Proceedings of the 6th International Conference of Control, Dynamic Systems, and Robotics (CDSR'19)
Ottawa, Canada – June, 2019
Paper No. CDSR 140
DOI: 10.11159/cdsr19.140

Home Automation with Arduino for the Internet of Things (IoT)

Aparicio Carranza[1], Syed Zaidi[3], Casimer DeCusatis[2] , Harrison Carranza[1], Danny Moonasar[1]
[1]New York City College of Technology of The City University of New York (CUNY)
186 Jay Street, Brooklyn, NY, USA
acarranza@citytech.cuny.edu; hcarranza@citytech.cuny.edu; mdk1985@gmail.com
[2]Marist College
3399 North Rd, Poughkeepsie, NY, USA
casimer.decusatis@marist.edu
[3]Bronx Community College
CUNY – 2155 University Ave. Bronx, NY, USA
syed.zaidi@bcc.cuny.edu

Abstract **-** The Internet of Things refers to the networking of electronic devices and sensors, usually monitored and controlled from the Internet. This new technology has led to a plethora of new applications such as home automation, providing the ability to access, view, and control the environment (including temperature, lighting, and more). In this paper, we test the viability of scalable, low cost home automation systems using Arduino microcontrollers. We report on the design, programming, and implementation of automated smart fans, window blinds, and tea kettles.

Keywords**:** Arduino, Home Automation, Internet of Things, Smart Appliances

1. Introduction

Home Automation has been around for quite some time and is becoming increasingly popular. According to most standard definitions, home automation or so-called "smart homes" refers to the connection of electronic devices to a central system which automates those devices based on sensors and user input [1]. According to Lindsey Turrentine, editor-in-chief of CNET.com, "Eighty-seven percent of Americans acknowledge the value of smart home technology, but only 1 in 4 has this technology in their homes [2]". Unfortunately, this technology has remained out of reach for many consumers due to its complicated setup, costly hardware, and lack of compatibility between manufacturers. Recently, with the creation of low cost microcontroller platforms (like Arduino or Raspberry Pi) as well as transducers (sensors), home automation is now becoming more widespread. In particular, Arduino microcontrollers are small, low power computers which can perform specific tasks and are more cost effective for home automation than general purpose computers. Further, by connecting such devices to the Internet using wireless connectivity, it becomes possible to monitor smart devices from anywhere in the world. The use of sensors to automate home appliances, and connect this data to the Internet, results in the so-called Internet of Things (IoT) [3, 4, 6, 7, 8].

In this paper, we discuss the design, programming, and implementation of IoT applications to control smart appliances. The paper is organized as follows: Following the introduction, we present an overview of the hardware, software, and design considerations for using an Ardiuno with Wi-Fi access as the development platform for home automation. Then we discuss three specific implementations for home automation including a smart household fan, smart window blinds, and a smart tea kettle. Schematics for the circuitry used to implement these devices will be provided, and programming commands used on the website ThingSpeak will be discussed. We illustrate the operation of these devices using sample data collected from the smart home appliances and uploaded to the ThingSpeak website.

2. Overview of Hardware, Software, and System Design

In order to connect our Arduino Uno micro-controller to the internet, we needed to obtain a Wi-Fi module. The Wi-Fi module will have the sole responsibility of transmitting data from the Arduino Uno to the Internet or more specifically the

website that we will be using to store and view our data in real time, www.ThingSpeak.com. The ESP8266-01 module was selected for its low cost and compact size (only 8 male header pins, as shown in Fig. 1).

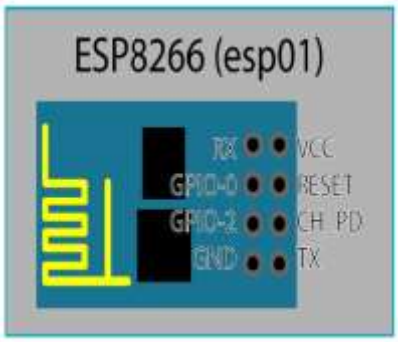

Fig. 1: The ESP8266 Board Pinout.

