Auto Brew

Robert Bower, Alonzo Ubilla,
Kleber Valencia, and David Rodriguez

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — This paper presents the design methodology utilized to create an automated home brewing system, that should significantly reduce the amount of labor associated with making beer. The subsystems implemented in this design are as follows: (1) I²C Communications; (2) Process Control; (3) Power Distribution; (4) User Interface; (5) Process Monitoring and Notification.**

***Index Terms* — analog amplifier, automation, beer, graphical user interface, high-power switching, I²C communications, linear power-supply, microcontroller, single-board computer, temperature control.**

I. INTRODUCTION

Home brewing beer can be an enjoyable hobby, however, it can also be very time consuming. The typical brewing time can range from 6 – 10 hours depending on the recipe and brewing equipment used during the process. Most of the time is spent setting up the equipment and transferring liquids. There are 5 basic steps in this process.

1. Mash: During this step grain is soaked in hot water with the goal of leaching the starches out of the grain.

2. Sparge: During this step the grain is rinsed of the starchy liquid (now called wort).

3. Boil: During this step the wort is heated to break down the starches into simple sugars.

4. Cooling: During this step the wort is cooled as quickly as possible to facilitate coagulation of proteins contributed from the grain in the mash step.

5. Fermentation: During this process yeast is added to the wort, and now the solution is called beer.

The expectations of the senior design team were to research, and design a system that would drastically reduce the time needed to produce home brewed beer. The senior design team required that the system must meet the following requirements: (1) Complete the brewing process without intervention from the user; (2) Provide real time data to the user; (3) Operate within defined safety limitations; (4) Keep the user updated with the process status; (5) Provide remote monitoring of the system.

II. SYSTEM OVERVIEW

The following consist of the overall major subsystems for the auto brew. Figure 1 depicts hardware flow for the integration of the subsystems.

A. Graphical User Interface

The automated process being created has to be driven by an initialization phase done by an actual human. This is where the group decided to establish a graphical user interface, and in this case it will be written in Python 2.7.

B. Communication

Communication between the two microcontrollers and the single board computer will be handled using the I²C protocol. The single board computer is the master while the microcontrollers are the slaves with a 3.3V to 5V level converter between the two. Communication between the companion Android application and the single board will be handled via Bluetooth.

C. Data acquisition

The system will constantly be taking this will be handled via the several analog sensors through the system sending information to the microcontroller. The data collected will also be the signals used to control the overall system. Thermistor and weight sensors contribute the most of the control conditions the MCU uses for the process. Once data is collected by the MCU it is then sent to the single board computer, the Raspberry Pi B+, stored for access via the GUI and sent to the companion app for the auto brew via Bluetooth.

D. Power Distribution

Varying voltages, both AC and DC, were used throughout the system, starting with 115 VAC mains for some of the higher power components, 24 VDC for solenoids and relays, 5 VDC for microcontrollers and analog sensors, and finally 3.3 VDC for I²C communication. While simple linear regulating IC's were able to be used for the lower voltage levels, a 24

VDC supply was designed to accommodate the maximum possible load from the 24V components.

E. High Power Switching

Due to the low-power nature of microcontrollers, it was necessary to incorporate a method of high power switching within the system. The system utilizes solenoid valves, a solenoid actuator, a heating element, and fluid transfer pumps which run on either 24VDC or 115VAC mains voltage and range in power consumption from 500mW to 1500 watts, microcontrollers are unable to supply this kind of power. Also, the fact that the components being switched are at a substantially higher potential, some form of isolation would be ideal in order to protect the microcontrollers from possible damage. For these reasons, a circuit that has the ability to switch on high power devices using a low voltage low current signal was designed

