Team 11 Documentation

Proximity-Activated Brewer

Christopher Robles | Justin Oh | Nancy Vuong

Table of Contents

Introduction

This project is a proximity-based coffee brewer. Coffee is a beverage consumed daily by millions of people, so why not integrate the brewing process seamlessly into your daily routine? This is perfect for anyone that wants coffee to start brewing once they arrive at work or home - by checking the location of the connected Arduino to your phone's location data, it can automatically start brewing when you are nearby and deactivate when you leave.

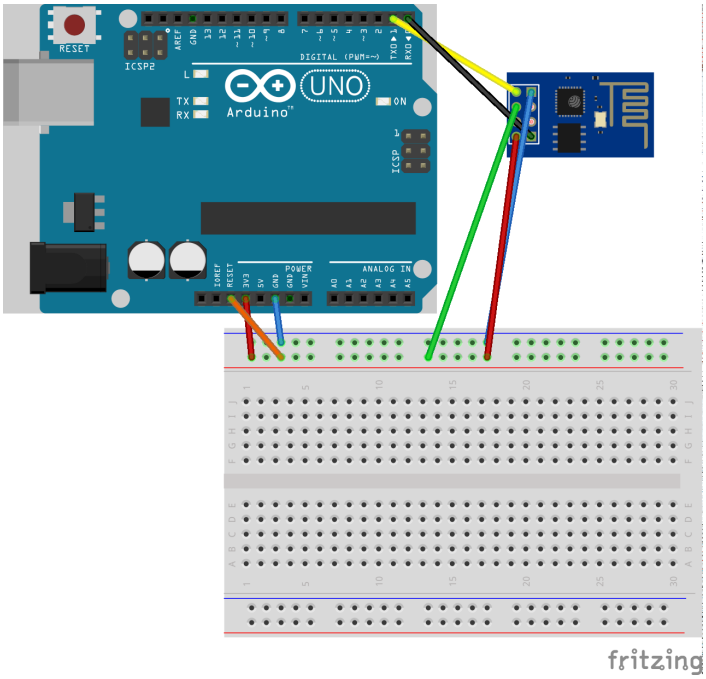
Materials

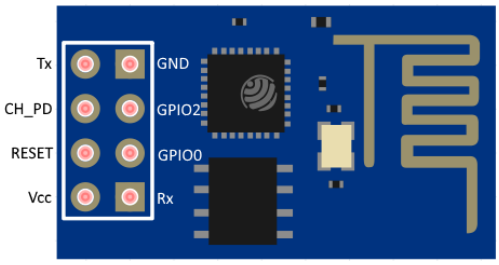
- Arduino Uno (1)
- ESP8266-01 Wifi Module (2)
    - AT-command support
- Relay Module (1)
    - 5v
    - 1 channel
- Coffee-Maker (1)
- USB 2.0 A-B Cable (1)
- 9v Battery (1)
- 9v Battery Snap (1)
- Jumper Cables
    - Male-Male (2)
    - Female-Male (8)

Instructions

Prepare ESP

In order to get the project to work, the ESP has to be made compatible with the Software Serial that we are working with in our code. This requires some tinkering with the baud rate of the ESP. To configure the baud rate, assemble the circuit below.





Connections

ESP		Ard
GND	->	GND
TX	->	TX
RX	->	RX
CH_PD	->	3V3
VCC	->	3V3

Also connect Arduino RST to GND

Ensure that the red LED on the ESP turns on. It indicates that the ESP is being powered. Open the Arduino IDE's Serial Monitor. Set line endings to "Both NL and CR". After typing in AT and pressing Enter, an "OK" should pop up - try setting the baud rate to 9600 or 115200 if either doesn't work. If nothing pops up on either, you may need to try other baud rates or update the ESP firmware.