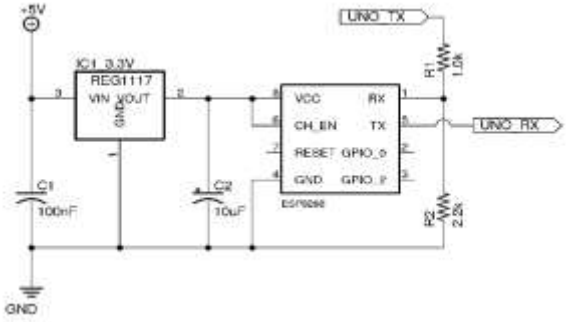

Fig. 2: Connecting an ESP8266 Module to Arduino.

2.1. Hardware and Software Design

The schematic in Fig. 2 depicts the necessary electrical circuit to connect the Wi-Fi module to an Arduino Uno microcontroller. Additional hardware was necessary since this module operates at 3.3V, which differs from the 5V supplied by the Arduino. An AMS1117-3.3 voltage regulator with the required 100 nano-farad and 10 micro-farad capacitors is used to supply a regulated 3.3V to the Wi-Fi module. This voltage regulator supports a voltage input of 5V and outputs a voltage of 3.3V at up to 1A. This is sufficient for our ESP8266 at its estimated max current draw of 300 milliamps. Note that voltages higher than 3.3 V may damage the module. Further, since the Arduino Uno's pins run at 5V, it would not be safe to connect the 3.3V tolerant RX line of the ESP8266 directly to any 5V pin of the Arduino. In order to overcome this issue safely, we utilized a voltage divider circuit on the RX line of the ESP8266 to the Arduino Uno. Since the TX line will only be transmitting, it is safe to connect it directly to the Arduino Uno as it can tolerate the 3.3V TX from the ESP8266. Normally the ESP8266 can be used as its own stand-alone micro-controller, which is why it has its own GPIO pins on it. But for our project we needed more pins for input and output sensors so using the Arduino Uno was required. Additional hardware–such as relays and motors–specific to each smart appliance will be discussed in the following sections.

Command	Description
AT	General test
AT+RST	Restart Module
AT+CWMODE=	Set mode or view supported modes. 1=STA, 2=AP, 3=both, ?=check supported modes
AT+CWLAP	List access points
AT+CWJAP="SSID","PASS"	Join access point
AT+CWSAP="SSID","PASS","CH","ENC"	Set AP parameters
AT+CWLIF	List IP address of join devices
AT+CIPSTART	set up TCP or UDP connection
AT+CIPMODE	Set data transmission mode
AT+CIPSEND	Send data
AT+CIPCLOSE	close TCP or UDP connection
AT+ CIPSERVER-"MODE","PORT"	Set as server
+IPD	received data

Fig. 3: Commands for the ESP8266WiFi Module [5].

For debugging and development purposes, the Arduino Software Serial Library was used. In this approach, serial data from one set of pins may be mirrored to another, which ultimately allows for data sent by the ESP8266 Module to be seen in the serial monitor, and data sent from the serial monitor to be sent to the ESP8266. The baud rate for the Software Serial object must correspond to that of the ESP8266. A list of common commands for the wi-fi module is shown in Fig. 3. In

order to program the ESP8266 we needed to utilize a UART (Universal Asynchronous Receiver/Transmitter) device. A UART is a computer hardware device for asynchronous serial communication. Its main purpose is to transmit and receive serial data. In order to test and program our ESP8266 we used the Arduino Uno as our UART device. The wiring schematic is exactly the same as shown above except for the pins going to the Arduino. The RX pin of the ESP8266 gets connected to pin 0 of the Arduino and the TX pin gets connected to pin 1 of the Arduino. The voltage divider circuit on the RX line of the ESP8266 is still needed. Once connected, you must upload a blank sketch or from the Arduino Examples a sketch called bare minimum. With the USB cable connecting the Arduino to the computer and making sure the Arduino is selecting the correct port, you must then open the Serial Monitor. Once the Serial Monitor is open, you need to select the correct baud rate for the ESP8266. We started testing the different baud rates on the serial monitor to see which one would respond with readable information. It is also important to note that for proper communication the option next to the baud rate of the Serial Monitor needs to be on the correct setting. For us, the option that worked flawlessly was "Both NL & CR" along with 115200 baud. For each baud rate, we tested we also reset the ESP8266 by simply unplugging the 5V line going to the regulator and plugging it back in.