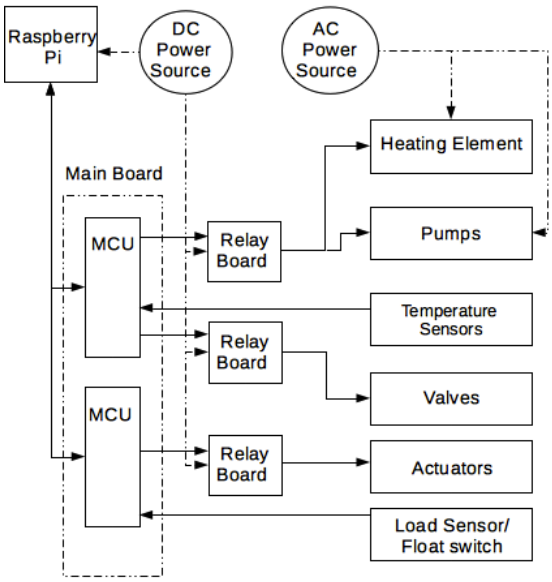


Fig. 1: Overall System Integration

III. USER INTERFACE

The graphical user interface designed for the Auto Brew system was coded using Python 2.7. The user interacts with the interface via a single board computer connected to a monitor and wireless keyboard. For the purpose of this project the Raspberry Pi B+ model was chosen as this single board computer. It runs a Linux based operating system called Raspbian. Comes with 512MB of RAM and utilizes an external 8GB SD card to boot the Raspbian operating system at startup.

Since one of the main objectives was to take out as much of the human interaction as possible for brewing beer at home, giving the user the sense they are still in control of some manual parts of the process was beneficial. The group did not want to create an automated system that was rugged, looked unappealing or unaesthetically pleasing to the home brewer's eyes. Having the interface embrace a sense of flow using eye-catching colors, along with giving the user that initial responsibility of initializing the system, makes for a very functional user interface as shown in the figure below.

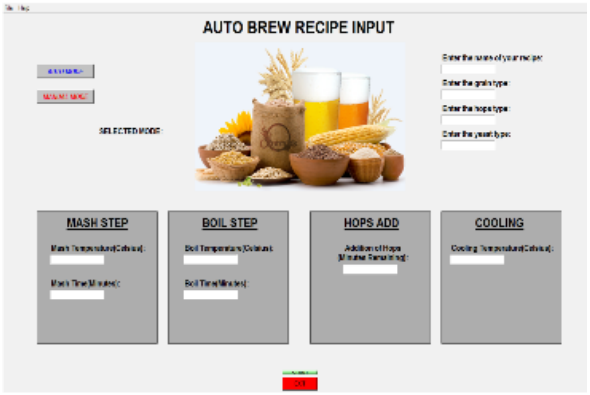


Fig 2: Main graphical user interface

This interface allows the user(s) to efficiently input all of their recipe variables into the system, which will drive all of the decisions made by the microcontrollers. All of the data input by the user will be packaged and sent all together to the control system once the user clicks on the green start button.

It is at this time that a second graphical user interface pops-up and sends a start command signal to the microcontroller to begin step 1 of the process. This second window allows the user to monitor the overall process, from the status of each valve/pump to temperature and level readings. A companion app running on an Android device allows the user to continue to keep tabs on the system while being away from the monitor. This interface communicates with the user's Android application through Bluetooth communication which is initialized once again through a Python script embedded into each of the function calls to update the individual system variables. This second window functions as a supervisor to the system as its running. It tells the microcontrollers when to start each step, feeds system data to the android application and the interface being displayed on the monitor. It is with this interface that the microcontrollers communicate back and forth with the most. This script will run until the last step of the process is complete, in

which case it will send a notification to the user and display a warning message on the monitor to bring attention to them.

IV. COMMUNICATIONS

A. *I²C Configuration*

The senior design team chose to use I²C communication to connect all of the subsystems. The communication system is structured to have one master and two slaves. Figure 2 shows the I²C configuration used for the system.

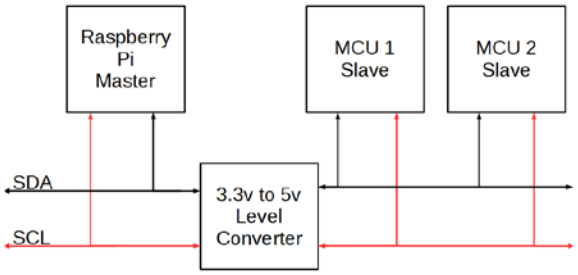


Fig. 2: I²C Communication Bus

Since the operating voltage of the I²C slaves was chosen to be 5V DC, it was necessary for the senior design team to implement a logic level converter. This allowed the team to interface the 5V DC I²C bus with the 3.3V DC I²C bus of the single board computer. Figure 3 shows the schematic of the logic level converter implemented by the senior design team.

