ECE 202 Final Project:

The Internet Accessible Coffee Maker

An Adventure in Making Coffee From Afar

By Marcus Benzel and Kaden Strand

Colorado State University

Electrical and Computer Engineering 202:
Circuit Theory Applications

22 April 2014

Internet Accessible Coffee Maker Project Proposal

Our idea for the ECE 202 project is to control a coffee maker from a custom web server using an Arduino microcontroller and a custom Android application. We both enjoy coffee, so we thought it would be an interesting idea to make a coffee maker that you could start remotely from work or school so that your coffee would be ready by the time you returned to your house. This project will have three main components:

1. Circuitry connecting Arduino to coffee maker to turn coffee maker on and off via Arduino control.
2. Web server to control Arduino and accept remote requests to start the coffee maker.
3. Android application to send message to the server to start the coffee maker.

Optionally, we may add more functionality to the coffee maker to automatically remove the coffee filters after use and refill the coffee for a new brew.

Mid Project Report

So far, Marcus and I have made significant progress on several aspects of our project.

First, we have created a functional website hosted on CSU's Computer Science department computers to control the Arduino remotely. This site can be accessed from anywhere. We are using PHP to interpret client side input and run server side code to handle the Arduino. We are still planning to improve the visual aesthetic of the website, although the most important functionality is complete.

Second, we have used a breadboard to successfully test our circuit to turn the coffee maker on and off via Arduino. The next step for our circuit is to permanently solder the components to a PCB board. We have also tested using a servo motor to control the mechanical automation of the coffee maker. In addition, we are using electronic valves to control the flow of water to the coffee maker. If we have time, we would like to be able to build the circuitry necessary to power all components with the coffee maker's AC power cord.

Finally, we have worked to create a mechanical automation to control inserting, brewing, and removing grounds every four brew cycles. We have designed and built the mechanical parts necessary to do this, however we need to finish combining them with each other and housing them in a chassis.

ECE 202 Final Project:
The Internet Accessible Coffee Maker

An Adventure in Making Coffee From Afar

Project Origins

As freshmen students living in the dorms last year, we thought it would be useful to be able to have fresh brewed coffee ready for us by the time we returned from our classes. Our initial idea was to set up a system by which one could simply send a text message to a coffee maker and it would start making coffee, so that the coffee could be consumed on return from a remote location. Last year we did not end up making this, so this year we decided to use this idea as a basis for our ECE 202 project. We chose this project because we thought it would provide us with a variety of different types of problems in circuitry, programming, and mechanics that we could realistically solve by ourselves in the time we had to work on this project.

Planning the Project

As we began planning this project, we made a few modifications to our original idea. These changes were not a drastic departure from our original goals, yet they added additional functionality to our design.

First, we decided to make the coffee maker accessible by website instead of via text message. We primarily did this because we thought that it would be easier to program a button click on a website than to set up a system to receive texts. We also considered the fact that smart phones with web browsing capability are exceedingly common, so making a website for our coffee maker would not be a major break from our original idea of controlling a coffee maker remotely from a mobile device. Using a website also has the benefit that the coffee maker can be controlled from a regular computer, not just a cell phone. In the planning stages, we hoped to eventually create an Android application that would basically serve the same purpose as the website, however we did not have time to add the application as a project feature.

Second, we decided to add mechanics to the coffee maker to allow for multiple cycles of coffee to be brewed before coffee grounds and water had to be refilled, and old grounds emptied. We decided to create a rotating set of canisters above the coffee pot that could all be filled with grounds when the coffee maker is first set up, and each canister in turn would be used to brew coffee and then have the remaining grounds flushed out.

