_vbox);
MW.strength_dial_title_label = gtk_label_new ("STRENGTH");
gtk_box_pack_start (GTK_BOX (MW.strength_vbox), MW.strength_dial_title_label, 0, 0, 0);
MW.strength_dial_fuzzy_label = gtk_label_new ("0");
gtk_box_pack_start (GTK_BOX (MW.strength_vbox), MW.strength_dial_fuzzy_label, 0, 0, 0);
MW.strength_frame = gtk_frame_new (NULL);
gtk_container_add (GTK_CONTAINER (MW.strength_vbox), MW.strength_frame);
MW.strength_adjustment = GTK_ADJUSTMENT (gtk_adjustment_new (0, 0, 100, 0.01, 0.1, 0));
MW.strength_dial = gtk_dial_new (MW.strength_adjustment);
background = gtk_widget_get_style (MW.strength_dial);
background->bg[GTK_STATE_NORMAL] = background_color;
gtk_widget_set_style (MW.strength_dial, background);
gtk_style_unref (background);
gtk_box_pack_start (GTK_BOX (MW.strength_vbox), MW.strength_dial_fuzzy_label, 0, 0, 0);
MW.strength_dial_label = gtk_label_new ("0.00");
gtk_box_pack_start (GTK_BOX (MW.strength_vbox), MW.strength_dial_label, 0, 0, 0);
g_signal_connect (G_OBJECT (MW.strength_adjustment), "value_changed",
G_CALLBACK (strength_value_changed), (gpointer) MW.strength_dial_label);

// ratio
MW.ratio_vbox = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (MW.dial_box), MW.ratio_vbox);
MW.ratio_dial_title_label = gtk_label_new ("RATIO
");
gtk_box_pack_start (GTK_BOX (MW.ratio_vbox), MW.ratio_dial_title_label, 0, 0, 0);
MW.ratio_dial_fuzzy_label = gtk_label_new ("0" ); // this is the one to update
MW.ratio_frame = gtk_frame_new (NULL);
gtk_container_add (GTK_CONTAINER (MW.ratio_vbox), MW.ratio_frame);
MW.ratio_adjustment = GTK_ADJUSTMENT (gtk_adjustment_new (0, 0, 100, 0.01, 0.1, 0));
MW.ratio_dial = gtk_dial_new (MW.ratio_adjustment);
background = gtk_widget_get_style (MW.ratio_dial);
background->bg[GTK_STATE_NORMAL] = background_color;
gtk_widget_set_style (MW.ratio_dial, background);
gtk_style_unref (background);
gtk_box_pack_start (GTK_BOX (MW.ratio_vbox), MW.ratio_dial_fuzzy_label, 0, 0, 0);
MW.ratio_dial_label = gtk_label_new ("0.00");
gtk_box_pack_start (GTK_BOX (MW.ratio_vbox), MW.ratio_dial_label, 0, 0, 0);
g_signal_connect (G_OBJECT (MW.ratio_adjustment), "value_changed",
G_CALLBACK (ratio_value_changed), (gpointer) MW.ratio_dial_label);

// button box
MW.start_button = gtk_button_new_with_label ("START 

");
gtk_signal_connect ( GTK_OBJECT ( MW.start_button),
"clicked",

83
GTK_SIGNAL_FUNC ( COFFEE_REMOTE_ACTIVATE ), NULL);

MW.wash_button = gtk_button_new_with_label("\n\n       WASH       
\n\n");
gtk_signal_connect ( GTK_OBJECT ( MW.wash_button), "clicked",
GTK_SIGNAL_FUNC ( WASH_REMOTE_ACTIVATE ), NULL);

gtk_box_pack_start (GTK_BOX (MW.button_box), MW.start_button, TRUE, FALSE, 0);
gtk_box_pack_start (GTK_BOX (MW.button_box), MW.wash_button, TRUE, FALSE, 0);

gtk_container_add ( GTK_CONTAINER ( MW.interface_box ), MW.dial_box );
gtk_box_pack_start (GTK_BOX (MW.button_box1), MW.button_box, FALSE, FALSE, 5);
gtk_container_add ( GTK_CONTAINER ( MW.interface_box ), MW.button_box1 );

gtk_box_pack_start (GTK_BOX (MW.main_hold_box), MW.feedback_box, FALSE, TRUE, 5);
gtk_box_pack_start (GTK_BOX (MW.main_hold_box), MW.feedback_box1, FALSE, TRUE, 5);
gtk_container_add ( GTK_CONTAINER ( MW.main_hold_box ), MW.interface_box );
gtk_container_add ( GTK_CONTAINER ( MW.main_window ), MW.main_hold_box );
gtk_widget_show_all ( MW.main_window );

}
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
#include <gtk/gtk.h>
#include <sys/io.h>

struct interface_widgets
{
    GtkWidget * main_window;
    GtkWidget * main_hold_box;
    GtkWidget * feedback_box;
    GtkWidget * feedback_box1;
    GtkWidget * interface_box;
    GtkWidget * dial_box;
    GtkWidget * button_box;
    GtkWidget * button_box1;

    double cup_local;
    double strength_local;
    double ratio_local;