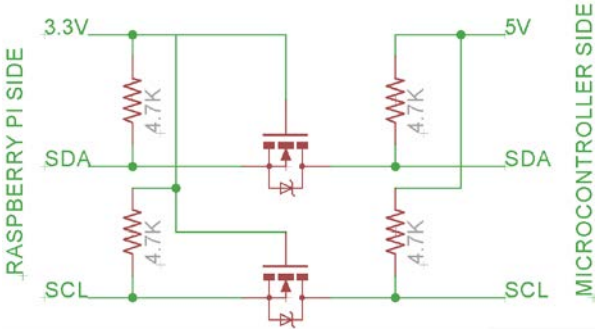


Fig. 3: Logic Level Converter

The logic level converter utilizes two 2N7000 N-Channel MOSFETs to isolate the different voltage levels. The senior design team chose the pull up resistors used for the serial data, and serial clock lines, based the on bus capacitance calculations given by the data sheets for the microcontrollers. For a serial clock frequency of less than 100 kHz, the minimum and maximum values of the pull up resistors were given by:

$$R_{PMIN} = \frac{V_{cc} - 0.4}{3mA} \qquad R_{PMAX} = \frac{1000ns}{C_b}$$

Table 1 shows the approximated range of possible resistor values, given the calculated bus capacitance.

Bus Capacitance (pF)	Minimum Resistor Value (kΩ)	Maximum Resistor Value (kΩ)
22.1	1.5	45

Table 1: Calculated Range for Pull Up Resistors

Through testing different transmission rates, the senior design team determined that 4.7 kΩ seemed to be the optimal resistor value for minimal transmission error.

B. *I²C Implementation*

Once the user has input all of the recipe data in to the GUI, the senior design team needed a way to package the data and send it out from the single board computer to the microcontrollers. It was determined that in order to properly organize the data on both the sending, and receiving ends, the data would need to be packaged into two byte packets. Table 2 shows the structure for each packet sent from the I²C master to each of the I²C slave microcontrollers.

First Byte	Second Byte
Address Call + R/W bit	Variable Data

Table 2: Two Byte Packet Structure

As shown in Table 2, the first byte of each packet contains the address of the destination with the least significant bit being a read or write bit. The second byte contains the actual information the I²C master wants to deliver. Once the I²C master has sent out the packet, the senior design team needed to organize how each of the I²C slaves processed the incoming data. In order to organize all of the data on the slave side, the senior design team declared two classes of data. Table 3 shows each class of data with a brief description.

Class	Description
Command Data	Start/Stop commands, On/Off commands, return specific data, etc...
Variable Data	Temperatures, times, liquid levels, etc...

Table 3: Data Classes

Each of the I^2C slaves processes the data in four byte increments. The senior design team implemented this structure to allow for the retrieval of any data desired, as well as the ability to fully manipulate the hardware in the system. This means that in order to store a variable, or execute some action, two of the aforementioned packets are sent to a slave, each time a command is issued. A command could tell one of the slaves to return a process variable, or it could tell one of the slaves to turn something on or off. Table 4 shows how command data is processed by the I^2C slaves and table 5 shows how variable data is processed by the I^2C slaves.

Byte Number	Hex Value	Action Taken
1	0x10	Address call received, read next byte.
2	0xA0	Turn on Valve Command Received
3	0x10	Address call received, read next byte
4	0x00	Null (Turn on Valve)

Table 4: Processing of Command Data

Byte Number	Hex Value	Action Taken
1	0x10	Address call received, read next byte.
2	0xC0	Prepare to send back process variable
3	0x10	Address call received, read next byte
4	0xE0	Send back temperature of boil kettle

Table 5: Processing of Variable Data

C. Bluetooth Connectivity

The Bluetooth portion of the system is used to send the data collected by the system to the companion app on an Android device for the Auto brew. The reason Bluetooth was implemented instead of just using an internet connection was to ensure that any user could use the app even without internet connectivity.