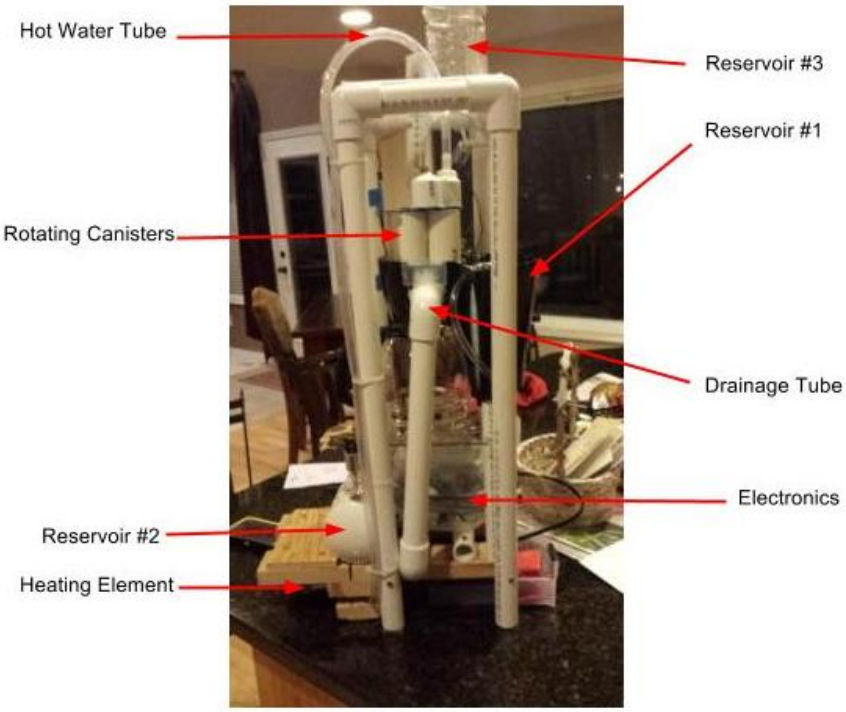
Full Design

Mechanics

Our coffee maker is fundamentally similar to a regular drip-brew type coffee maker in that after moving water through a heating element, the hot water is then poured through coffee grounds and a filter into a coffee pot.

In our design, a large reservoir holds water until coffee is ready to be brewed. An electronically controlled pump then allows enough water to flow into a secondary reservoir to brew one cup of coffee. A heating element is then turned on and water from the secondary reservoir passes through this element, boils, and rises up a tube connected up the side of the coffee maker. The heating element, coffee pot, and large water reservoir were taken from an existing coffee maker and integrated into our design. The heated water then passes through a canister full of coffee grounds, a permanent metal coffee filter, and into the coffee pot.

Once the coffee is brewed, a revolving canister system empties the used grounds and prepares the next canister of grounds to be brewed. We used short cuttings of PVC pipe for our canisters, and sandwiched them between two CDs with holes drilled where the pipes were placed so that water could flow through any of the canisters. We placed this canister component on top of another CD. This second CD has only two holes, one that is covered by a permanent metal filter and another that is used to drain used grounds. On top of the canister component we placed a final CD, also with two holes. One of the holes is connected to the hot water tube, while the other is connected to a tertiary reservoir (a connected water bottle) on top of the coffee maker. This tertiary reservoir is also controlled by an electronic pump; when it is opened, water flushes used grounds out of the canister. By holding the top and bottom CDs in place while rotating the center canister component with a servo motor, we can brew multiple cups of coffee in a row by using a fresh canister for each brew. The used grounds drainage hole in the bottom CD is connected to a pipe that feeds into a small plastic drawer, which collects drain water and grounds to be easily removed.