    GtkWidget * start_label;
    GtkWidget * wash_label;
    GtkWidget * tank_heat_label;
    GtkWidget * carafe_heat_label;
    GtkWidget * switch_heat_label;
    GtkWidget * pressure_label;
    GtkWidget * status_label;
    GtkWidget * coffee_ratio_label;
    GtkWidget * cups_label;
    GtkWidget * strength_label;
    GtkWidget * vertical_box;

    double cup_remote;
    GtkAdjustment * cup_adjustment;
    GtkWidget * cup_dial;
    GtkWidget * cup_frame;
    GtkWidget * cup_vbox;
    GtkWidget * cup_dial_label;
    GtkWidget * cup_dial_fuzzy_label;
    GtkWidget * cup_dial_title_label;
}
double strength_remote;
GtkAdjustment * strength_adjustment;
GtkWidget * strength_dial;
GtkWidget * strength_frame;
GtkWidget * strength_vbox;
GtkWidget * strength_dial_label;
GtkWidget * strength_dial_fuzzy_label;
GtkWidget * strength_dial_title_label;

double ratio_remote;
GtkAdjustment * ratio_adjustment;
GtkWidget * ratio_dial;
GtkWidget * ratio_frame;
GtkWidget * ratio_vbox;
GtkWidget * ratio_dial_label;
GtkWidget * ratio_dial_fuzzy_label;
GtkWidget * ratio_dial_title_label;

GtkWidget * start_button;
GtkWidget * wash_button;

} MW; // end of interface widgets
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
#include <gtk/gtk.h>
#include <sys/io.h>

void COFFEE_REMOTE_ACTIVATE();
void WASH_REMOTE_ACTIVATE();
void ACTIVATE_MOTOR();
void ACTIVATE_PUMP();
void ACTIVATE_WATER_FILL();
void ACTIVATE_WATER_CLEAN();
void ACTIVATE_DECAF_HOPPER();
void ACTIVATE_REGULAR_HOPPER();
void ACTIVATE_DECAF_AGITATOR();
void ACTIVATE_REGULAR_AGITATOR();
void ACTIVATE_INDICATOR();
void ACTIVATE_OFF();
void ACTIVATE_SWITCHA();
void ACTIVATE_SWITCHB();
void ACTIVATE_SWITCHC();
void ACTIVATE_SWITCHD();
void ACTIVATE_SWIETCHE();
void ACTIVATE_SWITCHF();
void ACTIVATE_SWITCHG();
void ACTIVATE_SWITCHH();
void TURN_OFF();
void TURN_OFF_PIDATA();
void TURN_OFF_PICTRL();
int WASTE_TIME();
int INPUT_CONVERT(int inputA, int inputB);
int INPUT_PERCENTAGE(int input);
void COFFEE_REMOTE_ACTIVATE()
{
    command_location = 1;
    COFFEE_ACTIVATE();
}

void WASH_REMOTE_ACTIVATE()
{
    command_location = 1;
    WASH_ACTIVATE();
}

void ACTIVATE_MOTOR()
{ outb( P1C1, port1_ctrl); }

void ACTIVATE_PUMP()
{ outb( 5, port1_data); }

void ACTIVATE_WATER_FILL()
{ outb( 1, port1_data); }

void ACTIVATE_WATER_CLEAN()
{ outb( 3, port1_data); }

void ACTIVATE_DECAF_HOPPER()
{ outb( 11, port1_data); }

void ACTIVATE_REGULAR_HOPPER()
{ outb( 9, port1_data); }

void ACTIVATE_DECAF_AGITATOR()
void ACTIVATE_REGULAR_AGITATOR ()
{ outb( P1C3, port1_ctrl); }

void ACTIVATE_INDICATOR ()
{ outb( P1C0, port1_ctrl); }

void ACTIVATE_OFF ()
{
    outb( 0, port2_data);
    outb( 11, port2_ctrl );
}

void ACTIVATE_SWITCHA () // Tank 12 Volt switch
{
    outb( 16, port1_data);
    sleep(1);
    outb( 0, port1_data);
}

void ACTIVATE_SWITCHB () // Tank 24 Volt switch
{
    outb( 48, port1_data);
    sleep(1);
    outb( 0, port1_data);
}

void ACTIVATE_SWITCHC () // Tank 60 Volt switch
{
    outb( 80, port1_data);
    sleep(1);
    outb( 0, port1_data);
}

void ACTIVATE_SWITCHD () // Tank 120 Volt switch
{
    outb( 112, port1_data);
    sleep(1);
    outb( 0, port1_data);
}

void ACTIVATE_SWITCHE () // Carafe 12 Volt switch
{
void ACTIVATE_SWITCHF () // Carafe 24 Volt switch
{
    outb( 176, port1_data);
    sleep(1);
    outb( 0, port1_data);
}

void ACTIVATE_SWITCHG () // Carafe 60 Volt switch
{
    outb( 208, port1_data);
    sleep(1);
    outb( 0, port1_data);
}

void ACTIVATE_SWITCHH () // Carafe 120 Volt switch
{
    outb( 240, port1_data);
    sleep(1);
    outb( 0, port1_data);
}

void TURN_OFF_P1CTRL ()
{
    outb( 11, port1_ctrl);
}

void TURN_OFF_P1DATA ()
{
    outb( 0, port1_data);
}

void TURN_OFF ()
{
    outb( 0, port1_data);
    outb( 11, port1_ctrl);
}

int WASTE_TIME()
{
int waster;
for (waster = 0; waster < 500000; waster++) // 500000
{
   // doing nothing but running a loop to kill time.
}
return 0;

int INPUT_PERCENTAGE ( int input )
{
   int input_in_percent;
   input_in_percent = (input*100)/255;
   return (input_in_percent);
}

int INPUT_CONVERT(int inputA, int inputB)
{
   int input_combined; // combination of two integer inputs
   int A_reformed;
   int B_reformed;
   inputA = inputA-7; // for some reason, P1 is seven off with its integer return.