We discovered that the baud rate 115200 is too fast for our Arduino. Once you connect it to Wi-Fi and it begins communicating, the information between the Arduino and ESP8266 becomes impossible to read. To address this, we needed to lower the baud rate of the ESP8266. While there are several ways to accomplish this (including flashing new firmware to the chips), eventually we located an updated AT command reference manual which describes an easier approach using the two commands AT+UART_CUR and AT+UART_DEF. It is important to note the difference between these two commands. The command that ends in CUR can be entered into the AT command serial monitor window and will test the values of different baud rates and settings. If any of the settings do not work or render the ESP8266 unresponsive, the chip can be manually reset by unplugging and re-plugging the power (it then defaults back to the original baud rate of 115200 baud). A screen shot of this result is shown in Fig. 4.

Fig. 4: Running the AT+UART_CUR command.

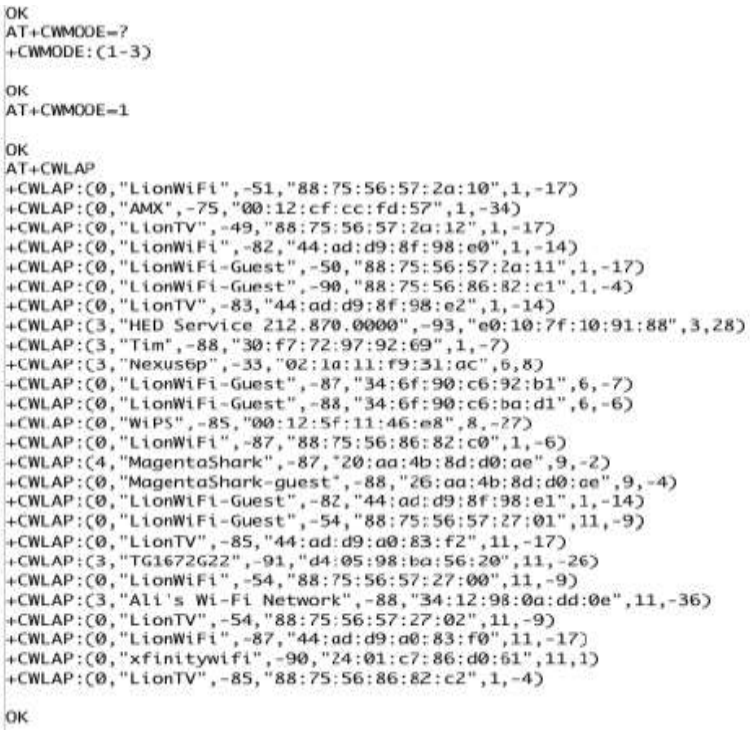

Fig. 5: Results of running the AT+UART_CUR command.

We found that the best approach was to first use the command AT+UART_CUR=9600,8,1,0,0. After this sets the proper configuration, we can use the other command to permanently store that baud rate into flash memory so that each time the device starts it will be in its new default baud rate of 9600. To do this we used the second command AT+UART_DEF=9600,8,1,0,0. Now we reset the device and set the serial monitors baud rate to 9600 and it worked

flawlessly. Other commands that we found useful include the AT command; by typing AT into the serial monitor, it will reply with an "OK". This means the ESP8266 is working fine and communicating. If you would like to check the version of the software on the chip you can enter the command AT+GMR. This will reply with the AT version of the command set along with its date, the SDK version, and the Build number and date.