Labeled Components

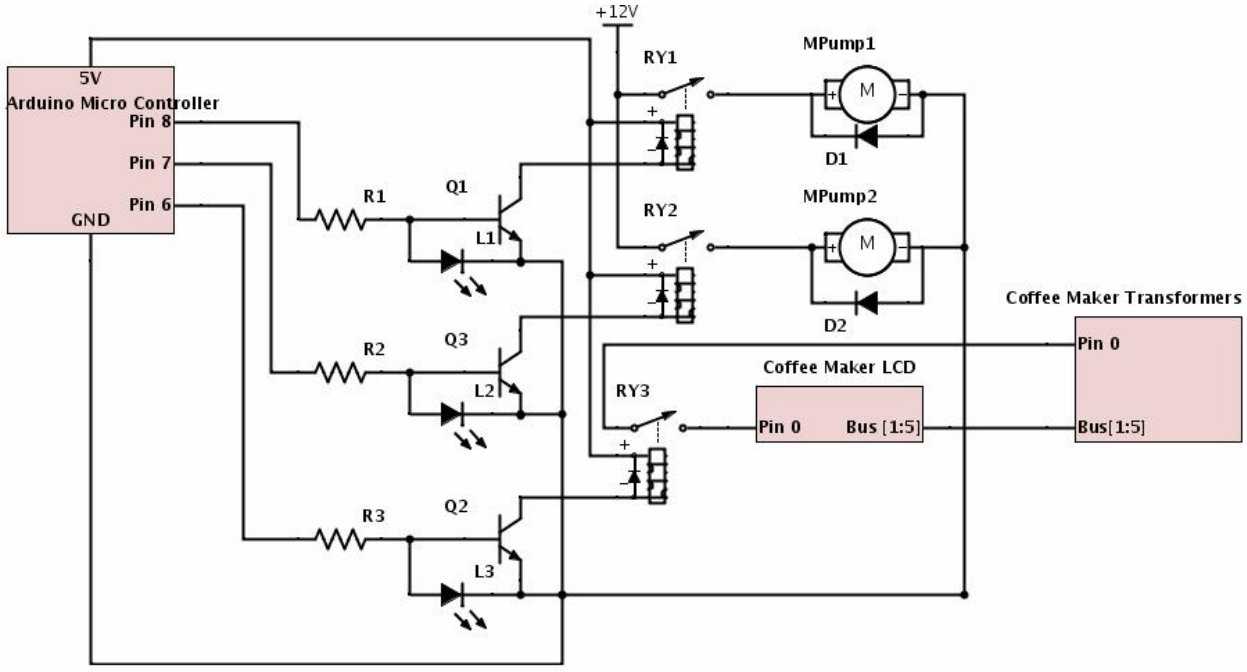
Rotating Canisters

Top View

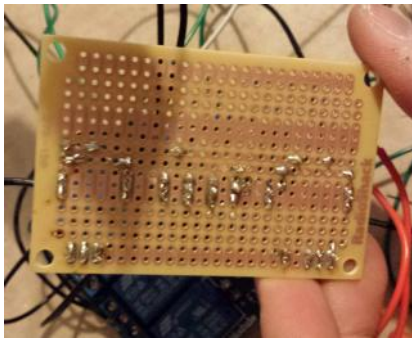
Electronics

This project required even more electrical components than we initially planned for. The coffee maker we bought initially turned the heating element on and off by using a series of transformers on one circuit board connected to a second circuit board with a small display. We determined which of the transformers turned on the heating element, and connected this transformer to be controlled by a power relay. The two electric pumps that allow water to fill the secondary reservoir and flush the grains from the canisters are also controlled by power relays. The electric pumps require diodes connected in parallel with each pump in order to reduce the back-EMF they produce. In addition, the canisters are rotated by a servo motor, which is voltage controlled. We also added LEDs to the power relay paths so that each time a relay was switched, a corresponding LED would turn on.

All of our circuitry is controlled by the output pins of an Arduino microcontroller. We soldered output wires from the Arduino onto our own PCB board. On this board we soldered all of our connections to the power relay circuit, coffee maker circuit boards, LEDs, and diodes. Below is the schematic for our circuitry:



Circuit Schematic



Circuit Soldered to PCB

All Circuitry

Enclosed Circuitry

Formula

This project did not require the use of many formulas. However, in order to calculate the correct resistor value to use for the LEDs, we used Ohm's Law. By the LED specification, the current we desired was 35 mA. The Arduino supplies a DC voltage of 5V, and the LED drops 1.5 V across its terminals. Using Ohm's law:

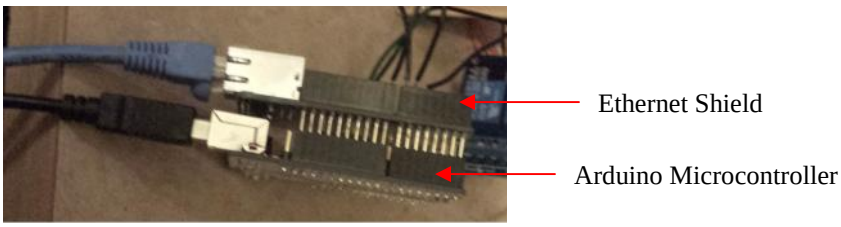
$$V = I \cdot R \Rightarrow R = V/I$$
$$R = (5 - 1.5)/.035$$
$$R = 100\Omega$$

Code

We programmed an Arduino microcontroller to control the electronic components. The Arduino code simply writes logical highs and lows to the output pins, delaying at each step of the brewing process to allow the water to cycle through the machine. The code snippet below shows how an output pin is enabled as an output and is then written to with a logical high signal. This corresponds to an output signal of 5V.

```
pinMode(Relay1, OUTPUT);
digitalWrite(Relay1, HIGH);
```

We also needed to connect to the internet in order to receive a signal to start the coffee maker. We used an Ethernet "shield" made by the same company that created Arduino to do this. The shield is another circuit board that plugs directly into the Arduino, and an Ethernet cord can be connected to it.



The shield can be set up to have a unique IP address that can accept variables passed as URL parameters. In our case, the IP address is 129.82.226.111. By entering http://129.82.226.111/$1 into a web browser, "1" is sent to the Ethernet shield, and this value can be interpreted by our Arduino code to turn the coffee maker on. Because the Ethernet shield has only a local network IP address, we decided to use the public personal website available to us at http://www.cs.colostate.edu/~kstrand/ as a public interface that could send signals the Ethernet shield set up on CSU's local network. A JavaScript button on the webpage runs a PHP script when it is clicked. The PHP script executes a Bash script on one of the CS department computers that opens a text version of http://129.82.226.111/$1 in the computer's terminal. Because this department computer is on the CSU network, the Ethernet shield receives "1" and starts the coffee making process. The full code for the project is attached in Appendix A.

Testing and Demonstration

Due to the complexity of integrating the mechanical, electrical, and programmed components of this project together, we conducted small tests before doing any building, and then continued testing each component as we connected components and slowly put together the coffee maker. We first tested our programs without connecting the Arduino to any of the any electronics, and we tested our circuits frequently before making solder joints.

We had the most difficulty trying to make our servo motor turn the canisters. The servo we are using connects poorly to the rotating beam, making it difficult to turn the canisters. The rotating canister system is also not perfectly level. For this reason we are demonstrating a single coffee brew cycle, and then we will demonstrate the turning of the canisters separately. We will not be brewing multiple cups of coffee in a row.

Conclusion

Our project is successful in most aspects of the design, with only a few exceptions. The coffee maker has the ability to receive a signal from a smart phone and brew coffee. Water is pumped through the system by electrically controlled pumps, and LEDs display the current state of the coffee making process. Our rotating canister system does not work as effectively as we had hoped, due to the problems we encountered with the servo motor. The canister system still turns, but it is not reliable for making multiple cups of coffee. Due to budget constraints, we could not spend a lot of money making sure that the servo was perfectly aligned so the canisters could rotate smoothly. Also due to budget considerations, our materials were not food safe.

We learned a great deal about how to plan a project from start to finish, as well as how to use information from multiple disciplines to create a final product. If we were to do this project again, we would most likely use a different servo motor and improve our design for rotating the canisters. With more time, we would also create an Android application that could control the coffee maker.

Acknowledgements

Thank you to Mr. and Mrs. Benzel, for allowing us to use their tools, and to the ECE and CS department professors who taught us how to create basic circuitry and programs.

