Intelligent Rain Window Protector 2.0

December 12th 2016

James Wyeth, Edward Ellis, Shixin Wu
ECE 110/120 Honors Lab
Fall 2016

Photo: Miracle Forest

Introduction

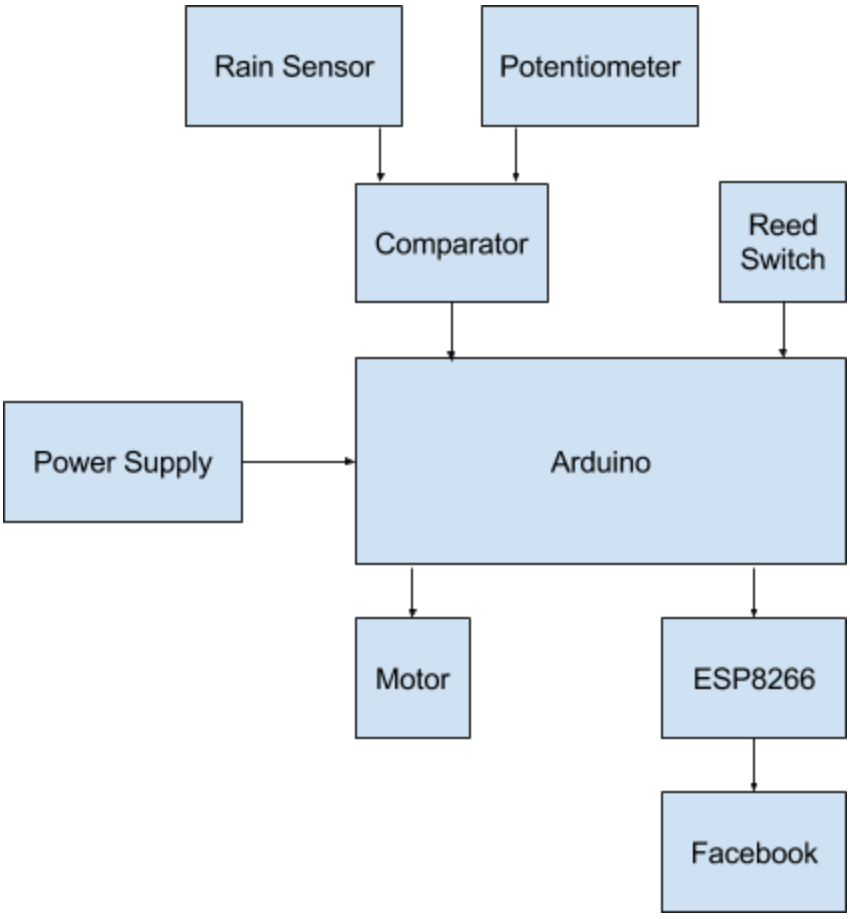
Due to the unpredictable nature of Illinois's weather, many have returned to their homes to discover expensive electronics and belongings ruined by rain incoming from a nearby window that was never closed. Sometimes the window was left wide open because of a temporary lapse in memory, and sometimes the storm opened the window on it's own. Either way, our project aims to solve this problem. The Intelligent Window Rain Protector 2.0 uses low cost electronic devices and the technological skills of college students to keep anything that may be damaged by rain away from harm. Our system uses an analog rain sensor to close your window automatically. This system can be easily adapted to fit any window, since it uses a standard motor to close the window.

As is commonly known, exposing electronics to water is bad. It creates shorts in the circuitry and can burn out the internals, rendering the entire device inoperable. We believe that by creating a system with a rain sensor to close your window, it is possible to keep this pesky water outside of your house, where it belongs. It's even possible to re-open the window once the rain stops, keeping air flowing through your house.

To make sure our system can compete in the information age, we added the ability to monitor your system remotely. It's always nice to know what's happening in your house, and this is becoming an expectation for all products. Our system is no different, and can, at any time, be asked about the status of the window and rain. As a bonus, the window can even be remotely controlled, all from the facebook messenger app.

Design

In this block diagram, the rain sensor and potentiometer provide analog signals to the comparator. The comparator and reed switch supply digital signals to the Arduino. The arduino creates a PWM signal for the motor driver and communicates with the ESP over UART. The power supply provides power to the circuit.



