Mechatronics Final Project

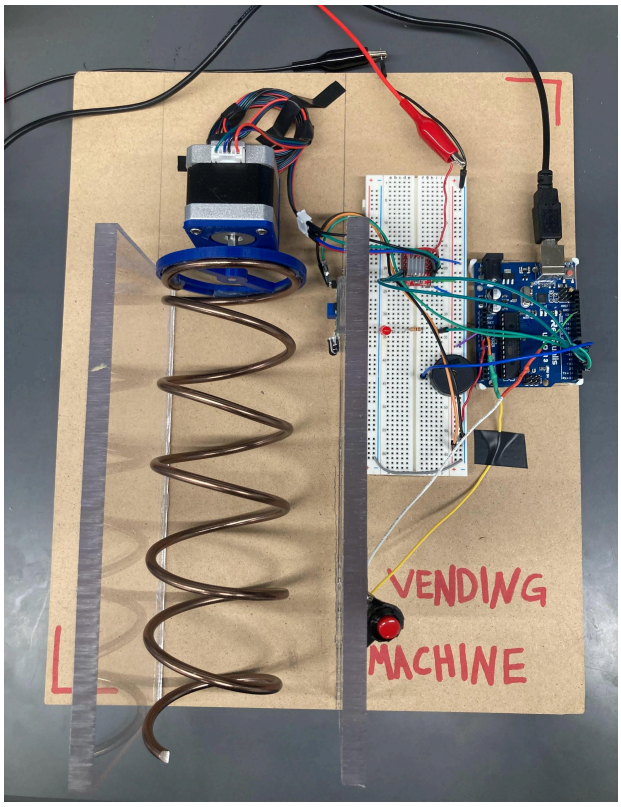
MCE433 – Musa Jouaneh

T.A. – Jacob Travisino

By: Jessica Tingley and Betty Hasse

Introduction:

This project involves the development and construction of a mechatronic system that resembles a vending machine. It consists of an aluminum dispensing coil, affixed to a NEMA 17 Bipolar Stepper motor. The motor is operated by way of an Arduino, breadboard, and A4988 Stepper Driver. The vending itself is controlled by an on-screen GUI that incorporates options to dispense different quantities of product from the coil, and keeps inventory of the system.



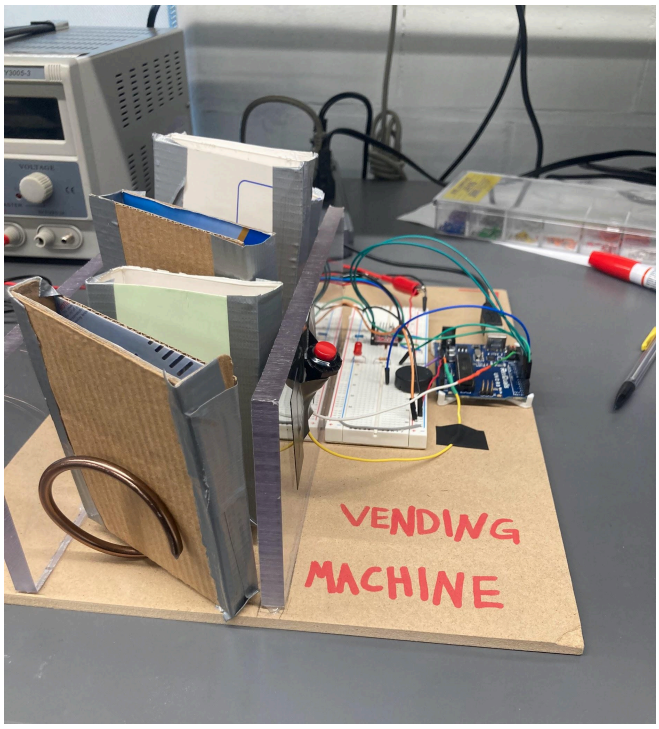


Figure 1: Top view of machine.

Figure 2: Isometric view of machine.

This system includes an infrared optical proximity sensor that is triggered when the sensing coil is empty – nothing is blocking its beam. When the sensor reads that the "vending machine" is empty, an LED and buzzer are triggered, prompting the user to begin the procedure of restocking the machine. To do this, the user must select the "restock" option on the corresponding GUI, this activates the stepper motor, but in the opposite direction from what is

undergone in the dispensing procedure. Once the coil is fully restocked with product, the physical reset button incorporated in the system can be pressed, completing the restock procedure and allowing normal dispensing to resume.

The GUI also contains a chart that tracks purchasing trends recorded by the machine. This was created by writing the number of items dispensed and the time of dispensing to a CSV file during operation and reading from it when updating the graph.

