Software Formalization

Year: 2019 Semester: Fall Team: 8 Project: Condiment Express

Creation Date: September 1, 2019 Last Modified: October 4, 2019

Author: Yuanqiu Tan Email:tan213@purdue.edu

Assignment Evaluation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| **Assignment-Specific Items** | | | | |
| **Third Party Software** | 5 | x2 | 10 |  |
| **Description of Components** | 5 | X3 | 15 |  |
| **Testing Plan** | 5 | x3 | 15 |  |
| **Software Component Diagram** | 5 | x4 | 20 |  |
| **Writing-Specific Items** | | | | |
| **Spelling and Grammar** | 4.5 | x2 | 9 |  |
| **Formatting and Citations** | 5 | x1 | 5 |  |
| **Figures and Graphs** | 5 | x2 | 10 |  |
| **Technical Writing Style** | 5 | x3 | 15 |  |
| **Total Score** | 99 | | | Good work |

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:

*Relevant overall comments about the paper will be included here*

1.0 Utilization of Third Party Software

We will be controlling the motor to position the condiment collector to the specified X-Y coordinate, which the process is similar to a CNC machine or 3D printer controls. The open-source library GRBL can be useful when implementing such a system [1]. The difficult part is to select which part of the GRBL library can be borrowed instead of the entire system and incorporate the useful functions into our own project. The GRBL allows use and redistribution their code when the conditions are met. There is no fee required for a license.

Another useful source for getting started with the development is the example files provided by ExpressIf on GitHub [2]. These examples provide sample use cases for GPIO, I2C protocol, UART protocol, etc. The beginning development used some of the source code and integrated them into our program.

While using the HX711 chip, we found several open source libraries online for Arduino, such as the repository provided by *bogde* [3]. Combining the library with the official documentation of the chip [4] and the datasheet for ESP32-WROOM [5], the code for reading values from the chip is developed in C language.

We are using many I2C modules such as SSD1306 chip for OLED display, SX1509 chip for GPIO expander, and others. While developing communication protocols for talking to them, the official datasheet of those chips is extremely helpful to understand how to establish communication with one another.

2.0 Description of Software Components

The computation flow is illustrated through the flowchart attached in Appendix A. Throughout the entire dispensing system, there will be several states corresponding to different stages of the dispense routine.

1. PRE-DISPENSE (shown in red) - the first state of the routine where the motor is moving the catcher from its starting location to the location of the condiments which is requested through the X-Y system.
2. DISPENSING (shown in dark blue) - the vibrating motor will vibrate causing the solid condiment pass through the opening to the catcher or the peristaltic pump pumping liquid condiments from container to the catcher. The weight sensor will be monitoring the weight being dispensed for solid condiments. Once it has enough collected in the catcher, the routine will move to the next state.
3. PREDELIVERY (shown in light blue) - the catcher move to the delivery point. This is similar to the PREDISPENSE state where all it is doing is moving the X-Y motor to the designated location.
4. DELIVERING (shown in black) - the state where the condiments are dispensed once more from the catcher to the delivery spoon where the user can take out of the machine and use it. The catcher will have an opening controlled by the motor. It will progress to the next state once the condiments have been dispensed to the spoon.
5. POSTDELIVERY (shown in purple) - the spoon will be unlocked in this state and users are free to take the condiments they requested.

One of the data structures we plan on using is a queue for storing user input commands. All user commands will be enqueued, which does not directly affect the operation of the dispensing system. The dispensing system will dequeue commands from the queue if available. The queue will be entirely implemented from scratch. The communication through BlueTooth with USART protocol. This part will be combining available code online with modification to our project/BlueTooth.

Those 5 states will be modulus design where each works independently from each other. However, PREDISPENSE state and PREDELIVERY state can share the same function call when moving the catcher but to different locations, also dispensing different liquid/solid condiments have the same hardware routine which can be handled by the same set of code. Reading the analog value from sensors can also be achieved by the same function but passing in different GPIO pins or analog channels.