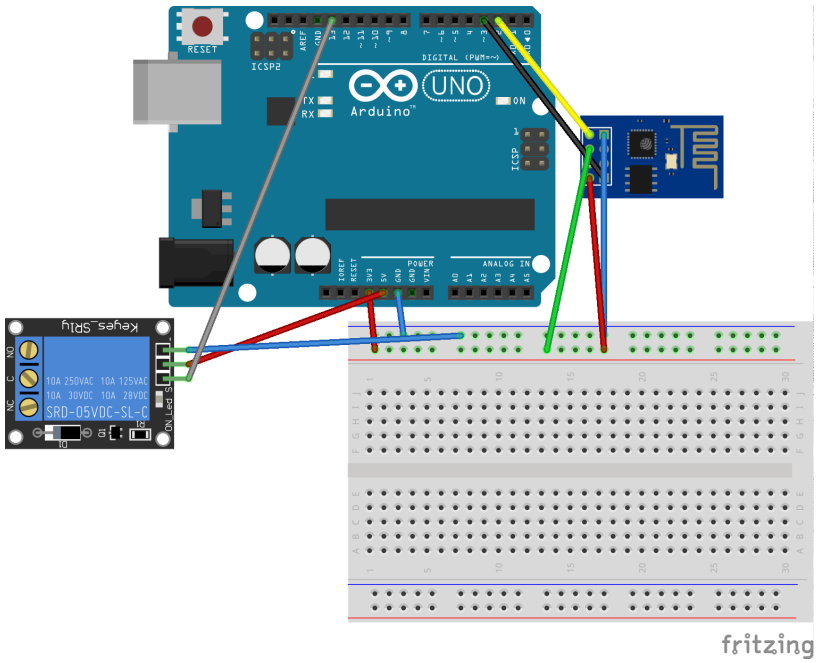
Once you are able to get the "OK" message back, you'll have to reset the baud rate to 9600 if it's not already 9600. The command used to do so can vary. Ours was able to be reset using "AT+CIOBAUD=9600".

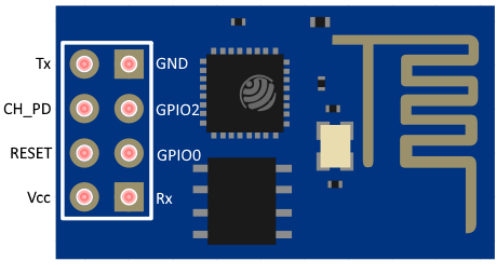
Build Circuit

Now that the ESP is ready to work with, the circuit should be completed so that we'll eventually be able to test our code.

First, the connection between the relay and the coffee maker should be figured out. Always make sure that the coffee maker is unplugged while working with its internal circuitry. Open up the coffee maker however you have to so that you can get to the wires that lead to and are from the external plug. Find the wire that supplies voltage to the coffee machine. This will most likely be red and coming directly from the external plug. You'll want to make a cut to that wire between its connection to the external plug and its connection to anything else; the 'anything else' will likely be the switch that allows the coffee-drinker to turn the machine on and off.

Now there are two ends of that wire where the cut was made. Take the end that is connected to the external plug and strip enough of the insulation to get it into the relay. Connect this end to the middle pin. You'll need a screwdriver to secure this wire in place. Assuming that you want the coffee maker to activate when a signal (high voltage) is sent to the relay, connect the other end of the just-cut wire to the NO (Normally Opened) pin. The rest of the circuit should be completed with jumper cables according to the diagram below.