The Bluetooth communication is one way between the app and GUI once the system has started. The android app was designed to act as a Bluetooth server while the raspberry pi was implemented as a Bluetooth client. The team implemented it this way because it was determined that the app would work only as a way to observe the system while away from it. When the app starts it will start the Bluetooth server on the device while the raspberry pi searches for the device's MAC address and connects to it. Once the connection is established the app applies the data for its own functionality.

The way the app will receive the data is through text files telling it the progress of the system. All the temperature data, fluid levels, and the valve status will be displayed on the app. It will also terminate the connection once the brew is complete. Figure 4 will display the communications flow with the companion app.

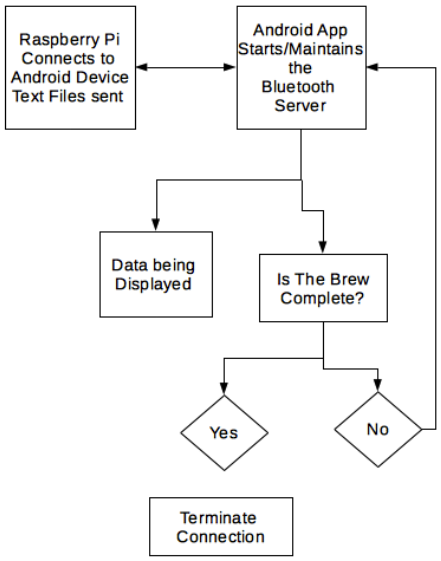


Fig. 4: Bluetooth Communication Flow

D. Push Notification

The push notification method was the best way to alert the user of major step completions. This allows for the user to know the progress of the current brew while away from the system. In order to have this functionality it is assumed there is an available internet connection to the Raspberry Pi. As long as the system has internet connection it will send an email or text notification to the user after the completion of each step. This is done by running a python script to connect to a Gmail server on a specified port and open the connection for the message to get through. The senior design team really took advantage of these Google services and also gave the Auto Brew its own email account to manage the necessary connections. This was necessary since the Python script logs into the email account and verifies a reliable connection to an existing account. This script is implemented each time the graphical user interface is given an update of a process step being completed from the microcontrollers. The SMS notification sent to the user's phone work in a similar fashion. The only prior knowledge needed for this script to function properly is the user's phone carrier and/or email address in order to send the appropriate text message and/or email notification.

V. DATA ACQUISITION

The team focused on using analog temperature and analog load reading to control the process. These readings are also important for the brewer to keep track of and improve recipes. The sensors are powered by 5VDC and amplified for the MCU to read.

The temperature reading are sent through a non-inverting amplifier. The readings are nonlinear voltages and need to be converted for the user to know what the temperatures are. This is done on the MCU and sent over to the microcomputer for the GUI. Once the microcomputer has the data it is sent to the companion app for display and storage of data. Figure 5 illustrates the circuit used for the readings.

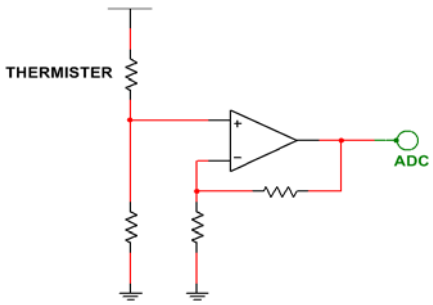


Fig. 5: Thermistor Circuit

The load readings also need to be amplified. The readings are acquired by using two strain gauges in a Whetstone bridge circuit. The signal is sent through a difference amplifier to get a clearer reading of the load on the boil kettle. This was better than just using a non-inverting amplifier because the resistance change is only a few ohms. Once the signal gets to the MCU it is then sent to the sent to the companion app like the temperature readings. Figure 6 is the strain gauge circuit used to measure the fluid level in the boil kettle.