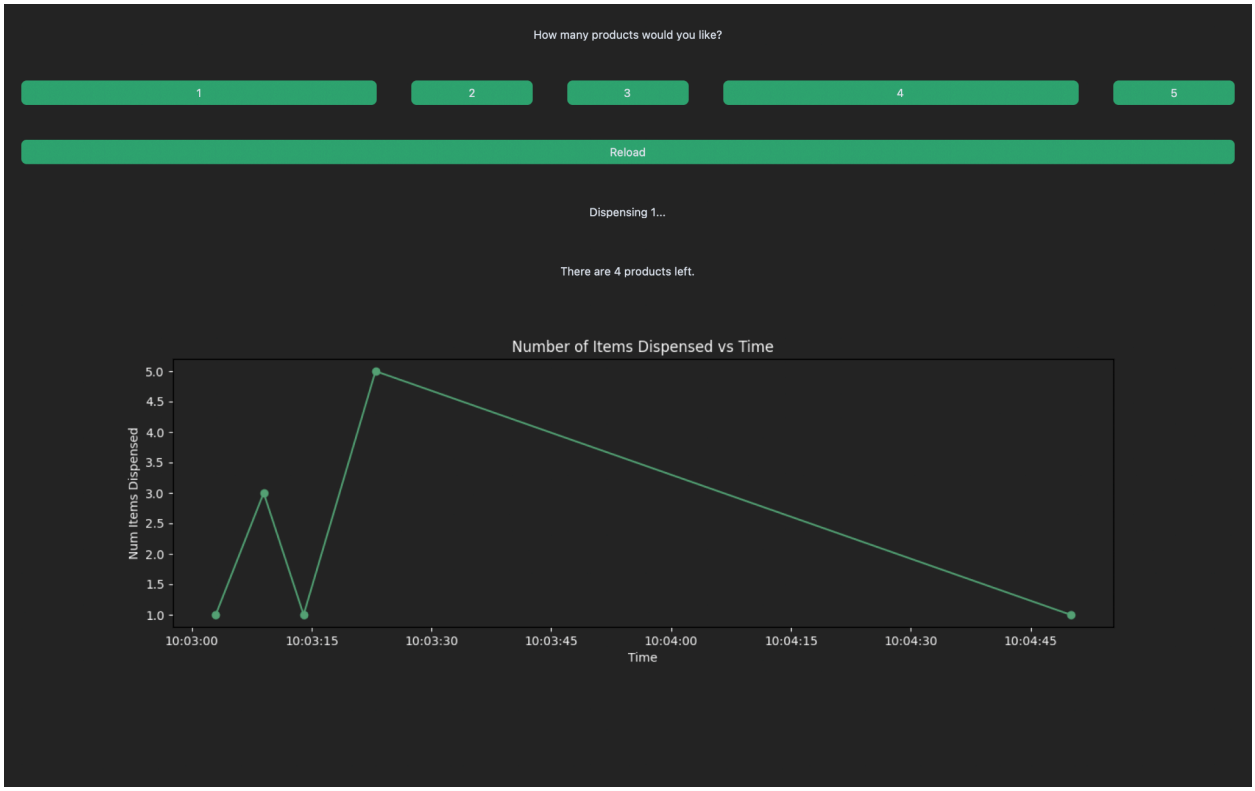


Figure 3: GUI with buttons and graph.

Approach:

To carry out the formerly described functions, we developed the project in the following manner:

We began by writing code for the stepper motor and testing this function with a prototype of the dispensing coil to demonstrate that this concept would be functional in operation. We

noted that it did indeed work as expected, moving products towards and away from the end of the coil in accordance with the direction of rotation specified. We then mocked-up different infrared sensor configurations in the dispensing area. We knew we needed the sensor to be triggered by the product, but not by the coil itself. To do this, we decided to widen the dispensing area slightly, shortening the receiving range of the sensor with the native control screw, and installed the sensor in the bottom-corner of the apparatus outside of the coil. We 3D printed custom SolidWorks models of appropriate mounts for the stepper, dispensing coil, and Arduino. We soldered leads of appropriate length to the reset button. After this, we mounted all aspects of the system to a wooden board base, and altered the system to make it appear tidy and function reliably.

Once the desired sequence of events was identified, various states were created and coded to realize this process. The system spends most of its time in the 'Idle' state, where it only needs to check for some trigger, such as the proximity sensor not detecting an object or a button press. When a dispense command is received, the system moves to the dispensing state and matches the number of items to be dispensed, as indicated by the command, to the necessary number of revolutions to achieve this. A PWM signal is then written to the stepper motor for an amount of time as determined by the equation: $\frac{Steps\ per\ Revolution}{Pin\ Frequency} * Number\ of\ Revolutions * 1000$.