As the surface of the rain sensor begins to collect rain, the resistance decreases, changing the voltage supplied to a comparator circuit. If the voltage surpasses a threshold, determined by comparing the voltage across the rain sensor with the voltage of a potentiometer, the comparator will send a digital signal to an arduino. Once this signal is received, the arduino will send a PWM signal to a brushed DC motor driver to run a motor, closing the window. If the reed switch detects that the window is closed, the motor will not be allowed to run. Each action will trigger a message to be sent to the ESP8266, which can be queried at any time as to the status of the window (Figure 1, Figure 2, Figure 3). This system is supplied power with a 9VDC wall wart.

Results

We found that the resistance of the rain sensor plate starts at around 1MOhm dry, and drops to around 10kOhm when wet. We decided to use a 100kOhm resistor for the other end of the voltage divider, so the rain detector voltage went from 4.5V dry to 0.5V wet, giving us a lot of

tolerance for determining when it's raining. We set the reference voltage on the potentiometer to around 3V, and it worked fine.

The motor was a 12V motor, but since we didn't need all the power it could provide and we wanted to be able to use a battery, we decided to power the whole system off 9V. At this voltage, the motor still had plenty of power to move the window at 40% duty cycle. The reed switch closed when it was about 4cm away from the magnet. When the rain sensor detected water, the motor turned the window until it was closed, showing that our project works.

To allow the user to control the window from a mobile device, it was first decided that to benefit the user experience and simplify the codebase, a Facebook Messenger bot would be used. To send and receive messages, HTTPS requests are used to communicate between the client and the backend database. Although many services offer online storage databases, Google's Firebase platform was used due to its superior scalability and it's excellent Firebase library used for the ESP8266, which significantly reduced the time taken to implement communication between the ESP8266 module (Figure 4) and Firebase (Figure 3). However, Firebase had some limitations including poor ability to implement scripting, which meant that a third party service had to be integrated to monitor the change in state in the Firebase database and immediately send a HTTPS POST Request to Facebook. Since short development time was prioritized, a simple, drag and drop solution was used. Losant (Figure 1) is a comprehensive IOT platform, providing excellent management tools and easy to follow diagrams for those of us not trained in JavaScript as a second language. After implementing features to enable bidirectional communication between the ESP8266 platform and Facebook Messenger, some simple commands were added, including those capable of checking status and changing the state of the window (Figure 2). In the future, tools will be added to integrate multiple windows into a network, allowing for both users to open more than one window at a time, as well as allow for machine learning networks to monitor large sets of data to optimize times the window should be open or closed.

Problems and Challenges

When we started this project, we thought we would use a TI MSP430 Launchpad as our control system, but we decided to switch to an arduino because it was more convenient. We also had some trouble finding a window that would work for our purposes, so we asked the ECE

machine shop to make one for us. The machine shop was nice enough to provide us with a motor that would work for the project, and embedded it into the window to eliminate any mechanical work we had to do.

Future Plans

This project is hopefully the first in a series of smart home devices that will make people's lives easier. The next step with this project would be to replace the rain sensor with one based on the internal reflection of IR light within a window. This would make the system more reliable and wouldn't require any pieces outside.