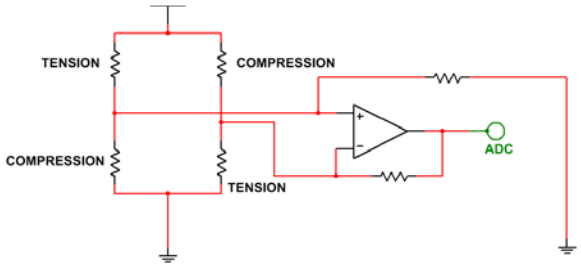


Fig. 5: Load Circuit for Fluid Levels

The single board computer will make request for data from the MCUs controlling the system. When a request is made, previously sampled data is sent back to the single board computer and a new sample routine is initiated. The new sample routine reads the specified ADC and increments a sample counter. It aggregates the sample in a single variable, and once the sample counter reaches ten, it takes an average of the aggregate and stores the data for the next request. Figure 6 demonstrates the routine in a flowchart.

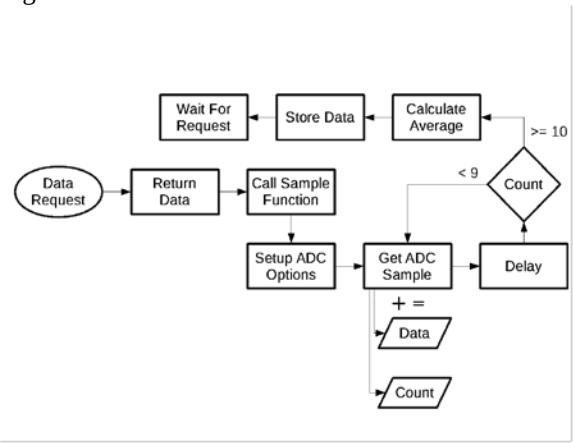


Fig. 6: New Sample Routine

VI. PROCESS CONTROL

A. Control System Configuration

In order to fully automate the beer brewing process, the senior design team needed to implement a control system that would be able to monitor all of the temperatures, times, and liquid levels within the system, as well as control the transfer of liquids to the appropriate locations. The same single board computer, being used for the user interface, was chosen to be the main controller as well. The single board computer is responsible for timing and process flow. Along with the single board computer, the senior design team used two microcontrollers to help share the work load throughout the entire process. Table 6 shows the responsibilities of each device in the control system.

The single board computer sends hexadecimal commands to the microcontroller board throughout the entire process. These hexadecimal commands are used to request data back from the microcontrollers, as well as turn on and turn off devices according to which step of the process the system is executing. Figure 7 shows the flow of liquids throughout the system. The number next to each line indicates the order in which the transfer takes place.

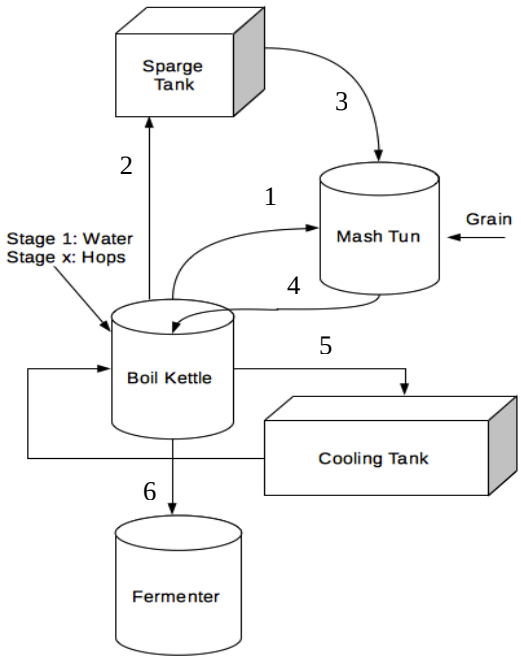


Fig. 7: Fluid Flow Through System

Device	Responsibility
Raspberry Pi	Timing, Process Flow, Information Display, Send Notifications to User
Microcontroller 1	Pump Control, Heating Element Control, Acquire Temperature Readings, Control Status Indicator
Microcontroller 2	Valve Control, Acquire Liquid Level Readings, Acquire Cooling Line Temperature

Table 6: Device Responsibilities

B. Timing Control Implementation

The senior design team needed a reliable and accurate real time clock to provide precise control of the recipe times input by the user. In order to achieve this goal, the senior design team took advantage of the real time clock already on the single board computer. Whenever a timed step is initiated

VII. POWER DISTRUBUTION

Within the system there is a variety of AC and DC voltages used to power different components. The table below shows the different voltages used within the system and the respective component associated with it.