Connections

ESP		Ard
GND	->	GND
TX	->	2
RX	->	3
CH_PD	->	3V3
VCC	->	3V3

RELAY		ARD
+	->	5V
-	->	GND
S	->	13

Code

```
/*************************************************************
 ESP8266    ARDUINO
 ————————————————
 RX         ->      3
 GND        ->    GND
 VCC        ->   3.3V
 TX         ->      2
 CH_PD      ->   3.3V

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

/* Comment this out to disable prints and save space */
#define BLYNK_PRINT Serial

#include <ESP8266_Lib.h>
#include <BlynkSimpleShieldEsp8266.h>
#include <math.h>
#define earthRadiusKm 6371.0

// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = “BLYNK_AUTH_TOKEN”;

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = “WIFI_SSID";
char pass[] = “WIFI_PASSWORD";

// Hardware Serial on Mega, Leonardo, Micro...
// #define EspSerial Serial1

// or Software Serial on Uno, Nano...
#include <SoftwareSerial.h>
SoftwareSerial EspSerial(2, 3); // RX, TX

// Your ESP8266 baud rate:
#define ESP8266_BAUD 9600

ESP8266 wifi(&EspSerial);

double currentLat = 43.612255;         // Latitude of coffee maker
double currentLong = -110.705429;     // Longitude of coffee maker
double phoneLat = currentLat;
double phoneLong = currentLong;
const double maxKmRange = 0.03;      // Range in km you want coffee maker activated
int coffeePin = 13;                          // Pin that sends signal to relay
```

```
// Converts decimal degrees to radians
double deg2rad(double deg) {
  return (deg * M_PI / 180);
}

// Converts radians to decimal degrees
double rad2deg(double rad) {
  return (rad * 180 / M_PI);
}

// Returns distance in km between two GPS coordinates
double distanceEarth(double lat1d, double lon1d, double lat2d, double lon2d) {
  double lat1r, lon1r, lat2r, lon2r, u, v;
  lat1r = deg2rad(lat1d);
  lon1r = deg2rad(lon1d);
  lat2r = deg2rad(lat2d);
  lon2r = deg2rad(lon2d);
  u = sin((lat2r - lat1r)/2);
  v = sin((lon2r - lon1r)/2);
  return 2.0 * earthRadiusKm * asin(sqrt(u * u + cos(lat1r) * cos(lat2r) * v * v));
}

// Executes every time the Blynk server receives an update from the app
BLYNK_WRITE(V13) {
  phoneLat = param[0].asFloat();
  phoneLong = param[1].asFloat();
  pinMode(coffeePin, OUTPUT);
  double dist = distanceEarth(currentLat, currentLong, phoneLat, phoneLong);
  if (dist <= maxKmRange) {
    digitalWrite(coffeePin, HIGH);
  } else {
    digitalWrite(coffeePin, LOW);
  }
  Serial.println(dist); // For debugging purposes
}

void setup()
{
  // Debug console
  Serial.begin(9600);
  delay(10);
  // Set ESP8266 baud rate
  EspSerial.begin(ESP8266_BAUD);
  delay(10);
  Blynk.begin(auth, wifi, ssid, pass);
}

void loop()
{
  Blynk.run();
}
```

Testing Tips

Once the code and circuit are ready, the project is ready to test.

If you don't have access to Wi-Fi that allows devices on the network, you could try using a phone hotspot instead. Use phones with good service. Make sure that the GPS stream widget is chosen in the blink app and that the virtual pin that it writes to matches the one that you call from the code in the BLYNK_WRITE method. Connect the Arduino to your computer and review the messages in the serial terminal to help debug. The ESP does not have a connection to the Blynk server until a ready/ping message is written to the terminal. To quickly test the connection, use a button widget and make it output to the relay signal pin; for our program this pin is 13. Pressing this button should toggle the light on the relay.

Conclusion

If time allowed, we would've wanted to create an app independent from Blynk that would be able to set custom brewing times and the location of the Arduino, along with gyroscope support to detect whether phone is in pocket, on table, etc. so we could coordinate early morning phone movements to coffee maker activation. Considering our adversities with the ESP, we'd want to use a stronger Wi-Fi chip and an external power source for the ESP for more stability.

Our greatest trouble was definitely the ESP - communicating to it and telling it to connect to the Internet gave us many struggles with the various errors it would give us. The moment it connected to Blynk through Wi-Fi was a great relief.

References

Distance in km between latitude and longitude:
https://stackoverflow.com/questions/10198985/calculating-the-distance-between-2-latitudes-and-longitudes-that-are-saved-in-a

Blynk sketches and documentation:
https://examples.blynk.cc
https://www.blynk.cc/getting-started/
http://docs.blynk.cc/

Wiring Arduino to ESP:
https://www.esp8266basic.com/flashing-instructions.html

Instructables:
http://www.instructables.com/id/Control-Your-Home-Appliances-Using-Arduino-and-Rel/

And so many more Google searches…