   if ( inputB == 0 ){
      B_reformed = 0;
   }
   else if (inputB == 8 ){
      B_reformed = 1;
   }
   else if ( inputB == 16 ){
      B_reformed = 2;
   }
   else if (inputB == 24 ){
      B_reformed = 3;
   }
   else if (inputB == 32){
      B_reformed = 4;
   }
   else if (inputB == 40){
      B_reformed = 5;
   }
   else if (inputB == 48){
      B_reformed = 6;
   }
else if (inputB == 56){
    B_reformed = 7;
}
else if (inputB == 64){
    B_reformed = 8;
}
else if (inputB == 72){
    B_reformed = 9;
}
else if (inputB == 80){
    B_reformed = 10;
}
else if (inputB == 88){
    B_reformed = 11;
}
else if (inputB == 96){
    B_reformed = 12;
}
else if (inputB == 104){
    B_reformed = 13;
}
else if (inputB == 112){
    B_reformed = 14;
}
else if (inputB == 120){
    B_reformed = 15;
}
else{
    B_reformed = 9999;
}

if ( inputA == 0 ){
    A_reformed = 0;
}
else if (inputA == 8){
    A_reformed = 16;
}
else if ( inputA == 16 ){
    A_reformed = 32;
}
else if ( inputA == 24 ){
    A_reformed = 48;
}
else if (inputA == 32){

}
A_reformed = 64;
}
else if (inputA == 40){
    A_reformed = 80;
}
else if (inputA == 48){
    A_reformed = 96;
}
else if (inputA == 56){
    A_reformed = 112;
}
else if (inputA == 64){
    A_reformed = 128;
}
else if (inputA == 72){
    A_reformed = 144;
}
else if (inputA == 80){
    A_reformed = 160;
}
else if (inputA == 88){
    A_reformed = 176;
}
else if (inputA == 96){
    A_reformed = 192;
}
else if (inputA == 104){
    A_reformed = 208;
}
else if (inputA == 112){
    A_reformed = 224;
}
else if (inputA == 120){
    A_reformed = 240;
}
else{
    A_reformed = 9999;
}

input_combined = A_reformed + B_reformed;
if (( 0 <= input_combined ) && ( input_combined <= 255)){
    return (input_combined);
}
else{
input_combined = -1;   // error check and avoidance return
return (input_combined);
}
}

#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
#include <gtk/gtk.h>
#include <sys/io.h>

// PORT DEFINITION
#define port1_data 0x3BC       // P1 data port (8)
#define port1_status 0x3BD     // P1 status lines (5)
#define port1_ctrl 0x3BE       // P1 control lines (4)
#define port2_data 0x378       // P2 data port (8)
#define port2_status 0x379     // P2 status lines (5)
#define port2_ctrl 0x37A       // P2 control lines (4)

// OUTPUT DEFINITION
#define P1C0 10                // grounded
#define P1C1 9                 // grounded
#define P1C2 15                // grounded
#define P1C3 3                 // grounded
#define P1D0 1                 // External functions Enable
#define P1D1 2                 // External functions S0
#define P1D2 4                 // External functions S1
#define P1D3 8                 // External functions S2
#define P1D4 16                // Heater Switch Bank Enable
#define P1D5 32                // Heater Switch Bank S0
#define P1D6 64                // Heater Switch Bank S1
#define P1D7 128               // Heater Switch Bank S2

// INPUT DEFINITION
#define P2C0 10                // Push Button Clear
#define P2C1 9                 // grounded
#define P2C2 15                // grounded
#define P2C3 3                 // grounded
#define P2D0 1                 // 74151 S0    DIGITAL
#define P2D1 2                 // 74151 S1    SELECT
#define P2D2 4                 // 74151 S2    MULTIPLEXER
#define P2D3 8       // ADC0804 RD   FLIP
#define P2D4 16      // ADC0804 WR   FLOP
#define P2D5 32      // 4051 S0    ANALOG INPUT
#define P2D6 64      // 4051 S1    SELECT
#define P2D7 128     // 4051 S2    SWITCH

#define MAX_PLAYING_SOUNDS 10
#define VOLUME_PER_SOUND SDL_MIX_MAXVOLUME / 2

extern int inputA;
extern int inputB;
extern int main_loop_tag;
extern int command_location;
extern FILE *outfile;
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
#include <gtk/gtk.h>
#include <sys/io.h>
#include "coffee.h"
#include "globdefs.h"
#include "interface.h"
#include "sound.h"
#include "utils.h"

FILE *outfile;

int inputA;
int inputB;
int main_loop_tag;
int command_location;
#ifndef __GTK_DIAL_H__
define __GTK_DIAL_H__

