Software Formalization

Year: 2024 Semester: Spring Team: #2 Project: MOUSE

Creation Date: ­02/12/2024 Last Modified: 02/16/2024

Author: Braden Kirkendall Email: bkirkend@purdue.edu

1.0 Utilization of Third Party Software

We are using third party software for our webserver hosting and then the basic C library for the ESP32S3 and interfacing provided by Espressif’s IDF for our microcontroller. On each of these platforms, we are developing our own code to be deployed within these third-party environments. This includes creating our own code for the microcontroller, backend, and frontend functions. An additional note is that while not a software application, we are using an AWS EC2 instance to host our webserver online.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | License | Description | Use |
| Nginx | MIT | “NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. NGINX is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption.” [1] | We will use this server to host our backend from our EC2 instance. This will allow the node.js code to be accessed remotely. |
| Node.js | MIT | “As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.” [2] | We will use this runtime to build our backend application for our webserver. |
| npm | Artistic License 2.0 | “npm is the world's largest software registry. Open source developers from every continent use npm to share and borrow packages, and many organizations use npm to manage private development as well.” [3] | We will use this registry to run commands from our package.json files which build and serve our frontend code. |
| Espressif IDF | Apache License 2.0 | “The ESP-IDF, Espressif IoT Development Framework, provides toolchain, API, components and workflows to develop applications for ESP32 using Windows, Linux and macOS operating systems.” [4] | We will use this framework and its libraries to develop code for the ESP32 as well as build and flash the images we create on it. |
| Vue.js | MIT | “Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative and component-based programming model that helps you efficiently develop user interfaces, be they simple or complex.” [5] | We will use this framework to create our frontend components and modify them dynamically based on data requests. |

2.0 Description of Software Components

The software components for the project are identified under 4 categories: Frontend Display, Microcontroller I/O, Sensor Data Processing, Path Recording and Playback. All of these are developed by the team for this project and thus are not ported or implemented from other projects. Each section is also supplemented by the software outline diagrams in Appendix 1.

The Frontend Display component is contained within our Vue.js framework that is running through the open-sourced npm run package. It is responsible for communicating via a socket to the Node.js backend to get status updates to trigger HTML and CSS for the user to interact with. The types of information that the frontend must fetch are the microcontroller connection status, which state of operation the user is in, and if a movement alert has been triggered. Some example states of operation are recording, playback, alert, or free roam which all dictate how the user can interact with the server to pass data back to the microcontroller. As part of this process, the state can be updated in the backend through buttons that send state messages to either drive the vehicle or change the mode of operation. There is also a layer of protection as the frontend is responsible for fetching response messages that indicate whether user actions are successful. For instance, you cannot simultaneously enter the recording state and the playback state, thus an error is sent back to the user alerting them that this action cannot be performed until one of these states is terminated.

The Microcontroller I/O operates on our ESP32S3 board and dictates the integration of the motors through a TB6612FNG motor driver, a shift register displaying diagnostic data, and external IR sensors. The software on microcontroller is additionally responsible for connecting to the webserver using a socket connection with its built-in Wi-Fi module. This communication channel is responsible for sending IR sensor data for movement detection and microcontroller connection status information while also being ready to receive drive commands generated by the webserver from the user. Monitoring these drive conditions involves modifying the frequency characteristics of PWM signals on GPIO outputs to the motor controller. To shift out data to the diagnostic LEDs, the microcontroller takes in battery health status from a coulomb counter integrated circuit and projects that into a linear display. The microcontroller must be able to accept and handle all these data streams simultaneously to operate effectively. A final feature of the microcontroller that we want to implement is the stretch goal of using a single servo that rotates to perform readings with an IR sensor in all directions when monitoring. This implementation requires the microcontroller to execute scanning logic to drive this motor and stop in specific orientations to take in sensor readings.

The Sensor Data Processing component consists of receiving data on the node.js backend using a socket endpoint with messages from the microcontroller. Based on the received collection of sensor data, an algorithm should be performed to analyze the data with the goal of determining whether a movement alert should be detected. When movement is then detected, this information should be logged, stored, and presented to the user. The corresponding actions for these alerts involve web socket signals to the frontend as well as data stored to log files for historical retrieval..

The Path Recording and Playback feature is designed as a state machine operating on node.js backend. The state is always in one of three modes: normal, recording, and playback. In normal mode, basic movement functionality is passed through from the frontend arrow keys, through the backend, to the microcontroller for motor execution. This information is not tracked in normal mode. When recording mode is initiated through a frontend socket request, the backend changes states. Now, when passing through directional data, the server will simultaneously write information to a recording file. This includes the direction of the data as well as the timing with which it was received relative to other commands. Until a signal is received to stop recording or a maximum size is exceeded, this process will continue. The result is a recording file that could be read later. This leads into the final state: playing back the recorded data. In playback mode, the directional buttons on the frontend will not be able to control MOUSE execution. This functionality is forfeited to the previously recorded data file. By reading each directional entry and employing a corresponding time-delay, the microcontroller is passed the same execution instructions as before and will traverse the same path. Since MOUSE is a security robot designed to serve long shifts, the playback is looping and will circularly execute the recorded path until it is told to stop. This allows a user to record a closed path which is then executed indefinitely, serving as a fixed patrol route.

