Solar Fan

Richard Andre Mercado, Devin DuBois, Braulio Quintana

Table of Contents

Synopsis

The Solar Fan is an IOT device that uses two Raspberry Pi's, a web-based IDE, and a dashboard combo to cool a user on a hot day based on the amount of light in the area. One Raspberry Pi is connected to a photoresistor and the other Raspberry Pi is connected to a 9V fan which provides great cooling for the user. The two raspberryPi's are connected via a node js server running on a third PC. The PC is also the central hub that hosts the dashboard and HTML pages. It also does all of the backend calculations between the sender and receiver side.

Sensor Pi

The sensor Pi is connected to the photoresistor which is connected to a capacitor. The photoresistor controls the amount of energy that gets to the capacitor. When the photoresistor detects more light it allows more electricity through charging the capacitor more quickly. The opposite is true when the photoresistor detects less light. The time it takes for the photoresistor to charge is calculated using a GPIO pin. Which waits for the current to transition from low to high (Once the capacitor is charged it lets the current through). The C++ code on the Raspberry Pi then calculates the time it has been since the current was high. After calculating the time it sends the value using cpprest and curl. It then makes an HTTP request to the server where it sends the sensor value obtained and then prints out whether it was received successfully or not.

Server

The server acts as a central hub, receiving data from the Sensor Pi and sending control signals to the Actuator Pi. It manages the backend calculations and communication between the sender and receiver sides. It also hosts the processes for the front-end dashboard and the IDE, which are later described in the document. It also acts as the compiler for the commands which are inputted into the app editor.

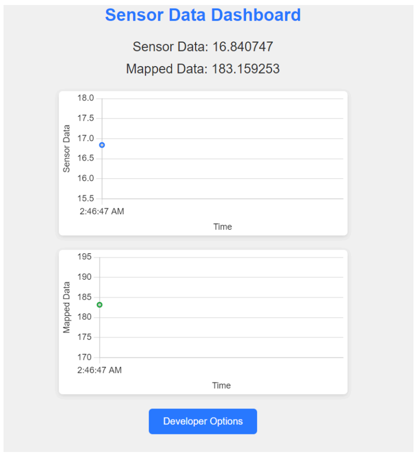
Actuator Pi

The actuator Pi is connected via an HTTP client request from the server. The breadboard circuitry comprises a fan attached to a DC motor powered via an external power source (battery) with an integrated circuit (L293D). Using the IC, GPio pins serve as output to the IC's input pins with the help of wiringPi and softPwm. In the C++ code, the curl is initialized with a URL to fetch callbacks from, performing a GET request / 3 seconds and parsing JSON data into double[]. Following that, the duty cycle % of the DC motor is changed per each reading of the request and the speed is accordingly based on the threshold.

Front End

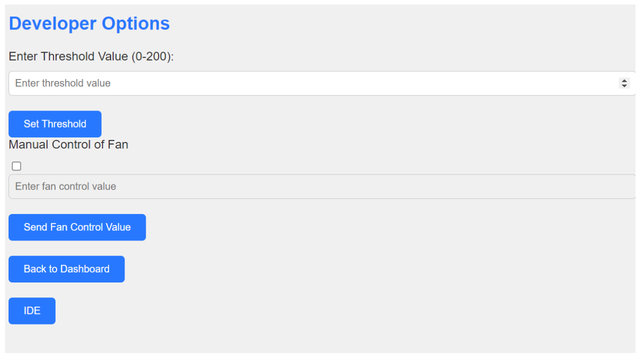
Our dashboard features two graphs that determine the change in the data that the pi is communicating over time. It shows the sensor data and the inversely correlated mapped data and has a timestamp for each one. This allows the user to have a visual representation of how the power of the fan and the light exposed changes over time.

The developer page allows manual input of values for testing. It allows the user or developer to change the threshold that the fan will activate. It also allows manual control of the fan.


Sensor Data Dashboard

Sensor Data: 16.840747

Mapped Data: 183.159253


Developer Options

Enter Threshold Value (0-200):

Manual Control of Fan

IDE

The IDE includes tabs for things, services, relationships, and apps. The Things tab shows a list of devices that are connected to the server and shows the status of whether they are online or offline in real-time. It describes each thing as a sender or receiver. It also includes a button back to the dashboard to see the light sensor data.

The services tab describes the services that are provided by each thing. The first Pi is a sender which sends light data to the server.

The apps tab is where the majority of the IDE functionality is located. The apps tab allows users to upload existing apps from their personal computer to the IDE and edit from there. While they are editing they can test their code in real time using the terminal provided to the user underneath the app editor. In addition, once a user is done working they can save their apps back to their PC to work on later. A user can also upload a working directory and see the files in it listed. The app editor implements its language with various get and post commands all separated by when a user enters a new line. The commands include “get sensorData”, “get fanstatus”, “get mappedValue”, “get sensorstatus”, “post threshold *integer*” and “post manualfan *integer*”. Unfortunately, the debugger is still under development but most users will find no issue in programming their solar fan how they want it.

Open-source packages were essential in the creation and deployment of the application throughout the project’s development. To start with setup, we installed mainstream open-source packages to help get our Atlas smart thing set up: such as wiringPi, curl, cpprest, thread, etc. To ensure communication between the pi and the server, we installed express to ensure reliable transfer-layer protocol communication and JSON to ensure parsing of input is of the data type we desired to receive and send. There are also other libraries we installed as well, those being bodyparser, Websocket, multer, socket, and ping. We also resorted to open-source forums such as Stackoverflow to guide how to implement desired functionality and features.