#include <gdk/gdk.h>
#include <gtk/gtkadjustment.h>
#include <gtk/gtkwidget.h>

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

define GTK_DIAL(obj)          GTK_CHECK_CAST (obj, gtk_dial_get_type (),
GtkDial)
define GTK_DIAL_CLASS(klass)  GTK_CHECK_CLASS_CAST (klass,
gtk_dial_get_type (), GtkDialClass)
define GTK_IS_DIAL(obj)       GTK_CHECK_TYPE (obj, gtk_dial_get_type ()
)

typedef struct _GtkDial      GtkDial;
typedef struct _GtkDialClass GtkDialClass;

struct _GtkDial
{
    GtkWidget widget;

    /* update policy (GTK_UPDATE_[CONTINUOUS/DELAYED/DISCONTINUOUS]) */
guint policy : 2;

    /* Button currently pressed or 0 if none */
guint8 button;

    /* Dimensions of dial components */
gint radius;
gint pointer_width;

    /* ID of update timer, or 0 if none */
guint32 timer;

    /* Current angle */
}
gfloat angle;
gfloat last_angle;

/* Old values from adjustment stored so we know when something changes */
gfloat old_value;
gfloat old_lower;
gfloat old_upper;

/* The adjustment object that stores the data for this dial */
GtkAdjustment *adjustment;

};

struct _GtkDialClass
{
    GtkWidgetClass parent_class;
};

GtkWidget*     gtk_dial_new                    (GtkAdjustment *adjustment);
GtkType        gtk_dial_get_type               (void);
GtkAdjustment* gtk_dial_get_adjustment         (GtkDial *dial);
void           gtk_dial_set_update_policy      (GtkDial *dial,
                                                 GtkUpdateType policy);

void           gtk_dial_set_adjustment         (GtkDial *dial,
                                                 GtkAdjustment *adjustment);

#ifdef __cplusplus
}
#endif /* __cplusplus */
#endif /* __GTK_DIAL_H__ */
#ifndef __GTK_DIAL_H__
#define __GTK_DIAL_H__

#include <gdk/gdk.h>
#include <gtk/gtkadjustment.h>
#include <gtk/gtkwidget.h>

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#define GTK_DIAL(obj)          GTK_CHECK_CAST (obj, gtk_dial_get_type (),
GtkDial)
#define GTK_DIAL_CLASS(klass)  GTK_CHECK_CLASS_CAST (klass,
gtk_dial_get_type (), GtkDialClass)
#define GTK_IS_DIAL(obj)       GTK_CHECK_TYPE (obj, gtk_dial_get_type ())

typedef struct _GtkDial        GtkDial;
typedef struct _GtkDialClass   GtkDialClass;

struct _GtkDial
{
    GtkWidget widget;

    /* update policy (GTK_UPDATE_[CONTINUOUS/Delayed/DISCONTINUOUS]) */
    guint policy : 2;

    /* Button currently pressed or 0 if none */
    guint8 button;

    /* Dimensions of dial components */
    gint radius;
    gint pointer_width;

    /* ID of update timer, or 0 if none */
    guint32 timer;

#ifdef __cplusplus}
#endif /* __cplusplus */
/* Current angle */
gfloat angle;
gfloat last_angle;

/* Old values from adjustment stored so we know when something changes */
gfloat old_value;
gfloat old_lower;
gfloat old_upper;

/* The adjustment object that stores the data for this dial */
GtkAdjustment *adjustment;
}

GtkWidget* gtk_dial_new (GtkAdjustment *adjustment);
GtkType gtk_dial_get_type (void);
GtkAdjustment* gtk_dial_get_adjustment (GtkDial *dial);
void gtk_dial_set_update_policy (GtkDial *dial,
                               GtkUpdateType policy);

void gtk_dial_set_adjustment (GtkDial *dial,
                              GtkAdjustment *adjustment);

#ifdef __cplusplus
}
#endif /* __cplusplus */
#endif /* __GTK_DIAL_H__ */
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
#include <gtk/gtk.h>
#include <sys/io.h>

extern void calc_fuzzy_cups ( int input_cups );
extern void calculate_total_dispense_time ();
extern void calculate_ratio_times ();
extern void calculate_water_dispense_time ();
//extern void calculate_tank_heat_vector ();
//extern void calculate_carafe_heat_vector ();
extern void calculate_heat_control_vector ();

struct fuzzy_variables
{
    double total_dispense_time;
    double regular_dispense_time;
    double decaf_dispense_time;
    double water_dispense_time;
    double tank_heat;
    double carafe_heat;
    double pressure;
    int switch_value;
}

FUZZY; // end of fuzzy_variables struct
CALCULATE_TOTAL_DISPENSE_TIME

MISO FUZZY CALCULATOR

Input
- Cups (0-100)
- Strength (0-100)

Output
- total_dispense_time

void calculate_total_dispense_time ()
{
  double total_disp_output [3][6];
  double u [1][6];
  double v [3][1];
  double c [6][3];
  double I1R [1][3];
  double I1RI2 [1][3];
  double total_dispense_time;

  /* RULE BASE */
  c[0][0] = 3; c[0][1] = 7; c[0][2] = 11;
  c[1][0] = 15; c[1][1] = 19; c[1][2] = 23;
  c[2][0] = 4; c[2][1] = 8; c[2][2] = 12;
  c[3][0] = 16; c[3][1] = 20; c[3][2] = 24;
c[4][0] = 5; c[4][1] = 9; c[4][2] = 13;
c[5][0] = 17; c[5][1] = 21; c[5][2] = 25;