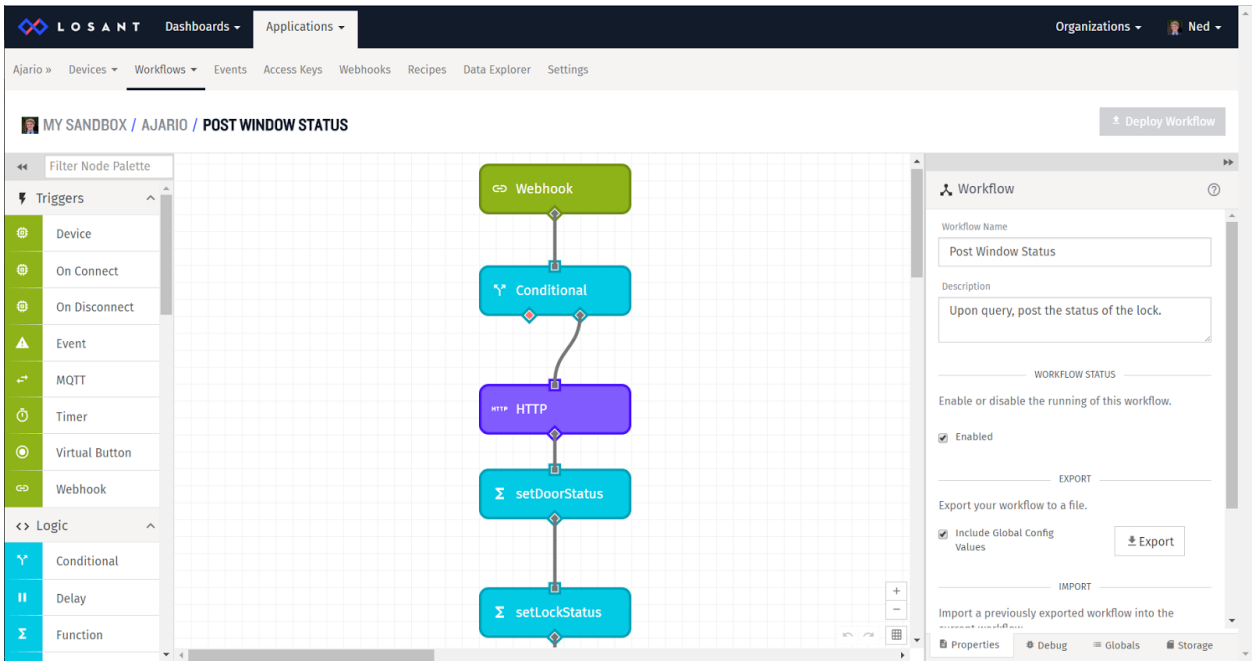


Figure 1. Losant IOT Interface used to process HTTPS POST and GET requests

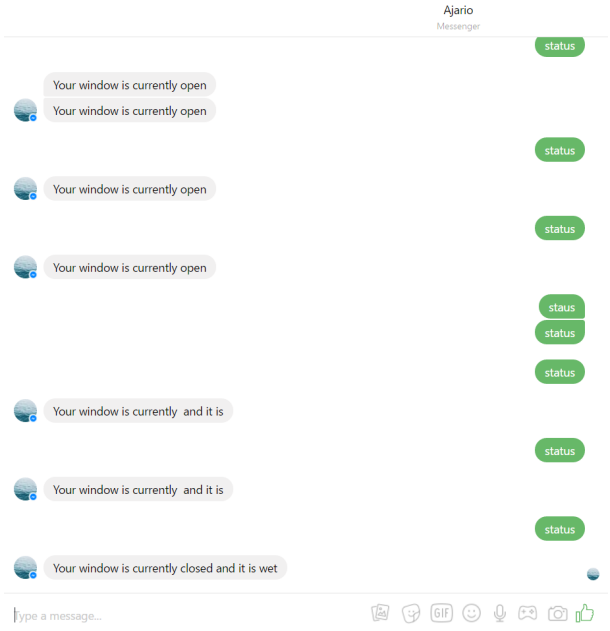