If the reload command of 'b' is received in the 'Idle' state, the system then moves to the 'Reloading' state wherein the stepper direction is written to 'Low' to change the direction. A PWM signal is sent to the stepper for the amount of time determined by:

$\frac{Steps\ per\ Revolution}{Pin\ Frequency} * 1000$. If the proximity sensor is triggered from the 'Idle' state, the system transitions to the 'Empty' state where a buzzer is triggered for two seconds and an LED is turned on. From this state, the user may only choose to reload the system. Once the system is reloaded,

the physical button must be pressed to reset the system before product can be dispensed. In this ‘Reset’ state, the reading of the proximity sensor is checked to be low before the LED is turned off and the system resumes normal operation. Aside from the ‘Empty’ state, all states send operation back to ‘Idle’ upon their completion.

The GUI provides seamless functionality for the user and tracks various system components throughout operation. During operation, a CSV is populated with the number of products left, number of products dispensed, and the time at which said products were dispensed. This CSV ensures that, in the case of power failure, the system can reboot knowing how many items are left. The CSV, a line graph, and a running number of products variable are updated each time a button is pressed. The GUI also communicates with Arduino through serial communication to monitor whether or not a system reset is necessary. This ensures that the GUI does not falsely update product counts when no product is actually being dispensed.

Conclusion:

Overall, this project was successful. We saw the construction of the machine through from start to finish, compiling physical components and code into a fully functioning system. Although we encountered some issues in compiling the code, mainly due to its complexity and the number of devices we were trying to incorporate, we were able to overcome them by consulting the Professor and T.A. of the class. We faced some hardware issues initially – we burnt out one motor driver and one buzzer – but we learned our lesson here. We were eventually able to wire everything correctly and move past our mistakes.

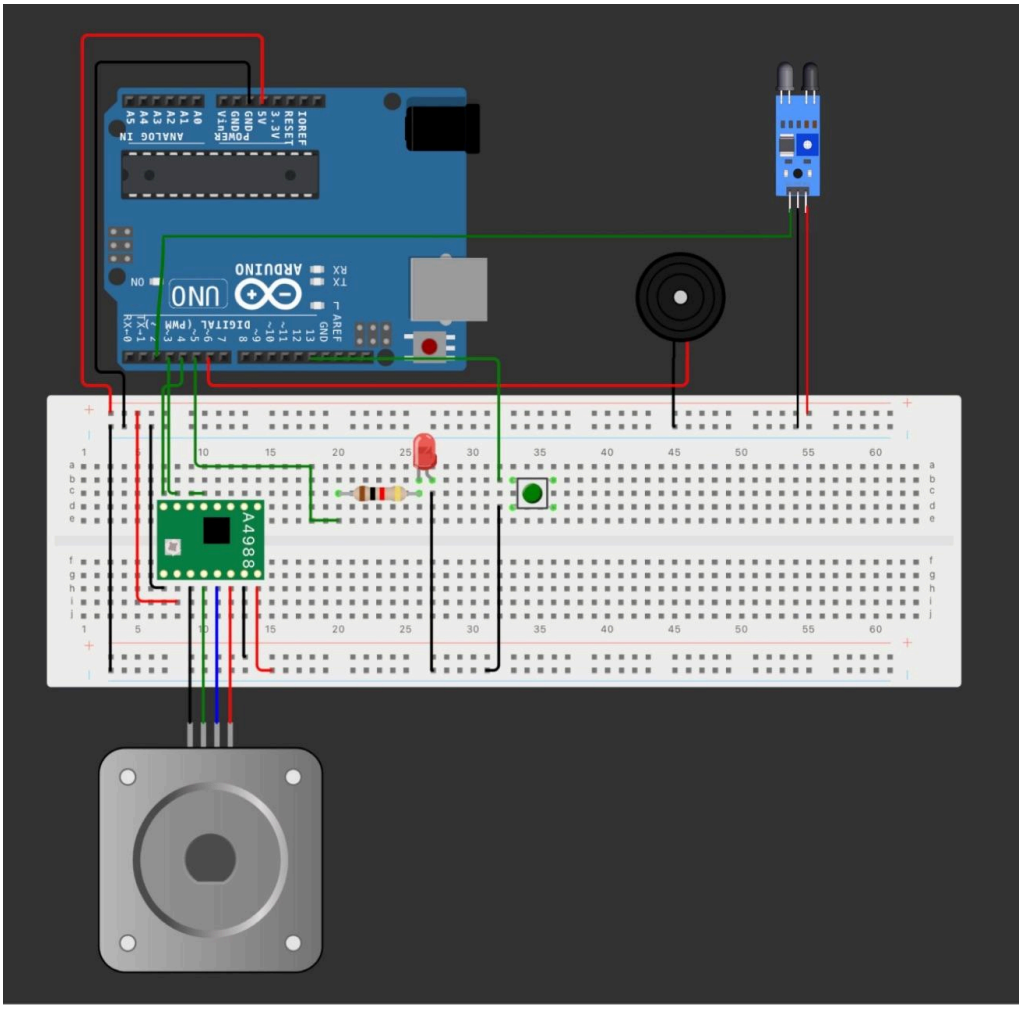
Physical Components:

- Arduino Mega
- Nema 17 Bipolar Stepper
- A4988 Stepper Driver
- EK 1254 Sensor
- LED
- Buzzer
- Button

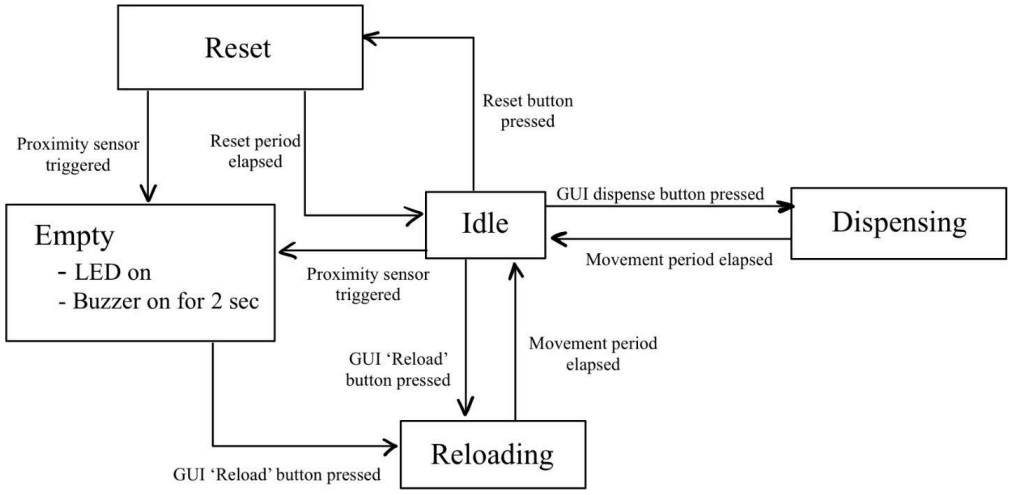
Processes Used

- PWM actuation
- Timing
- Digital input/digital output

Circuit Diagram:



State Transition Diagram:



Code:

a) State Machine (C):

```
void ControlTask(void){
 int x = 0;
 int reading = 0;

 switch(nextState){
   // IDLE STATE
   case idle:
     // Action
     reading = digitalRead(sensorPin);

     // Exit
     if(reading){
       nextState = empty;  // No object detected -- go to empty state
     }
     else if(command == 'b') {
       nextState = reloading;  // Reverse button pressed -- go to reverse state
     }

     else if(!digitalRead(buttonPin)){
       nextState = reset;  // Reset button pressed -- reset system
     }
```

```
    else if(command != 'n') {
      // Ensure system is reset before dispensing again
      if(LEDVal){}
      else{
        nextState = dispensing;  // Dispense button pressed -- go to dispensing state
      }
    }
    break;

  // DISPENSING STATE
  case dispensing:
    // Entry
    if(!dispFlag){
      // Match number of items to be dispensed with number of revolutions to turn
      switch(command){
      case '1': numRevs = 1.; break;
      case '2': numRevs = 2.; break;
      case '3': numRevs = 3.; break;
      case '4': numRevs = 4.; break;
      case '5': numRevs = 5.; break;
      default:
        command = 'n';
        nextState = idle;
        return;
      }
      dispFlag = 1;
      // Prep for stepper movement
      stepCount = 0;
      digitalWrite(dirPin, HIGH);
      stepperStart = millis();
    }

    // Action
    analogWrite(stepPin, 128);  // PWM to stepper motor

    // Exit
    if((millis() - stepperStart) >= ((stepsPerRev/pinThreeFreq)*numRevs)*1000){
      analogWrite(stepPin, 0);
      command = 'n';
      nextState = idle;
      dispFlag = 0;
    }
```

```
  break;

// RELOADING STATE
case reloading:
  // Entry
  if(!revFlag){
    // Prep for stepper movement
    digitalWrite(dirPin, LOW);
    revFlag = 1;
    revStart = millis();
  }

  // Action
  analogWrite(stepPin, 128);  // PWM to stepper motor

  // Exit
  if((millis() - revStart) >= (stepsPerRev/pinThreeFreq)*1000){
    analogWrite(stepPin, 0);
    command = 'n';
    nextState = idle;
    revFlag = 0;
  }
  break;

// EMPTY STATE
case empty:
  // Entry
  if(!buzzFlag){
    tone(buzzerPin, 262, 2000);
    digitalWrite(LEDPin, HIGH);
    LEDVal = HIGH;
    buzzFlag = 1;
  }

  // Exit -- may only exit to reloading state
  if(command == 'b'){
    nextState = reloading;
  }
  break;

// RESET STATE
case reset:
```