/**************** CUPS FUZZIFIER **********************/
u[0][0] = 0; // INITIALIZE FUZZY CUP INPUT MATRIX
u[1][0] = 0;
u[2][0] = 0;
u[3][0] = 0;
u[4][0] = 0;
u[5][0] = 0;
// MEMBERSHIP A LEFT SIDE
if ( FUZZY.cups_input < 14.286 )
{
    u[0][0] = 1;
}
// MEMBERSHIP A RIGHT SIDE, MEMBERSHIP B LEFT SIDE
else if (( FUZZY.cups_input >= 14.286 ) && ( FUZZY.cups_input < 28.572 ))
{
    u[0][0] = (-1/14.286)*FUZZY.cups_input + 1;
    u[0][1] = (1/14.286)*FUZZY.cups_input - 1;
}
// MEMBERSHIP B RIGHT SIDE, MEMBERSHIP C LEFT SIDE
else if (( FUZZY.cups_input >= 28.572 ) && ( FUZZY.cups_input < 42.858 ))
{
    u[0][1] = (-1/14.286)*FUZZY.cups_input + 2;
    u[0][2] = (1/14.286)*FUZZY.cups_input - 2;
}
// MEMBERSHIP C RIGHT SIDE, MEMBERSHIP D LEFT SIDE
else if (( FUZZY.cups_input >= 42.858 ) && ( FUZZY.cups_input < 57.144 ))
{
    u[0][2] = (-1/14.286)*FUZZY.cups_input + 3;
    u[0][3] = (1/14.286)*FUZZY.cups_input - 3;
}
// MEMBERSHIP D RIGHT SIDE, MEMBERSHIP E LEFT SIDE
else if (( FUZZY.cups_input >= 57.144 ) && ( FUZZY.cups_input < 71.43 ))
{
    u[0][3] = (-1/14.286)*FUZZY.cups_input + 4;
    u[0][4] = (1/14.286)*FUZZY.cups_input - 4;
}
// MEMBERSHIP E RIGHT SIDE, MEMBERSHIP F LEFT SIDE
else if (( FUZZY.cups_input >= 71.43 ) && ( FUZZY.cups_input < 85.716 ))
{
    u[0][4] = (-1/14.286)*FUZZY.cups_input + 5;
    u[0][5] = (1/14.286)*FUZZY.cups_input - 5;
else if (( FUZZY.cups_input >= 85.716 ) && ( FUZZY.cups_input <= 102 ))
{
    u[0][5] = 1;
}

/******************* STRENGTH FUZZIFIER *******************/

// INITIALIZE FUZZY STRENGTH INPUT MATRIX
v[0][0] = 0;
v[1][0] = 0;
v[2][0] = 0;

else if (( FUZZY.strength_input < 25 )
{
    v[0][0] = 1;
}

else if (( FUZZY.strength_input >= 25 ) && ( FUZZY.strength_input < 50 ))
{
    v[0][0] = (-1/25)*FUZZY.strength_input + 2;
    v[1][0] = (1/25)*FUZZY.strength_input - 2;
}

else if (( FUZZY.strength_input >= 50 ) && ( FUZZY.strength_input < 75 ))
{
    v[1][0] = (-1/25)*FUZZY.strength_input + 3;
    v[2][0] = (1/25)*FUZZY.strength_input - 3;
}

else if (( FUZZY.strength_input >= 75 ) && ( FUZZY.strength_input < 102 ))
{
    v[3][0] = 1;
}

/* Multiply I1*C */

I1R[0][0] = u[0][0]*c[0][0] +
           u[0][1]*c[1][0] +
           u[0][2]*c[2][0] +
           u[0][3]*c[3][0] +
           u[0][4]*c[4][0] +
           u[0][5]*c[5][0];
I1R[0][1] = u[0][0]*c[0][1] +
    u[0][1]*c[1][1] +
    u[0][2]*c[2][1] +
    u[0][3]*c[3][1] +
    u[0][4]*c[4][1] +
    u[0][5]*c[5][1];

I1R[0][2] = u[0][0]*c[0][2] +
    u[0][1]*c[1][2] +
    u[0][2]*c[2][2] +
    u[0][3]*c[3][2] +
    u[0][4]*c[4][2] +
    u[0][5]*c[5][2];

I1R[0][0] = I1R[0][0]*v[0][0];
I1R[0][1] = I1R[0][1]*v[1][0];
I1R[0][2] = I1R[0][2]*v[2][0];

total_dispense_time = I1R[0][0]+I1R[0][1]+I1R[0][2];
FUZZY.total_dispense_time = total_dispense_time;

} // end calculate_total_dispense_time

писать_радио_время

MISO FUZZY CALCULATOR

Input
- total_dispense_time
- ratio (0-100)

Output
- regular dispenser time
- decaf dispenser time

void calculate_ratio_times ()
{
    double u[0][1];

    u[0][0] = (1/100)*FUZZY.ratio_input;
    u[0][1] = (-1/100)*FUZZY.ratio_input + 1;
FUZZY.regular_dispense_time = u[0][0] * FUZZY.total_dispense_time;
FUZZY.decaf_dispense_time = u[0][1] * FUZZY.total_dispense_time;
}