Figure 2. Facebook Messenger client used to interface with the window wirelessly

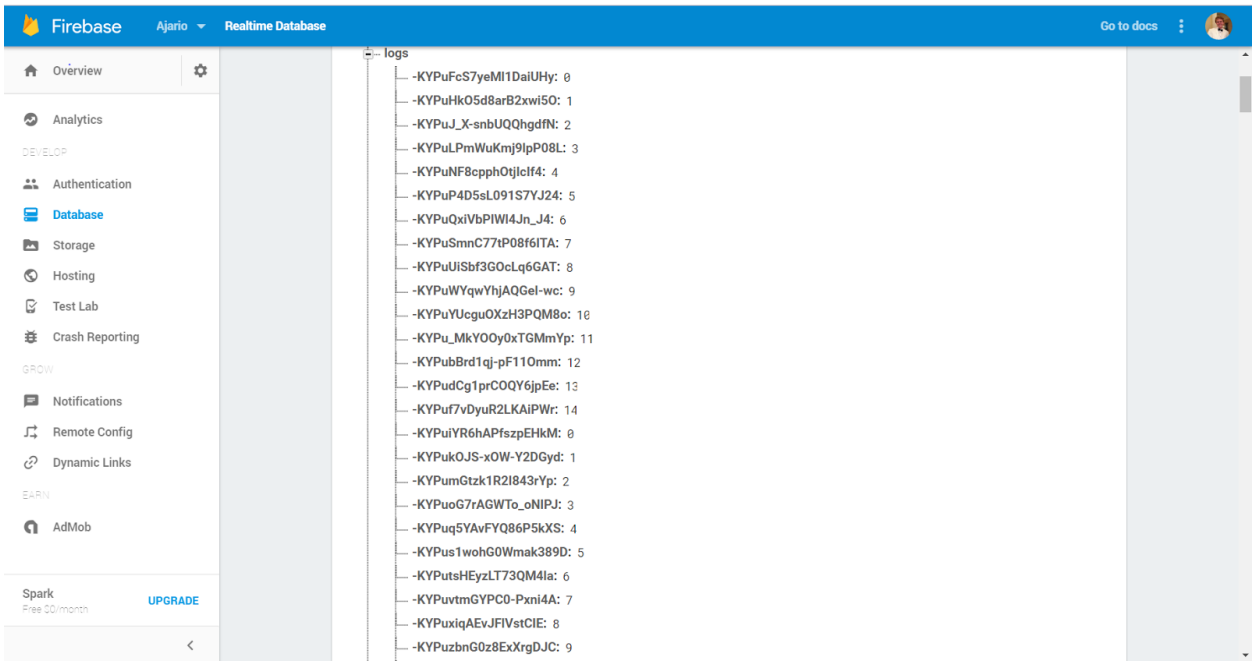


Figure 3. Google Firebase backend used to store window statuses on cloud.