2.2. Connecting to Wi-Fi

Now we are ready to connect our device to WIFI. In the command interface of the serial monitor, we can use the command "AT+CWMODE=?" to display the different modes that this chip supports. In this case this chip supports three different states:

1) Stationary Mode
2) Access Point
3) Dual mode

Since we will be connecting our device to the internet through an existing WIFI access point, we need to set it to Stationary Mode. We do this by entering the command AT+CWMODE=1. Now we enter the command AT+CWLAP to list all the Access Points in the vicinity of our ESP8266; typical results are shown in Fig. 5.

Fig. 6: Testing Wi-Fi module with suitable AT commands.

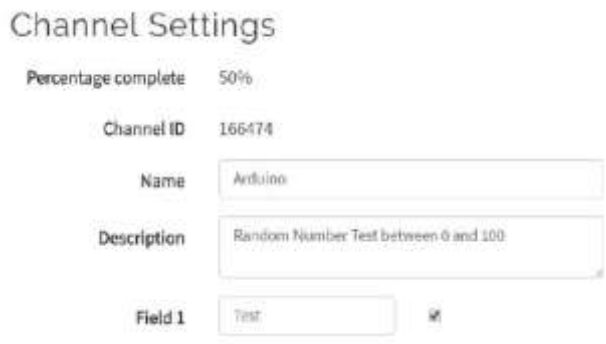

Fig. 7: Configuring a ThingSpeak Device.

To test the module, we created an access point with a Nexus 6p smartphone. To connect to our hotspot, we entered the command AT+CWJAP="Nexus6p","dantheman" where Nexus6p is our SSID and dantheman is our password. After a second or two, it will respond with the messageWIFI connected followed by WIFI GOT IP and then OK, as shown in Figs. 5 and 6. Now to verify our connection, we can type in the command AT+CIFSR to view our IP address and Mac address. Each time the ESP8266 is powered on and off it will automatically connect to the last AP point it was connected to, providing it is within range. If it is not, it will respond with the message WIFI DISCONNECTED.

2.3. The ThingSpeak Website

Now that our wi-fi settings are configured, we can move on to a simple sketch on the Arduino Uno that will test the Arduino Uno's ability to send data through the ESP8266 to the internet on the ThingSpeak website. ThingSpeak is an Internet of Things (IoT) website that allows users to upload their data form the sensors on their projects. The data is then stored and presented in a convenient fashion which allows us to further analyze and visualize the data in Matlab. ThingSpeak supports sensor data being sent from Arduinos, Raspberry Pi, and BeagleBone Black, along with other hardware as well. After signing up and creating a new account on ThingSpeak, we created a new channel labeled Arduino. This will be the channel that will be receiving the data for our project. Once inside the Arduino Channel we are presented with the option to create fields. Fields in this website would represent data from each of the sensors in our project. For instance, we created the first field which would be our test field. We also labeled the first field as Test which we will be using to test our connection with the

Arduino Uno and the ESP8266 we programmed earlier. ThingSpeak allows you to have up to 8 fields per channel. In our project, we will use 1 field for measuring the ambient light, 1 field for the temperature, and 1 field for viewing the status of the window blinds. The configuration window is shown in Fig. 7.

We can now create test code to upload data to ThingSpeak from our Arduino Uno using the ESP8266. Our ESP8266 is programmed to connect to wi-fi automatically, so now we can rewire the RX and TX pins to the previous wiring schematic. So, the TX of ESP8266 should be connected to pin 10 of the Arduino and pin 11 of the Arduino should be connected to the RX pin of the ESP8266 via the voltage divider circuit. To test our circuit and our internet connection with ThingSpeak, we will create a very simple program that will create a random number from 0 to 100 and assign it to a variable called Test. To do this we will be using the SoftwareSerial library in Arduino. The reason for using the SoftwareSerial library is because the Arduino Uno isn't capable of communicating with the serial monitor of the PC and the ESP 8266 at the same time. Instead we will use the physical serial monitor connection to view the status of our program and troubleshoot if necessary and we will create a virtual serial monitor using the software serial library to communicate with the ESP8266 via the AT command set. A sample of the recorded data for our Test variable is shown in Fig. 8.