/******************************************************************************

CALCULATE_WATER_DISPENSE_TIME

MISO

Input
-pressure (measured PSI)
-cups (0-100)

******************************************************************************/
void calculate_water_dispense_time ()
{
  double cups, time, i;
  if ( command_location == 0 ) // Activation came from local user interface
  {
    cups = MW.cup_local;
  }
  else // Activation came from remote user interface
  {
    cups = MW.cup_remote;
  }

  ACTIVATE_WATER_FILL (); // Open water supply solenoid

  if ( cups < 16.7 ) // 2 cups
  {
    time = 450000000; // 500000000
    for (i = 0; i < time; ++i){}
  }
  else if ( 16.7 >= cups || cups < 33.36 ) // 4 cups
  {
    time = 600000000;
    for (i = 0; i < time; ++i){}
  }
  else if ( 33.36 >= cups || cups < 50.06 ) // 6 cups
  {
    time = 1350000000;
    for (i = 0; i < time; ++i){}
  }
else if ( 50.06 >= MW.cup_remote || MW.cup_remote < 66.7 ) // 8 cups
{
    time = 1800000000;
    for (i = 0; i < time; ++i){}
}
else if ( 66.7 >= MW.cup_remote || MW.cup_remote < 83.46 ) // 10 cups
{
    time = 1125000000; //2250000000;
    for (i = 0; i < time; ++i){}
    for (i = 0; i < time; ++i){}
}
else if ( 83.46 >= MW.cup_remote || MW.cup_remote < 100 ) // 12 cups
{
    time = 900000000; //2700000000;
    for (i = 0; i < time; ++i){}
    for (i = 0; i < time; ++i){}
    for (i = 0; i < time; ++i){}
}

//ACTIVATE_WATER_FILL (); // Open water supply solenoid
//for (i = 0; i < time; ++i){}
TURN_OFF_P1DATA ();  // Close water supply solenoid
ACTIVATE_SOUND7 ();
}

//***************************************************************************

CALCULATE_HEAT_CONTROL_VECTOR

SISO

Input
-heat sensor information - in degrees Fahrenheit
-switch input

TARGET TEMPERATURE CARAFE - 160 degrees Fahrenheit
TARGET TEMPERATURE TANK   - 200 degrees Fahrenheit

HEATER CONTROL MANAGEMENT
Possible combinations without fault:
DATA BITS    CARAFE (V)  TANK (V)
0 + 0 = 0    0  0
1 + 0 = 1    120  0
void calculate_heat_control_vector ()
{
    int carafe_current, tank_current;
    int carafe_new, tank_new;
    double CA,CB,CC,CD;
    double TA,TB,TC,TD;

    // boundary definitions
    CA = 130; // temperature barrier between 120 and 60 volt setting
    CB = 140; // temperature barrier between 60 and 24 volt setting
    CC = 155; // temperature barrier between 24 and 12 volt setting
    CD = 165; // temperature barrier between 12 volt and OFF setting

    TA = 180; // temperature barrier between 120 and 60 volt setting
    TB = 190; // temperature barrier between 60 and 24 volt setting
    TC = 197; // temperature barrier between 24 and 12 volt setting
    TD = 203; // temperature barrier between 12 volt and OFF setting
/* Calculate what switches are currently on */

if ( FUZZY.switch_value == 0 )
{
    carafe_current = 0;
    tank_current = 0;
}
else if ( FUZZY.switch_value == 1 )
{
    carafe_current = 120;
    tank_current = 0;
}
else if ( FUZZY.switch_value == 2 )
{
    carafe_current = 60;
    tank_current = 0;
}
else if ( FUZZY.switch_value == 4 )
{
    carafe_current = 24;
    tank_current = 0;
}
else if ( FUZZY.switch_value == 8 )
{
    carafe_current = 12;
    tank_current = 0;
}
else if ( FUZZY.switch_value == 16 )
{
    carafe_current = 0;
    tank_current = 120;
}
else if ( FUZZY.switch_value == 32 )
{
    carafe_current = 0;
    tank_current = 60;
}
else if ( FUZZY.switch_value == 64 )
{
    carafe_current = 0;
    tank_current = 24;
}
else if ( FUZZY.switch_value == 128 )
{