3.0 Testing Plan

Each component provided in this section will be tested in the priority level associated with its section header. For instance, 3.1 is the most important and would be higher precedence than 3.2. Within each section, the tests for each functionality component are then ranked sequentially in order of importance or logical progression. For instance, a test might be ranked accordingly because it incorporates an addition to the functionality shown in a previous section.

3.1 Microcontroller I/O: This is the most critical component since the actions of motor movement and scanning for movement detection are critical features of our minimum viable product.

1. Test IR sensors by printing direct output detection data on connected terminal output to ensure satisfactory test cases for instances in which there is and is not movement in the surroundings.

2. Test PWM configuration and modification by driving a single GPIO output and performing waveform-validation on the signal frequency using an oscilloscope. Then, connect a motor and verify visually that the motion is smooth and modifiable with the signal modulation parameters.

3. Be able to show a message received from the microcontroller on the webserver and then show on the microcontroller that a signal has been received from the webserver. Next, do the same using a command to trigger various actions such as toggling the motors.

4. For our stretch PSDR, connect servo motor to a GPIO output and verify that it is able to be rotated in 90-degree increments, stopping at each direction to simulate taking measurements.

3.2 Sensor Data Processing: This section also falls under critical functionality as the purpose of our design is to be able to process and alert users of security warnings. Failure to process these warnings from the microcontroller and translate them into messages to the user would nullify our design goals.

1. Test that the microcontroller can send a signal with data from the IR sensor with data suggesting either successful or unsuccessful motion detecting.

2. Be able to show a popup on the frontend received via a signal from the node.js server after a signal indicating movement from the microcontroller has triggered an alert.

3. Be able to run the MOUSE over the duration of multiple alert detections and then retrieve a log of all detection events that occurred during that session and when they occurred.

3.3 Frontend Display: While the frontend display offers a lot of features simply for product and user convenience, it is essential for employing state changes and generating movement commands.

1. Be able to demonstrate precise controlling of motor operations by using the frontend arrow keys to navigate a simple course created by tape. The MOUSE should be able to stay within a 2-foot path and perform tight turns.

2. Responsiveness will be tested by using a stopwatch to find the difference in time between the forward button being pressed and the MOUSE beginning to move forward. This is key to making the controller navigation feel precise and gives a sense of the latency between communications.

3. Try unplugging and plugging in the microcontroller to change the connection status shown on the main dashboard and verify that it is updating accurately and in a timely manner.

3.4 Recording and Playback: This set of components seeks to achieve the stretch goal of being able to set a path manually, record it, and allow it to be reproduced repeatedly to perform long, overnight security shifts. However, they are not essential to the core functionality of the product.

1. On just the backend, be able to record and then playback a series of directional arrow commands passed in by the user. The commands should be varied in timing and value and can be compared to the code-generated response parroting the same actions. Ideally, we will have a delay that is only a couple milliseconds and nearly indiscernible to the user.
2. This is more of a capstone test involving using the functionality of part 1 while also connecting the microcontroller, having the path recorded physically as the mouse moves via the directional arrows. The test will have the MOUSE follow along a path created on the ground, marked with a visual indicator such as tape. Then, after the path is complete, the MOUSE should be placed back at its starting position and start executing its playback. The goal is for the MOUSE to always be within 1 meter of the desired path ensuring precise recording, playback, and motor-driving functionality in tandem with each other.

4.0 Sources Cited:

[1] “Welcome to NGINX Wiki! | NGINX,” *www.nginx.com*. <https://www.nginx.com/resources/wiki/#:~:text=NGINX%20is%20a%20free%2C%20open>

[2] Node.js, “About,” *Node.js*. <https://nodejs.org/en/about>

[3] “About npm | npm Docs,” *docs.npmjs.com*. <https://docs.npmjs.com/about-npm>

[4] “About - ESP32 - — ESP-IDF Programming Guide latest documentation,” *docs.espressif.com*. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/about.html#:~:text=The%20ESP%2DIDF%2C%20Espressif%20IoT (accessed Feb. 16, 2024).

[5] “Vue.js,” *vuejs.org*. https://vuejs.org/guide/introduction.html#:~:text=on%20VueMastery.com- (accessed Feb. 16, 2024).

Appendix 1: Software Component Diagram

A diagram of a process flow

Description automatically generated

*Figure 1. Outline of Microcontroller Software Structure*

*A diagram of a software system

Description automatically generated*

*Figure 2. Outline of Node.js Backend Software Structure*

*A diagram of a software system

Description automatically generated*

*Figure 3. Outline of Vue.js Frontend Software Structure*