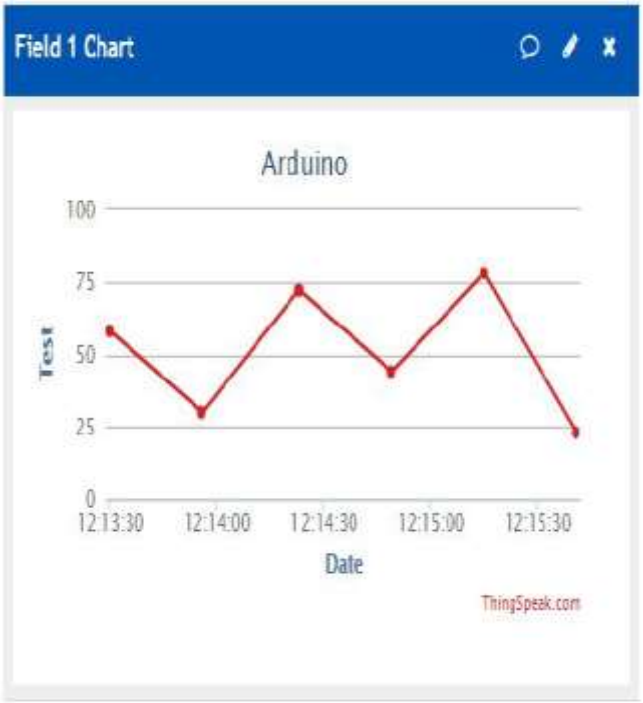

Fig. 8: Viewing recorded data on Thingspeak.

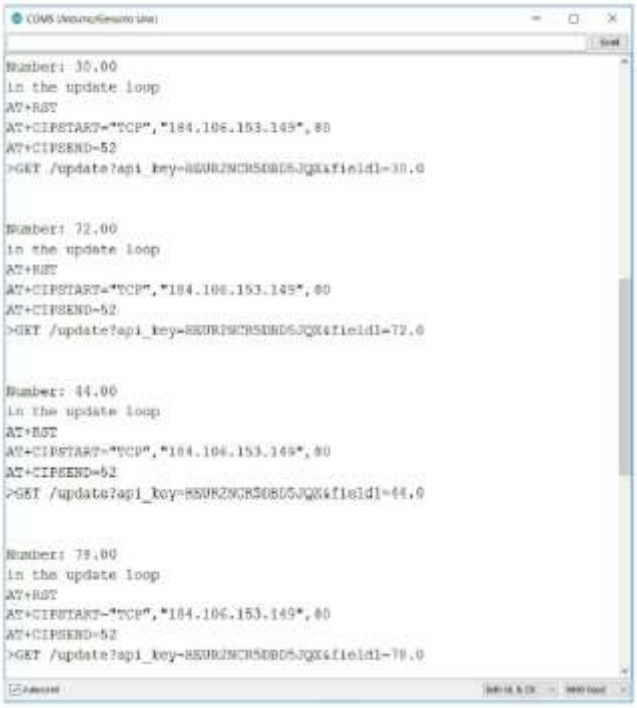

Fig. 9: Viewing recorded data on Thingspeak.

After assigning our pins for the ESP8266 and assigning our variables, we move on to the setup of the program. Here we start two serial monitor connections. The first Serial.being(9600) is our connection to the PC for monitoring while ESP8266.being(9600) is our connection to the ESP8266. The commands we send to the ESP8266 are mirrored in the alternate serial monitor as well for troubleshooting purposes. We will first start by resetting the ESP8266 with the AT+RST 10 command. Then we will delay and wait 5 seconds for it to reboot and stabilize. Then we will send an AT command which tests the ESP8266 to see if it is ready to accept commands. If the ESP8266 responds with an OK, then we know the ESP8266 and the software serial library are connected via our Arduino Uno. Now we will wait a few more seconds to insure a new IP address is obtained for the wi-fi connection. A sample screen shot is given in Fig. 9. During the main loop of our program, we have a variable called Test which is a floating point variable. We create a random number between 0 and 100 and assign it to the variable Test as a float data type. We then create a buffer space of 16 which saves a moderate amount of space to be used for the incoming data to be assigned. We then create a String which we assign to a function which uses our test variable. The dtostrf(test,4,1,buffer) is a function that converts the float variable test into a string variable that will be 4 characters