carafe_current = 0;
tank_current = 12;
}
else if (FUZZY.switch_value == 17)
{
carafe_current = 120;
tank_current = 120;
}
else if (FUZZY.switch_value == 33)
{
carafe_current = 120;
tank_current = 60;
}
else if (FUZZY.switch_value == 65)
{
carafe_current = 120;
tank_current = 24;
}
else if (FUZZY.switch_value == 129)
{
carafe_current = 120;
tank_current = 12;
}
else if (FUZZY.switch_value == 18)
{
carafe_current = 60;
tank_current = 120;
}
else if (FUZZY.switch_value == 34)
{
carafe_current = 60;
tank_current = 60;
}
else if (FUZZY.switch_value == 66)
{
carafe_current = 60;
tank_current = 24;
}
else if (FUZZY.switch_value == 130)
{
carafe_current = 60;
tank_current = 12;
}
else if (FUZZY.switch_value == 20)
```c
{  
carafe_current = 24;  
tank_current = 120;  
}
else if ( FUZZY.switch_value == 36 )  
{
  carafe_current = 24;  
tank_current = 60;  
}
else if ( FUZZY.switch_value == 68 )  
{
  carafe_current = 24;  
tank_current = 24;  
}
else if ( FUZZY.switch_value == 132 )  
{
  carafe_current = 24;  
tank_current = 12;  
}
else if ( FUZZY.switch_value == 24 )  
{
  carafe_current = 12;  
tank_current = 120;  
}
else if ( FUZZY.switch_value == 40 )  
{
  carafe_current = 12;  
tank_current = 60;  
}
else if ( FUZZY.switch_value == 72 )  
{
  carafe_current = 12;  
tank_current = 24;  
}
else if ( FUZZY.switch_value == 136 )  
{
  carafe_current = 12;  
tank_current = 12;  
}
else
{
  // I'm really not sure what to do in the event of an unknown
  printf("Unknown scenario found at \%i\n", FUZZY.switch_value );
  CloseTheApp ();
}
```
if ( FUZZY.carafe_heat <= CA )
    { carafe_new = 120; }
else if (( FUZZY.carafe_heat > CA ) && ( FUZZY.carafe_heat <= CB ))
    { carafe_new = 60; }
else if (( FUZZY.carafe_heat > CB ) && ( FUZZY.carafe_heat <= CC ))
    { carafe_new = 24; }
else if (( FUZZY.carafe_heat > CC ) && ( FUZZY.carafe_heat <= CD ))
    { carafe_new = 12; }
else if ( FUZZY.carafe_heat > CD )
    { carafe_new = 0; }
*/
// Decipher what membership the carafe output should be in.
if ( FUZZY.carafe_heat <= 158 )
    { carafe_new = 120; }
else if (( FUZZY.carafe_heat > 158 ) && ( FUZZY.carafe_heat <= 161 ))
    { carafe_new = 12; }
else
    { carafe_new = 0; }

// Decipher what membership the tank output should be in.
if ( FUZZY.tank_heat <= 198 )
    { tank_new = 120; }
else if (( ( FUZZY.tank_heat > 198 ) && ( FUZZY.tank_heat < 201 )))
    { tank_new = 12; }
else
    { tank_new = 0; }

if ( INV_VAR.system_down == 1 )
{
    carafe_new = 0;
    tank_new = 0;
}

// printf("tank temp is %lf and tank new is %d\n",FUZZY.tank_heat, tank_new);
// fgets(line, sizeof(line), stdin);
// sscanf(line, "%d", &value);
// Compare the old with the new
if ( FUZZY.heat_control_pause == 0 )
{
    if ( carafe_current != carafe_new )
    {
        FUZZY.heat_control_pause++;
        INV_VAR.initial_run = 0; // reset input lines to prevent mis-triggers
        // turn off current and turn on new based on carafe_new
        if ( carafe_current == 0 )
        {
            // no need to turn anything off... everything already is
        }
        else if ( carafe_current == 12 )
        {
            fprintf(outfile, "TURN OFF CARAFE: 12V\n");
            fprintf(carafe_file, "TURN OFF CARAFE: 12V\n");
            ACTIVATE_SWITCHE();
        }
        else if ( carafe_current == 24 )
        {
            fprintf(outfile, "TURN OFF CARAFE: 24V\n");
            fprintf(carafe_file, "TURN OFF CARAFE: 24V\n");
            ACTIVATE_SWITCHF();
        }
        else if ( carafe_current == 60 )
        {
            fprintf(outfile, "TURN OFF CARAFE: 60V\n");
            fprintf(carafe_file, "TURN OFF CARAFE: 60V\n");
            ACTIVATE SWITCHG();
        }
        else if ( carafe_current == 120 )
        {
            fprintf(outfile, "TURN OFF CARAFE: 120V\n");
            fprintf(carafe_file, "TURN OFF CARAFE: 120V\n");
            ACTIVATE SWITCHH();
        }
        // Finally, turn on the new switch
        if ( carafe_new == 0 )
        {
            // don't turn anything back on
        }
        else if ( carafe_new == 12 )
        {
        }
        else if ( carafe_new == 24 )
        {
fprintf(outfile, "ACTIVATE CARAFE: 12V\n");
fprintf(carafe_file, "ACTIVATE CARAFE: 12V\n");
ACTIVATE_SWITCHE();
}
else if (carafe_new == 24)
{
    fprintf(outfile, "ACTIVATE CARAFE: 24V\n");
    fprintf(carafe_file, "ACTIVATE CARAFE: 24V\n");
    ACTIVATE_SWITCHF();
}
else if (carafe_new == 60)
{
    fprintf(outfile, "ACTIVATE CARAFE: 60V\n");
    fprintf(carafe_file, "ACTIVATE CARAFE: 60V\n");
    ACTIVATE_SWITCHG();
}
else if (carafe_new == 120)
{
    fprintf(outfile, "ACTIVATE CARAFE: 120V\n");
    fprintf(carafe_file, "ACTIVATE CARAFE: 120V\n");
    ACTIVATE_SWITCHH();
}
else
{
    // leave it alone -- everything's keen
}