Following table illustrates different functions of software and if we are using any sources.

|  |  |  |
| --- | --- | --- |
| Function Name | Functionality | Source |
| Main | Initialize all I/O pins and set up all necessary states for dispense system. Empty the queue and move the motors to homing locations. | N/A. |
| Main Loop | Handles user inputs and updates corresponding motors based on the state of the dispense routine. Call other functions to operate the machine. | N/A. |
| Read analog input | Takes in the GPIO pin and/or the channel number as the input and trigger analog-to-digital converter (ADC) to read in the sensor value. | Past software. |
| Dispense solid condiments: DISPENSING | Control the vibration motor to vibrate at the bottom of the container and let the condiments drop onto the catcher. | Documentation of the motor and StackOverflow if needed. |
| Dispense liquid condiments: DISPENSING | Control the peristaltic pump to dispense the required amount of liquid condiments by inputting the number of rotations. | Documentation of the motor and StackOverflow if needed. |
| Moving the catcher: PREDISPENSE and PREDELIVERY | Move the catcher to the designated location (given as arguments) and set the flag when it is done. | Implement similar functionality as the GRBL open-source library [1]. |
| Open the catcher and let the collected condiments drop onto the spoon for user: DELIVERY | A digital function that opens up the catcher and adds a timer to the delivery process. once the timer is up, the user will be notified and able to grab the prepared condiments from the machine. | N/A |
| Communication through BlueTooth | Use the USART protocol to communicate with the BlueTooth. | Documentation of the ESP32 module and StackOverflow if needed. |
| Displaying on OLED display | Display the message on the OLED screen to notify the user what state is processing at the moment and display the correct message once the entire routine is done. | Documentation of the OLED display and StackOverflow if needed. |

A lot of protocols we use with ESP32, the microcontroller we will be using, can be found on its official documentation, which is what we will be using a lot [6].

3.0 Testing Plan

All components will be tested as we build the machine. Most of the functions will be tested manually by giving a set of inputs and determine if it has performed correctly. The following section illustrates how each step will be tested and the priority level (1 to 4) associated with it. A higher number means higher priority.

**Initialization** - Priority 4: the initialization is the first thing it will perform at bootup, and it is hard to test without other components. The first way to test the initialization is to check the corresponding registers based on the documentation. Once that is determined to be correct, the next step will be having it tested with other functions, like reading analog inputs and controlling the motors.

**Main Loop** - Priority 4: use breakpoints to step through see if the routine behaves as illustrated in the flow chart. While stepping through breakpoints, we can also detect wrong function calls or incorrect order of operations.

**Read analog input and sensors** - Priority 2: This can be tested by using a potentiometer with a voltage source. We can manually set the potentiometer to the designated value and use the multimeter to read the actual voltage reading. Compare the voltage reading with the reading provided by the ADC.

**Dispensing solid condiments** - Priority 4: The first round of tests involves controlling the vibrating motor and cause it to vibrate for a predetermined period. This can be easily tested by only hooking up the vibrator to the microcontroller. The next phase will be assembling it onto the overall machine and attach the vibrator to the container, then test the functionality by checking if 1) vibrator is vibrating, and 2) if solid condiments are dropping from the container to the catcher.

**Dispensing liquid condiments** - Priority 4: The first round of test involves controlling the peristaltic pump and dispense a predetermined amount of liquid. The second phase will be assembling it onto the overall machine and attach it to the liquid condiment container, then test the functionality by checking if 1) the pump is rotating and pumping out liquid or not, and 2) if the amount of liquid condiment being pumped out is same as requested.

**Moving the catcher** - Priority 4: The ultimate goal of this function will be checking if the catcher can move to the designated location by giving the function the index of the desired condiments. Given that we are using stepper motors to control this, one intermediate test would be given the amount of steps/rotations, check if the motor performs as what we expect.