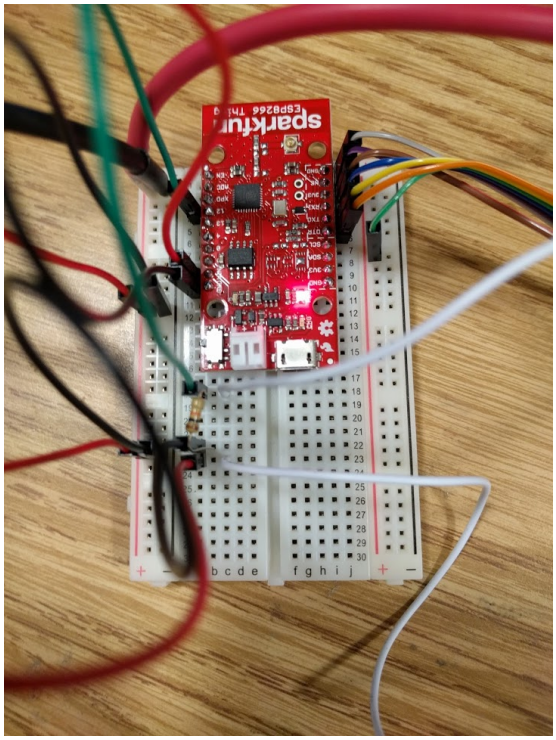


Figure 4. ESP 8266 SoC used to interface with WiFi LAN

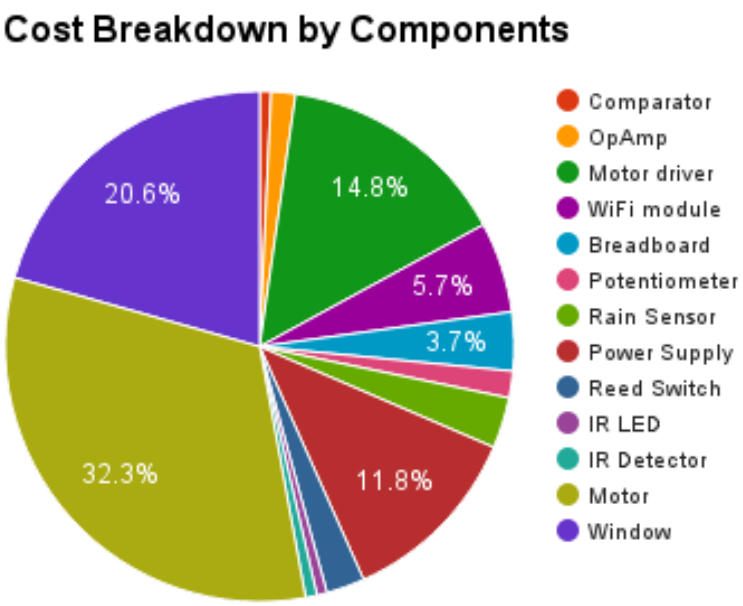


Figure 5. Relative cost per part

Arduino Code

```
int swIn = 8;

int rain = 4;

int mSped = 3;

int mDir = 2;

int openDir = 1;


void setup() {

  pinMode(swIn, INPUT);
```

```
  pinMode(rain, INPUT);
  pinMode(mSped, OUTPUT);
  pinMode(mDir, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  int raining = digitalRead(rain);
  int closed = digitalRead(swIn);

  digitalWrite(mDir, !openDir);

  if(!closed && raining)
  {
    analogWrite(mSped, 100);
  }
  else
  {
    analogWrite(mSped, 0);
  }

  uint8_t packet = (raining << 1) | closed;
```

```
  Serial.print(packet);


  delay(10);
}
```

ESP8266 Code

```
#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>
#include <Servo.h>

// Login data for WiFi and Firebase
#define FIREBASE_HOST "ajario-2b883.firebaseio.com"
#define FIREBASE_AUTH "uJyP8aE33JAbONiSSW6SaEZhvPB6wLLFsWqLLqIb"
#define WIFI_SSID "IllinoisNet_Guest"
#define WIFI_PASSWORD ""

// Input/Output Pin Locations
const int LED_PIN = 5; // Onboard LED, indicates WiFi Status
const int REED_SENSOR = 12; // Digial Input, Measures Reed Sensor
const int SERVO = 13; // Digital Output, controls servomotor to open lock

void setup() {
```

```
// initialize all in/out connections
pinMode(LED_PIN, OUTPUT);
pinMode(REED_SENSOR, INPUT_PULLUP);
pinMode(SERVO, OUTPUT);

// Initialize LED status
byte ledStatus = LOW;
digitalWrite(LED_PIN, ledStatus);

// Initialize Servo Position
int pos = 0;

// connect to wifi.
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("connecting");
while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
  delay(500);

  /* Blink the LED */
  digitalWrite(LED_PIN, ledStatus);
  ledStatus = (ledStatus == HIGH) ? LOW : HIGH;
}
```

```
  ledStatus = HIGH;
  digitalWrite(LED_PIN, ledStatus);
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
}

int n = 0;

void loop() {
  // set value
  Firebase.setFloat("number", 42.0);
  // handle error
  if (Firebase.failed()) {
      Serial.print("setting /number failed:");
      Serial.println(Firebase.error());
      return;
  }
  delay(1000);

  // update value
  Firebase.setFloat("number", 43.0);
  // handle error
  if (Firebase.failed()) {
```

```
    Serial.print("setting /number failed:");
    Serial.println(Firebase.error());
    return;
}
delay(1000);

// get value
Serial.print("number: ");
Serial.println(Firebase.getFloat("number"));
delay(1000);

// remove value
Firebase.remove("number");
delay(1000);

// set string value
Firebase.setString("message", "hello world");
// handle error
if (Firebase.failed()) {
    Serial.print("setting /message failed:");
    Serial.println(Firebase.error());
    return;
}
```

```
delay(1000);


// set bool value
Firebase.setBool("truth", false);
// handle error
if (Firebase.failed()) {
    Serial.print("setting /truth failed:");
    Serial.println(Firebase.error());
    return;
}
delay(1000);


// append a new value to /logs
String name = Firebase.pushInt("logs", n++);
// handle error
if (Firebase.failed()) {
    Serial.print("pushing /logs failed:");
    Serial.println(Firebase.error());
    return;
}
Serial.print("pushed: /logs/");
Serial.println(name);
delay(1000);
```

```
}
```