//if (FUZZY.heat_control_pause == 0)
//{
//    if (tank_current != tank_new)
//    {
//        FUZZY.heat_control_pause++; // reset input lines to prevent mis-triggers
//        INV_VAR.initial_run = 0;
//        printf("tank_current does NOT equal tank_new\n");
//        fgets(line, sizeof(line), stdin);
//        sscanf(line, "%d", &value);
//        // turn off current and turn on new based on tank_new
//        if (tank_current == 0)
//        {
//            printf("WANT TO ACTIVATE NOTHING\n");
//            fgets(line, sizeof(line), stdin);
//            sscanf(line, "%d", &value);
//            // no need to turn anything off... everything already is
} else if ( tank_current == 12 )
{
    // printf("WANT TO DEACTIVATE 12 V SWITCH\n");
    // fgets(line, sizeof(line), stdin);
    // sscanf(line, "%d", &value);
    fprintf(outfile, "TURN OFF TANK: 12V\n");
    fprintf(tank_file, "TURN OFF TANK: 12V\n");
    ACTIVATE SWITCHA ();
}
else if ( tank_current == 24 )
{
    // printf("WANT TO DEACTIVATE 24 V SWITCH\n");
    // fgets(line, sizeof(line), stdin);
    // sscanf(line, "%d", &value);
    fprintf(outfile, "TURN OFF TANK: 24V\n");
    fprintf(tank_file, "TURN OFF TANK: 24V\n");
    ACTIVATE SWITCHB ();
}
else if ( tank_current == 60 )
{
    // printf("WANT TO DEACTIVATE 60 V SWITCH\n");
    // fgets(line, sizeof(line), stdin);
    // sscanf(line, "%d", &value);
    fprintf(outfile, "TURN OFF TANK: 60V\n");
    fprintf(tank_file, "TURN OFF TANK: 60V\n");
    ACTIVATE SWITCHC ();
}
else if ( tank_current == 120 )
{
    // printf("WANT TO DEACTIVATE 120 V SWITCH\n");
    // fgets(line, sizeof(line), stdin);
    // sscanf(line, "%d", &value);
    fprintf(outfile, "TURN OFF TANK: 120V\n");
    fprintf(tank_file, "TURN OFF TANK: 120V\n");
    ACTIVATE SWITCHD ();
}
// Finally, turn on the new switch
if ( tank_new == 0 )
{
    // printf("WANT TO ACTIVATE NOTHING\n");
    // fgets(line, sizeof(line), stdin);
    // sscanf(line, "%d", &value);
    // don't turn anything back on
if ( tank_new == 12 ) {
  printf("WANT TO ACTIVATE 12 V SWITCH\n");
  fgets(line, sizeof(line), stdin);
  sscanf(line, "%d", &value);
  fprintf(outfile, "ACTIVATE TANK: 12V\n");
  fprintf(tank_file, "ACTIVATE TANK: 12V\n");
  ACTIVATE_SWITCHA();
}
else if ( tank_new == 24 ) {
  printf("WANT TO ACTIVATE 24 V SWITCH\n");
  fgets(line, sizeof(line), stdin);
  sscanf(line, "%d", &value);
  fprintf(outfile, "ACTIVATE TANK: 24V\n");
  fprintf(tank_file, "ACTIVATE TANK: 24V\n");
  ACTIVATE_SWITCHB();
}
else if ( tank_new == 60 ) {
  printf("WANT TO ACTIVATE 60 V SWITCH\n");
  fgets(line, sizeof(line), stdin);
  sscanf(line, "%d", &value);
  fprintf(outfile, "ACTIVATE TANK: 60V\n");
  fprintf(tank_file, "ACTIVATE TANK: 60V\n");
  ACTIVATE_SWITCHC();
}
else if ( tank_new == 120 ) {
  printf("WANT TO ACTIVATE 120 V SWITCH\n");
  fgets(line, sizeof(line), stdin);
  sscanf(line, "%d", &value);
  fprintf(outfile, "ACTIVATE TANK: 120V\n");
  fprintf(tank_file, "ACTIVATE TANK: 120V\n");
  ACTIVATE_SWITCHD();
}
else {
  // leave it alone -- everything's keen
  printf("tank values match\n");
  fgets(line, sizeof(line), stdin);
  sscanf(line, "%d", &value);
}
} else {

    FUZZY.heat_control_pause++;
    if ( FUZZY.heat_control_pause >= 30 ) {
        FUZZY.heat_control_pause = 0;
    }

} // end of calculate_heat_control_vector
```c
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>

/* Structure for loaded sounds. */
typedef struct sound_s {
    Uint8 *samples; /* raw PCM sample data */
    Uint32 length; /* size of sound data in bytes */
} sound_t, *sound_p;

/* Structure for a currently playing sound. */
typedef struct playing_s {
    int active; /* 1 if this sound should be played */
    sound_p sound; /* sound data to play */
    Uint32 position; /* Current position in the sound buffer */
} playing_t, *playing_p;
```
/*********************************************************************/

GCC makefile

***************************************************************************/

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

main ()
{
    system("gcc globdefs.h globdefs.c fuzzy.h fuzzy.c gtkdial.c gtkdial.h utils.h utils.c sound.h sound.c interface.h interface.c coffee.c coffee.h -o coffee_interface `sdl-config --cflags --libs` `pkg-config gtk+-2.0 --cflags --libs` ");
}
Bibliography