long with a precision of 1 number after the decimal point. The buffer is where these characters are written before they are saved to the variable testString. We then print the number to our Serial Monitor so we can view it as then run a new function that communicates our String value to ThingSpeak.

To make our code more efficient we decided to use an if statement to filter the results that will be sent to ThingSpeak. If the data we are sending is within a 0.2 difference of the previous number, there is no need to send that number again. This will reduce the number of uploads we make to ThingSpeak if the numbers are too close to one another. If the number is more than a 0.2 difference, the update function will be called and use a group of AT commands to send the data to ThingSpeak. ThingSpeak only allows data to be received once every 15 Seconds. For our data, we will provide enough delay time so that we are uploading data approximately once every 30 to 45 seconds. The last part of our test program is the update function itself which is responsible for sending the AT commands to the ESP8266 to upload the data to ThingSpeak.com. This update function contains 3 parameters. The first parameter is String value, which will be the testString that we created by converting the float variable test to a string. The second parameter labeled String field is the field spot which will be uploaded to the correct field location on the ThingSpeak website (in our case Field 1). Finally, the last parameter is a String value, "previous value var", which is used in keeping track of the previous variable and comparing it to the new one.

3. Application: Smart Fan

In this application, we used a temperature sensor to turn on a fan when the room becomes too warm, and turn the fan off when the room is a comfortable temperature. The temperature data is uploaded to ThingSpeak, so we can monitor it from mobile devices anywhere on the Internet as shown in Fig. 10.

Fig. 10: Smart fan temperature reporting using ThingSpeak.

4. Application: Smart Window Blinds

In this application, we used a photoresistor to detect the ambient light entering a window, a temperature sensor to detect the temperature near the window, two Hall effect sensors which work in tandem with a neodymium magnet to sense when the window blind is open or closed, and a motor which is used to open or close the blinds based on certain conditions. We can not only automate the window blind operation, but by uploading the data to ThingSpeak we can remotely access and view the status from anywhere on the Internet. Data from this project is shown in Figs. 11 – 12.

Fig. 11: Smart window blinds data reported to ThingSpeak.

Fig. 12: Smart window blinds temperature data reported to ThingSpeak.

5. Application: Smart Kettle

An electric kettle is an excellent candidate for a smart home appliance. With few–if any moving parts–controlling one with an Arduino may be achieved with the following circuit for the electronic isolation with a relay. A 15A safety fuse was also added in series with the heating element, as depicted in Fig. 13.

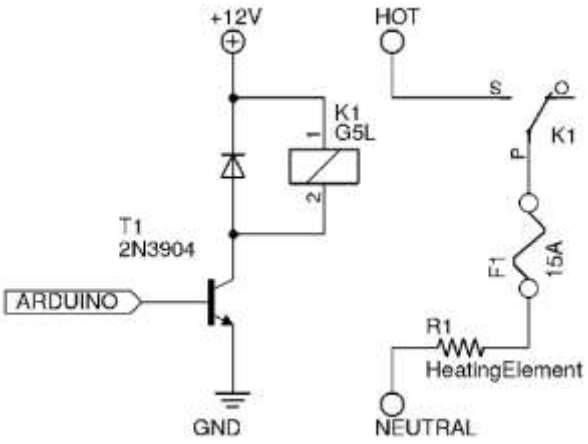

Fig. 13: Relay Isolation Circuit for Electric Kettle.

Fig. 14: Smart kettle application.