**Communication through BlueTooth** - Priority 3: This will be tested by 1) check if any device with BLE capability can detect the signal and 2) when user input string from another device like laptop, check if the microcontroller can receive the text string.

**Displaying on OLED display** - Priority 1: This will be tested to ensure the OLED display can show the given a test string for each scenario.

4.0 Sources Cited:

[1] GRBL (2019). *grbl/grbl*. [online] GitHub. Available at: https://github.com/grbl/grbl [Accessed 4 Sep. 2019].

[2] GitHub. (2019). *espressif/esp-idf*. [online] Available at: https://github.com/espressif/esp-idf [Accessed 5 Oct. 2019].

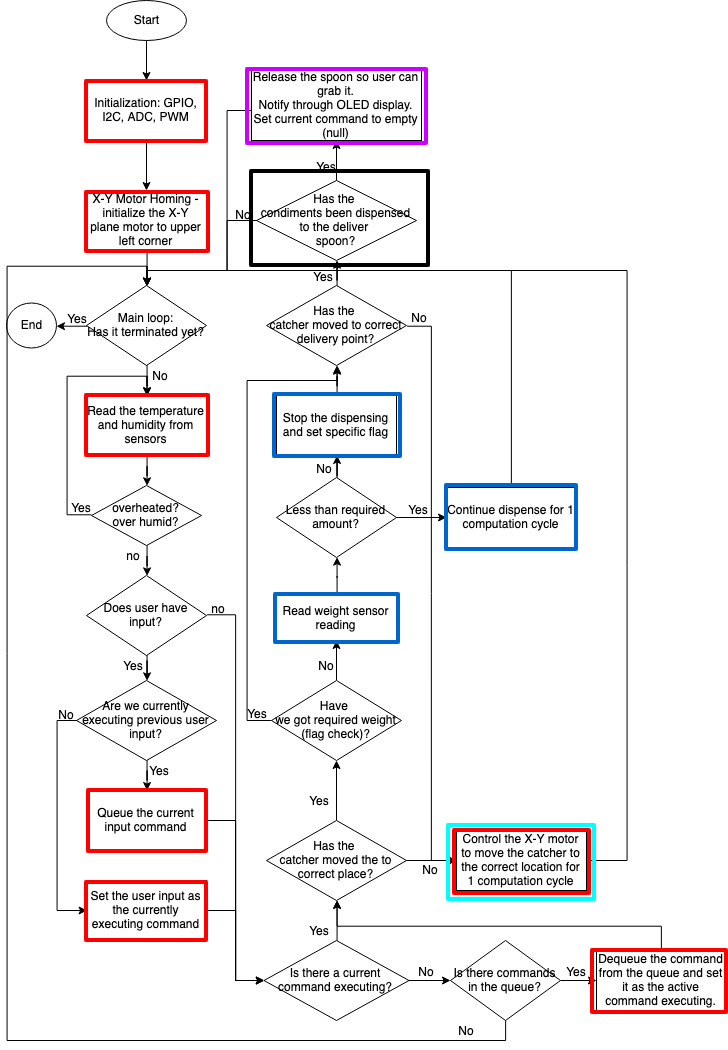
[3] GitHub. (2019). *bogde/HX711*. [online] Available at: https://github.com/bogde/HX711 [Accessed 5 Oct. 2019].

[4] Cdn.sparkfun.com. (2019). [online] Available at: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711\_english.pdf [Accessed 5 Oct. 2019].

[5] Mouser.com. (2019). [online] Available at: https://www.mouser.com/ds/2/891/esp-wroom-32\_datasheet\_en-1223836.pdf [Accessed 5 Oct. 2019].

[6] Docs.espressif.com. (2019). *API Reference — ESP-IDF Programming Guide v4.1-dev-141-ga7e8d87d3 documentation*. [online] Available at: https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/index.html [Accessed 7 Sep. 2019].

Appendix 1: Software Component Diagram - Flow Chart

**