Appendix A: Code Base

PHP Script

```
<?php
	$output = exec('/s/bach/j/under/kstrand/public_html/testScript', $out,
	$return);
	echo $return;
	echo "<pre>$output</pre>
?>
```

Bash Script

```
#! /bin/bash
echo "Test Script";
xdg-open 'http://129.82.226.111/$1'
```

Demo 1

```
/*
  Web Server Demo
  thrown together by Randy Sarafan

 Allows you to turn on and off an LED by entering different urls.

 To turn it on:
 http://your-IP-address/$1

 To turn it off:
 http://your-IP-address/$2

 Circuit:
 * Ethernet shield attached to pins 10, 11, 12, 13
 * Connect an LED to pin D2 and put it in series with a 220 ohm resistor to ground

 Based almost entirely upon Web Server by Tom Igoe and David Mellis

 Modified by Marcus Benzel and Kaden Strand

 */

#include <SPI.h>
#include <Ethernet.h>
#include <Servo.h>

Servo dial;
int score;
const int Relay1=6;
const int Relay2=7;
const int Relay3=8;
const int servo=5;
const int x=240000; //delay for the bottom valve to fill tank
const int w=240000; //delay for coffee to brew 4 minutes
const int y=240000; //delay for it to clear out the grounds
const int z=150; //delay to fill the tube with grounds
const int begining =35; //begining position for servo
const int final=127; //servo final for first rotation
int current =0; //servo position
```

```
boolean incoming = 0;

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xD5, 0xF1 };
IPAddress ip( 129,82,225,171 ); //<<< IP ADDRESS

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);

void setup()
{
  //setting up the pin outputs
  pinMode(Relay1, OUTPUT);
  pinMode(Relay2, OUTPUT);
  pinMode(Relay3, OUTPUT);
  dial.attach(servo); //attach the servo pin
  //setting the outputs for starting voltages
  digitalWrite(Relay3, LOW);
  digitalWrite(Relay2, LOW);
  digitalWrite(Relay1, HIGH);
  dial.write(begining); //get the servo to the start position
  current=begining;

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.begin(9600);
}

void loop()
{
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so you can send a reply

        //reads URL string from $ to first blank space
        if(incoming && c == ' '){
          incoming = 0;
        }
        if(c == '$'){
          incoming = 1;
        }

        //Checks for the URL string $1 or $2
        if(incoming == 1){
          Serial.println(c);

          if(c == '1'){
            digitalWrite(Relay3, HIGH); //open the bottom valve
            delay(x); //wait for the tank to fill;
            digitalWrite(Relay3, LOW); //close the valve
            digitalWrite(Relay1, LOW); //turn on the coffeemaker
```

```
            delay(w); //wait for all coffee to brew;
            digitalWrite(Relay1,HIGH); //turn the coffeemaker off
            /*turn the servo to the gournds dump
            for(int i=current;i<final;i++){
              dial.write(i);
              delay(30);
            }
            current=final;
            */
            digitalWrite(Relay2, HIGH); //open the top valve
            delay(y); //delay to clear the grounds out
            digitalWrite(Relay2, LOW); //close the top valve
          }
          if(c == '2'){
            Serial.println("fill");
            dial.write(groundsBegin);//turns servop to 1st fill spot
            delay(z);//waits for use to fill with gounds
            //rotates to next spot
            for(int i=groundsBegin;i<groundsNext;i++){
              dial.write(i);
              delay(30);
            }
            current=groundsNext;//sets the current position of the servo
          }

        }

        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(1);
    // close the connection:
    client.stop();
  }
}
```