The selected device is a 1750W Electric Kettle as shown in Fig. 14. A 15A, 125V electromagnetic relay was selected for switching the power to the heating element. A 2n3904 NPN Transistor is used to switch power to the coil from a 12V source. The Arduino Uno sends either a 5V or 0V signal to the base of the transistor to ultimately control power to the heating element of the kettle. The rudimentary task of boiling water is to apply heat to the liquid until it reaches the desired temperature of at least 100 degrees C. As an electric kettle integrated with the Internet of Things, our Arduino Uno with ESP8266 WiFi Module will use both signals from the temperature sensor and commands sent over the internet to switch the heating element on or off. This process is depicted in the feedback control diagram of Fig. 15.

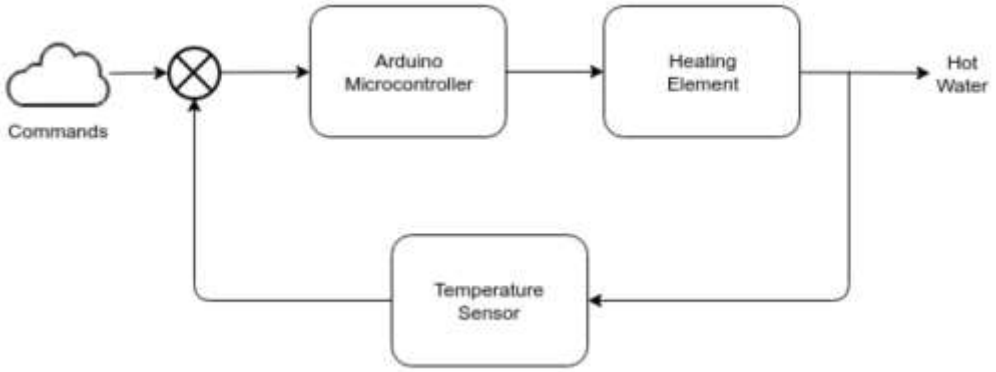

Fig. 15: Feedback Control Diagram for Smart Kettle.

6. Conclusions

Based on our successful results, we feel that a combination of an Arduino with basic Wi-Fi connectivity and the ThingSpeak website provides a viable home automation system for rudimentary Internet of Things applications. Our cost for constructing these projects was under $35, while pre-assembled home automation devices can sell for two to three times this amount. The ability to monitor the status of common household appliances over the Internet provides a new dimension for home energy management, security, and safety.

References

[1] J. Tuohy. (2015, January 26). What is home automation and how do I get started. [online]. Available: https://www.networkworld.com/article/2874914/what-is-home-automation-and-how-do-i-get-started.html

[2] D. Olick. (2016, May10). Just what is a 'smart home' anyway?. [online]. Available: http://www.cnbc.com/2016/05/09/just-what-is-a-smart-home-anyway.html

[3] Kishen. (2016, April 28). Home Automation Technology Products that can make your house a Super Power. [online]. Available: https://www.techuntold.com/best-home-automation-technology-products/

[4] Y. Khan. (2016, February 10). Smart world here we come: Internet of Things sensors are ushering in a new era. [online]. Available: https://centricdigital.com/blog/internet-of-things/smart-world-internet-of-things-sensors/

[5] ESP8266 Serial WIFI Module. [online]. Available: https://www.itead.cc/wiki/ESP8266_Serial_WIFI_Module

[6] A. Carranza, H. Kim and X. L. Chen, "Overview of Wireless Sensor Network (WSN) Security", The 55th NYSETA Fall 2018 Conference, SUNY Polytechnic Institute, Utica, NY. October 18 – 19, 2018. [online]. Available: https://sunypoly.edu/sites/default/files/nyseta/2018presentations/CarranzaA_KimH_ChenX_Fall2018.pdf

[7] Arduino. cc. (2018). Arduino – Introduction. [online]. Available: https://www.arduino.cc/en/Guide/Introduction

[8] Adafruit. (2018). Adafruit Industries, Unique and fun DIY electronics and kits. [online]. Available: https://www.adafruit.com/