Voltage	Use
115 VAC	Mains system input, resistive heating element, solenoids
24 VDC	Solenoid valves
5 VDC	Microcontroller power, analog reference voltage, sensor amplifier power
3.3 VDC	I^2C communication, raspberry pi

Table 7: List of Required Power Supply Outputs

The supplies for power at each voltage level are cascaded from the highest voltage to the lowest in series, this can be seen in figure 8. Within this distribution system, the 24VDC power supply had to be designed rather than using a voltage regulator IC due to the potential for a high current draw. The lower voltage levels utilized basic low-dropout linear regulators as the components beneath them did not draw much power.

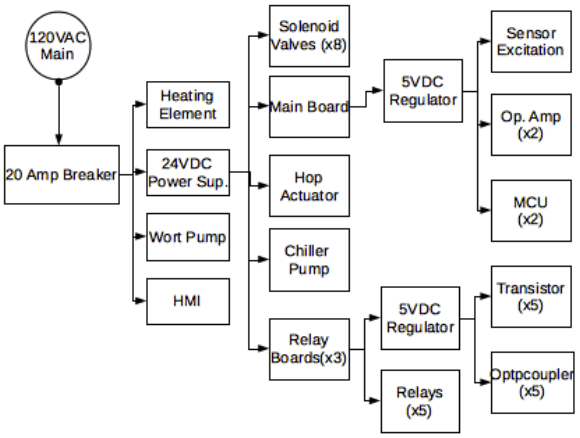


Fig. 8: Power Distribution Diagram

The 24VDC power supply design begins with a power transformer that drops mains voltage down to 24VAC. The lowered voltage power signal is then fed into a bridge rectifier to eliminate the negative portion of the cycle. A large filter capacitor then smoothes the peaks of the unregulated signal. The smoothed unregulated signal is then regulated down to the proper voltage with a zener diode and a darlington pair to amplify the current of the regulated signal to a level that is useful for powering the system components. The circuit for the 24V power supply is shown in figure 9. The transistor is attached to a heat sink to dissipate excess heat energy. The power supply utilizes three rails that check and balance each other in order to provide the most consistent power possible, as well as help distribute the heat load more evenly on the heat sink.

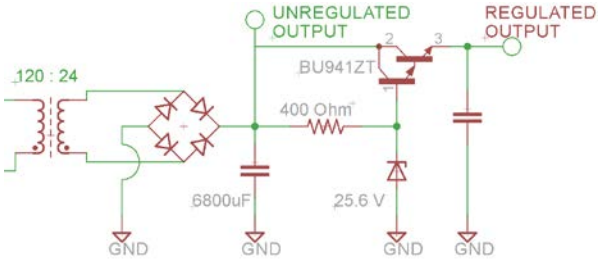


Fig. 9: 24V Power Supply Schematic

VI. HIGH POWER SWITCHING

Microcontroller I/O interface power output is typically limited to the supply voltage and around 30mA. The ATMEGA 328P-PU specifically is limited to 25mA sinking or sourcing on an I/O pin and the supply voltage used for the chips in this project is 5V. This is far from the voltage and current necessary to switch the solenoids, pumps, and heating elements on and a high power switching circuit needed to be designed. Table 8 below shows each high power component in the system along with its respective operating voltage, current consumption, and subsequent power consumption.