Demo 2

```
#include <SPI.h>
#include <Ethernet.h>
#include <Servo.h>

Servo dial;
int score;
const int Relay1=6;
const int Relay2=7;
const int Relay3=8;
const int servo=5;
const int x=240000; //delay for the bottom valve to fill tank
const int w=240000; //delay for coffee to brew 4 minutes
const int y=240000; //delay for it to clear out the grounds
const int z=150; //delay to fill the tube with grounds
const int begining =35; //begining position for servo
const int final=127; //servo final for first rotation
int current =0; //servo position


boolean incoming = 0;
```

```
// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xD5, 0xF1 };
IPAddress ip( 129,82,225,171 ); //<<< IP ADDRESS

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);

void setup()
{
  //setting up the pin outputs
  pinMode(Relay1, OUTPUT);
  pinMode(Relay2, OUTPUT);
  pinMode(Relay3, OUTPUT);
  dial.attach(servo); //attach the servo pin
  //setting the outputs for starting voltages
  digitalWrite(Relay3, LOW);
  digitalWrite(Relay2, LOW);
  digitalWrite(Relay1, HIGH);
  dial.write(begining); //get the servo to the start position
  current=begining;

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.begin(9600);
}

void loop()
{
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so you can send a reply

        //reads URL string from $ to first blank space
        if(incoming && c == ' '){
          incoming = 0;
        }
        if(c == '$'){
          incoming = 1;
        }

        //Checks for the URL string $1 or $2
        if(incoming == 1){
          Serial.println(c);

          if(c == '1'){
            digitalWrite(Relay3, HIGH); //open the bottom valve
            delay(x); //wait for the tank to fill;
            digitalWrite(Relay3, LOW); //close the valve
            digitalWrite(Relay1, LOW); //turn on the coffeemaker
            delay(w); //wait for all coffee to brew;
            digitalWrite(Relay1,HIGH); //turn the coffeemaker off
            /*turn the servo to the gournds dump
            for(int i=current;i<final;i++){
              dial.write(i);
              delay(30);
            }
            current=final;
            */
            digitalWrite(Relay2, HIGH); //open the top valve
            delay(y); //delay to clear the grounds out
```

```
            digitalWrite(Relay2, LOW); //close the top valve
          }
          if(c == '2'){
            Serial.println("fill");
            dial.write(groundsBegin);//turns servop to 1st fill spot
            delay(z);//waits for use to fill with gounds
            //rotates to next spot
            for(int i=groundsBegin;i<groundsNext;i++){
              dial.write(i);
              delay(30);
            }
            current=groundsNext;//sets the current position of the servo
          }

        }

        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(1);
    // close the connection:
    client.stop();
  }
}
```

Demo 3

```
#include <SPI.h>
#include <Ethernet.h>

boolean incoming = 0;

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xD5, 0xF1 };
IPAddress ip( 129,82,226,111 ); //<<< IP address

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);

void setup()
{
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  digitalWrite(6, HIGH);

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.begin(9600);
}

void loop()
{
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
```

```
      if (client.available()) {
        char c = client.read();
        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so you can send a reply

        //reads URL string from $ to first blank space
        if(incoming && c == ' '){
          incoming = 0;
        }
        if(c == '$'){
          incoming = 1;
        }

        //Checks for the URL string $1 or $2
        if(incoming == 1){
          Serial.println(c);
          //check for 4
          if(c == '4'){
            Serial.println("on");
            while(true){
            //turn LED 7 on and off
            digitalWrite(7,HIGH);
            delay(1000);
            digitalWrite(7,LOW);
            //turn LED 6 on and off
            digitalWrite(6,LOW);
            delay(1000);
            digitalWrite(6, HIGH);
            //turn LED 8 on and off
            digitalWrite(8,HIGH);
            delay(1000);
            digitalWrite(8,LOW);
            }
          }
          //check for 2
          if(c == '2'){
            //turn LED 8 on and off
            digitalWrite(8,HIGH);
            delay(1000);
            digitalWrite(8,LOW);

          }
          //check for 3
          if(c == '3'){
            //turn LED 6 on and off
            digitalWrite(6,LOW);
            delay(1000);
            digitalWrite(6,HIGH);

          }


        }

        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(1);
    // close the connection:
    client.stop();
  }
}
```