Second Prototype Test Report

Team: 31
Members: Michelle Thevenin, Jiawei Liao, Keven DeOliveira, Samarah Uriarte, Michael Harkess
Test Date: 3/1/22

Equipment Summary

For the hardware aspect of our second prototype testing, we used an ESP32-WROOM-32 microcontroller, a computer that acted as the system's power source and showed server data, a 12 Volts and 1.5 Amperes Stepper Motor with a L298N Motor Driver, a 12V Battery Pack, a L7805 5V Voltage Regulator, and a DHT22 temperature and humidity sensor .

For the software side, a React Native program acted as the front end, and temporary back end, of the mobile application. Additionally, React JS was used with Node.js to design the website. For the server, we use Amazon EC2 running Linux (Debian) and Apache 2.4.51. In the database, Amazon RDS hosts the MySQL database, and the microcontroller was programmed in the Arduino IDE.

In order to demonstrate the mobile app's functionality, Expo Go was integrated into the testing process. Using this app, a team member connected to the EcoSwitch mobile application through a smartphone and interacted with the simple interface that was available for testing purposes. The setup starts with a circuit that is already connected with the sensor, the microcontroller, the motor driver, the stepper motor and the powered battery pack. When the input temperature from the mobile app does not match the collected temperature value, the motor will slightly turn. All of the equipment used during the second prototype testing was consistent with the expectations outlined in the test plan.

Setup Summary

Before testing, we set up the server, database, mobile app, admin website and the ESP32 program. For the server side, we initially made sure both the server and database instances on AWS were not running so that it would start collecting values once testing had begun. Then, we initiated the server and database instances in AWS and made some test queries to ensure activation. For the mobile app side, we opened Expo Go on a team member's smartphone and scanned the QR code for the EcoSwitch mobile app to access it. In order to initialize the temperature and humidity values with the most recent entry from the server database, we pressed the "Update" button that was displayed through the mobile app's UI. To update the database with the user's desired temperature we typed a value into the provided text box and submitted the response by pressing the "Set" button displayed in the app's UI.

For the admin website, we started a Node.js instance on a team member's computer and navigated to both the Main and StudentAdd pages. For the main page, to initialize the temp and humidity values for every unique entry in the database, we pressed the "Get Data" button displayed by the website's UI. For the StudentAdd page, we put some test values into the displayed text boxes and pressed the "Submit" button that was displayed to update the database. Finally, for the ESP32 side, we made certain that the device was correctly connected to a teammate's computer. Then, we used the Arduino IDE to run the appropriate program in the corresponding directory. We also made sure that the motor connected to the ESP32 had sufficient batteries to power the component as well as making sure the battery housing was providing power to the motor.

Testing Summary

We first set up the testing environment, with a few exceptions that were required to be built as we described the testing procedure and how our solution worked on a high level. The description of our second prototype included the circuit and its new additions, which consist of the step motor, the corresponding driver, the voltage regulator, and the battery pack that powered it. The other components that were present in the first prototype were the DHT sensor and the ESP32. We then explained that the DHT sensor would read in temperature and humidity information, send it to the AWS server instance we have constantly running, and store it in the database that is accessible to the mobile app and website via API calls.

Following this high-level overview of the system, we demonstrated the mobile app interface and functionalities of retrieving data from the EcoSwitch device and sending back a user-inputted temperature to the database, which then stored this value.

The website was also shown, as well as the various pages that were currently available with this version. All available EcoSwitch devices and their corresponding information were queried and displayed in a clean chart once the appropriate button was pressed. This demonstrated the website's ability to connect to the server with the aforementioned API interface.

Unfortunately, we were facing WiFi issues that prevented the ESP32 script from connecting to BU WiFi, which impeded the program from controlling the actuator based on the user input that was retrieved from the mobile app. At the end of the testing session, the ESP32 was able to connect, allowing us to demonstrate this functionality as well, however, with a few undesirable patterns. Similarly, the website was unable to update the student user database by registering a student to a specific test device as a result of API connection difficulties. With previous testing sessions prior to the official prototype testing, the script that was responsible for this mechanism successfully established a connection with the API endpoint that represented the student user database and allowed users to add students to it without issues. While the testing process did not go completely smoothly, we were provided with a great deal of advice and have a good sense of what our next steps will be.

Detailed Measurements

Most of the measurements made for the prototype testing were ensuring that aspects of the project were working as anticipated when tested real-time. The first measurement made was confirming that the information gathered by the sensor was accurately delivered to the mobile device. We verified that the values present on the phone were the same as those logged in the ESP32 console, as well as the values stored in the database. This was similar to the first prototype testing.

Following this, we began demonstrating the new functionalities of the project. We entered a desired temperature through the mobile application and sent it, then confirmed that the database received that information correctly. We also ensured that the database successfully relayed the data to the ESP32. Once the ESP32 received the information, we were able to demonstrate that it was accurate using the console, and the motor spun. However, at the time of testing, the motor functionality was not working as fully desired; there were some bugs in the programming that were fixed afterwards.

The administrator website was able to display dummy data that was manually entered into the database, as well as the information from the ESP32 sensor used in testing. However, due to a bug in the API calls, we were unable to demonstrate and measure the success of adding a new device/student via the website.

Conclusion

Upon completion of the testing, it was determined that the holding torque of the stepper motor is ultimately not enough to turn the switch. Additionally, the motor that connects vertically to the attachment needs to be supported. To solve these testing issues, a stronger stepper motor will be bought and mounted onto the attachment. An additional piece that can be screwed onto the FCU will be implemented to apply a downward force on the motor to hold it down. This box will also serve as an insulator for the circuit so that the temperature sensor is not affected by the active FCU. There are two screw holes on the side of the FCU switch that will be used for the purpose of securing the insulation box. These two design changes will provide a direct force to eventually turn the switch under the attachment.

On the software end of the project, the app and website currently have working functionalities that need to be updated further. The mobile application needs Google authentication, UI updates, and to be tested extensively. The website also needs Google authentication along with POST request functionalities and an updated UI. Once the hardware mechanisms have been established, our final step would be to determine the logic in the device software that we will use to automatically turn the switch to reach the user desired temperature.