Component	Operating Voltage	Current Consumed	Power
Actuator	24 VDC	708 mA	17 W
Type 1 Valve	24 VDC	300 mA	7.2 W
Type 2 Valve	24 VDC	250 mA	6 W
Type 3 Valve	115 VAC	130 mA	15 W
Wort Pump	115 VAC	1.391 A	160 W
Cooling Pump	24 VDC	1.458 A	35 W
Heating Element	115 VAC	13.04 A	1500 W

Table 8: Components and Power Parameters

It was also considered that when the microcontroller is switching a device that is operating at a much higher voltage than the microcontroller, so some form of isolation would be ideal to separate the different potential levels utilized. In the figure 10 the circuit used is shown. The microcontroller output is used to bias the base of a bipolar junction transistor putting the transistor into saturation mode. Once the transistor is in saturation, the cathode of optocoupler is grounded and current is allowed to flow through the LED in the optocoupler. This then grounds the coil of a high power relay which switches on the high power components. The optocoupler provides isolation between the microcontroller and the 24VDC while the high power relay isolates the microcontroller and the 24 VDC traces from the 115 VAC mains voltage. Since relays function from magnetism in a coil, energy stored within the coil could cause problems, therfore a diode was added in parallel with the relay coils to prevent surge current into the optocoupler output side.

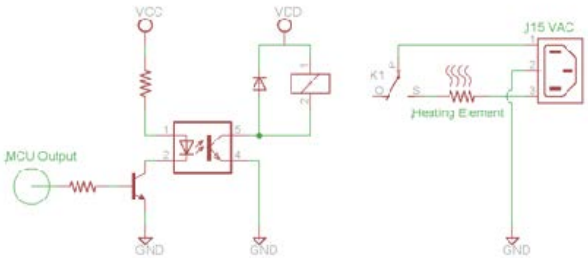
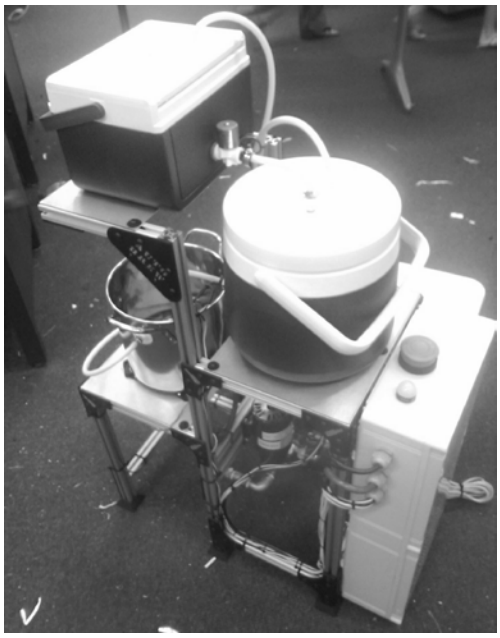


Fig. 10: High Power Switching Schematic

VIII. Conclusion

This project is mainly to demonstrate a proof of concept. The automated home brewing system, the Auto Brew as it is to be known, remains to be comprised of a portable stable aluminum frame, along with the various sensors and circuits that are needed for the internal system, a central unit for data gathering and processing, two microcontrollers to ease the workload of programming communications and control, and finally a Bluetooth server available for the user along with a very simple Android application for on-the-go monitoring.



The Auto Brew

Robert Bower will graduate with a Bachelor of Science in Electrical Engineering. He is currently working as a Gas Turbine Controls Engineer for Mitsubishi Hitachi Power Systems and will continue his position full time upon graduation.

Alonzo Ubilla, a senior at the University of Central Florida, plans to graduate from the College of Engineering and Computer Science with two bachelor's degrees, one in Electrical Engineering and the other in Mechanical Engineering. He is currently employed by Chemical Injection Technologies and plans to get hired as a full time engineer upon graduation.

Kleber Valencia will graduate with a Bachelor's of Science in Electrical Engineering from the University of Central Florida. He plans on working for the United States Patent Office. His longer term goals are going to law school to practice intellectual property.

David Rodriguez, a senior student of the computer engineering department at the University of Central Florida. He will join the workforce as a full time software engineer in Chicago upon graduation. He looks to get his Master's in Business Administration within the next two years while continuing to work full time.