Software Overview

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

Creation Date: September 1, 2019 Last Modified: September 9, 2019

Author: Binhan Xu Email: xu932@purdue.edu

Assignment Evaluation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| **Assignment-Specific Items** | | | | |
| **Software Overview** |  | x2 |  |  |
| **Description of Algorithms** |  | x2 |  |  |
| **Description of Data Structures** |  | x2 |  |  |
| **Program Flowcharts** |  | x3 |  |  |
| **State Machine Diagrams** |  | x3 |  |  |
| **Writing-Specific Items** | | | | |
| **Spelling and Grammar** |  | x2 |  |  |
| **Formatting and Citations** |  | x1 |  |  |
| **Figures and Graphs** |  | x2 |  |  |
| **Technical Writing Style** |  | x3 |  |  |
| **Total Score** |  | | |  |

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 Software Overview

Given that we are designing a machine that can automatically prepare condiments for users based on requests. Multiple stages will be working collaboratively to achieve that. 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. PREDISPENSE - the first state of the routine where the motor is moving the catcher from its starting location to the location of the condiments which requested through the X-Y system.
2. DISPENSING - 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 - 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 - the state where the condiments are dispensed once more from the catcher to the delivery spoon where user can take out of the machine and use. 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 - the spoon will be unlocked in this state and users are free to take the condiments they requested.

Given those states and the flow of the entire dispensing routine, expectations can be divided into the following categories:

1. Being able to communicate with computers through BlueTooth that is built into the ESP32. This requires the correct setup of the I/O on the microcontroller and using USART protocol to send/receive information between two endpoints. Another part of the communication from the software perspective is being able to parse the user command, with specific format, that what is being communicated is all ASCII text.
2. The main part of this project is to control the motors. Therefore, another expected

functionality would be to move the motor of the X-Y system to the desired location given the chosen condiment. This involves setting up the PWM signal for the motor driver and being able to drive the stepper motor by the correct number of rotations so that the catcher end up at the desired location. The peristaltic pump will also be controlled by the PWM signal to dispense liquid condiments. Being able to dispense a predetermined amount of liquid condiments by precisely control the pump with controlling the PWM duty cycle and frequency is also one of the expected functionalities.

1. The entire system also has multiple sensors that require analog-to-digital conversion: the temperature sensor, the humidity sensor, and the weight sensor. The temperature and humidity sensors will be working together to provide information regarding the environment of the condiments to users through the OLED display. The weight sensor will be working with the solid dispensing system to precisely measure the amount being dispensed. There are two software expectations: 1) being able to read in the value from the sensor using the built-in analog-to-digital converter and 2) use the reading from the weight sensor to determine whether the system should dispense more solid condiments or stop.
2. An OLED will be used as a way to display information to the user, such as the step being processed out of the entire dispensing routine. The OLED will be communicated by using the SPI protocol, some expected functionalities here are to correctly setup display through SPI and being able to display the correct information rather than random pixels.

2.0 Description of Algorithms

Giving that we need to control stepper motors to a level of high precision, we are planning on having a lookup table for the number of steps. At bootup, the stepper motors of the X-Y plane system will be “homed” to its default location (at the moment we are considering the upper left position as the default location of the X-Y system). For every condiment, the number of steps it takes from the default location to the condiment will be predetermined through trials of tests and hardcoded into the program to save real-time computation. For example, the catcher needs to move from condiment A to condiment B: the software can subtract the number of steps it takes from home location to A by that of B, returning the number of steps it takes from A to B (the idea is similar to vector addition and subtraction). This can help because the number of steps is directly proportional to the number of cycles of the PWM signal being passed to the driver, thus can determine the distance which the X-Y motor will travel. Following pseudo-code illustrates the process.

int NUM\_STEPS[ROW][COL][2] = {...}

int sign(int n) {

return n >= 0 ? 1 : -1;

}

void MoveMotor(row\_i, col\_i, row\_f, col\_f) {

// move the motor from initial row and coloum (row\_1, col\_1) to final row and column (row\_f, col\_f)

int horizontal\_move = NUM\_STEPS[row\_i][col\_i][0] - NUM\_STEPS[row\_f][col\_f][0];

int vertical\_move = NUM\_STEPS[row\_i][col\_i][1] - NUM\_STEPS[row\_f][col\_f][1];

SendPWM(sign(horizontal\_move), horizontal\_move, sign(vertical\_move), vertical\_move);

}

Since there is no heavy calculation needed for this project based on our design right now, there isn’t a lot of algorithms needed. A backup plan for moving the catcher across the X-Y plane with accuracy, in case where our design does not meet our expectation, we might try borrowing algorithms from the GRBL open-source library [1]. The GRBL library is used for controlling machines like CNC and 3D printers, which share some similarity with our project.

3.0 Description of Data Structures

One of the most important data structures what we will be using is arrays. Since we have a lot of motors that do the similar things but for different condiments (such as different peristaltic pumps for different liquid condiments), the best way is to store them into arrays and access them by index instead of having different variables with different naming for each variable. Using array can make everything more organized and easier to build a single function routine that can be reused more frequently by simply changing the index of the array that should be accessed.

Together with the array, enum will also be used to make the code cleaner and more readable. Continue with the previous example, we can create an array of different PWM channels for different peristaltic pumps and an enum that group them together through a common naming system:

enum LIQUID\_CONDIMENTS {

OIL,

VINEGAR,

SOY

};

int PERISTALTIC\_PWM\_CHANNEL[LIQUID\_CONDIMENTS];

In this way, if we want to access the PWN channel for vinegar, we can directly do PERISTALTIC\_PWM\_CHANNEL[VINEGAR] making the code easier to read and develop.

Another main data structure we will be using is a queue. Given that the system can only process condiments one by one due to hardware limitation, we will be queueing user commands into a queue and process them one by one. The queue will also be realized by using a fixed-sized array with circular indexing; therefore can only take on a certain number of commands instead of infinite of them. Two “pointers” (or simply integer recording the indices) will be used to record the start and the end of the queue, preventing them from being overflowed. Part of the information of the queue will also be displayed on the OLED panel.

4.0 Sources Cited:

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

Appendix 1: Program Flowcharts

*A close up of a map

Description automatically generated*

Appendix 2: State Machine Diagrams

*A close up of a logo

Description automatically generated*