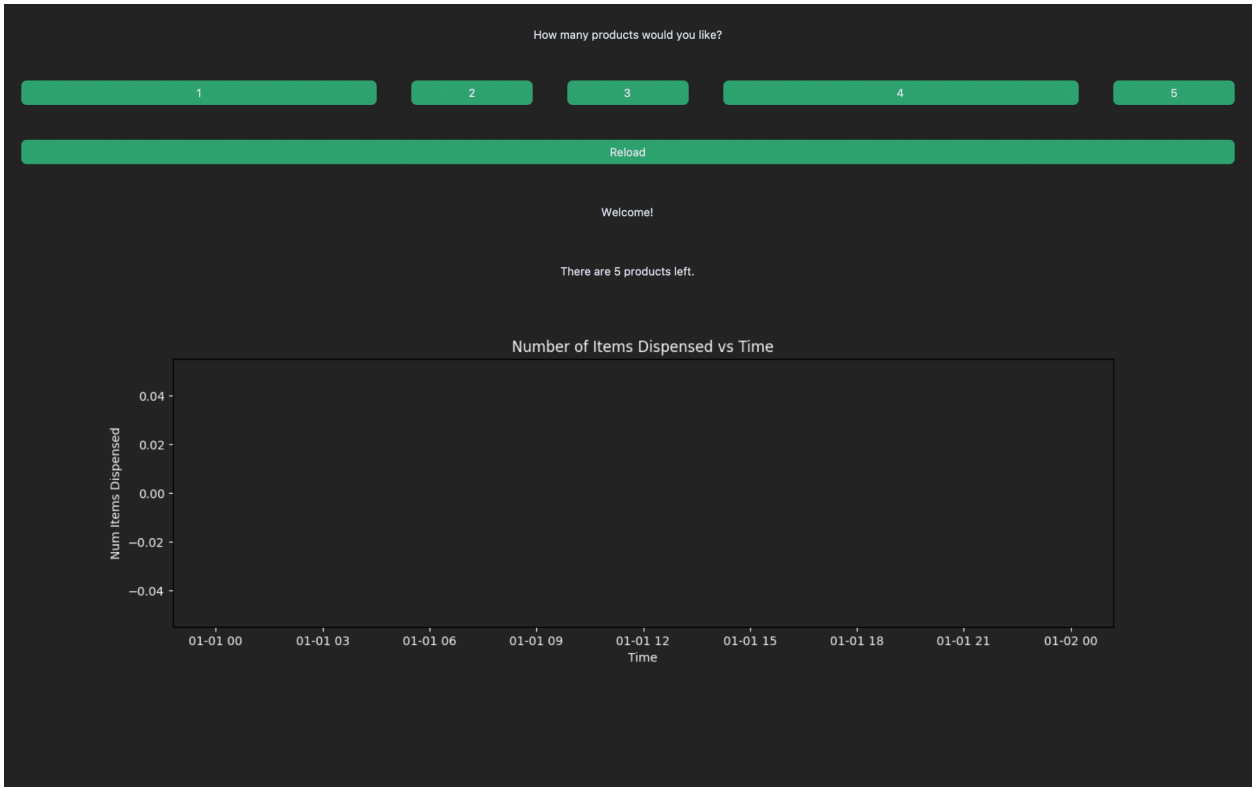
```
    // Action
    reading = digitalRead(sensorPin);

    // Exit -- check that system is no longer empty
    if(!reading){
      digitalWrite(LEDPin, LOW);
      LEDVal = LOW;
      buzzFlag = 0;
      nextState = idle;
    }
    else{
      nextState = idle;
    }

  default:
    nextState = idle;

 }
}
```

b) GUI (Python)



How many products would you like?

1 2 3 4 5

Reload

Must reset system!

There are 0 products left.