ME102B (Prof. Stuart)
Oniqa Urmi, Alejandro Diaz, SooMin Kang
December 12, 2021

Automated Rice Dispenser

Opportunity

In reducing the time and effort needed to perform mundane household tasks, including preparing to cook, we thought of innovative designs. Among feasible choices including a rice cooker, fridge organizer, automatic vegetable slicer, etc. we chose to emphasize being able to predetermine an amount of ingredient needed and dispensing. In the case of our developed concept—the automated rice dispenser—we will be controlling the amount of rice and dispensing it from storage.

High Level Strategy

1. The rice dispenser will stay idle on a table top with 1 ESP32 microcontroller, 2 LEDs, 3 push buttons, 1 potentiometer, 1 ultrasonic sensor
    a. If button 1 is pressed, the rice dispenser will turn on (turning led1 on)
    b. Using a potentiometer, we can change the amount of rice that is dispense—choosing between 1cup (range of 0-2048) and 2cups (range of 2048-4095)
    c. If button 2 is pressed, the rice dispenser will dispense by starting/rewinding motor with and without a delay (1 cup — assigned steps ; 2 cups — assigned steps with a delay).
    d. Using an ultrasonic sensor, the distance from the sensor to the level of rice will be sensed. If there is no rice left (reaches the distance between sensor and the bottom of the container), the led2 will turn on, which alerts the user to refill the container.
    e. If button 3 is pressed, then the led2 turns off.
2. The rice dispenser will detect whether there is sufficient rice to dispense.
3. The rice dispenser will dispense the desired amount of rice that the user needs.
4. Special considerations were not included due to complexity of the electrical design (coding and integrating the multiple controls), but initially dealt with connecting the rice dispenser to a rice cooker that will direct it to cook automatically.

From our initial desired functionality, we were able to achieve all of the desired functions. We included additional buttons to make the system more clear. Also, we specified the steps of the stepper motor.

Integrated Physical Device

Figure 1: Final Product at Expo

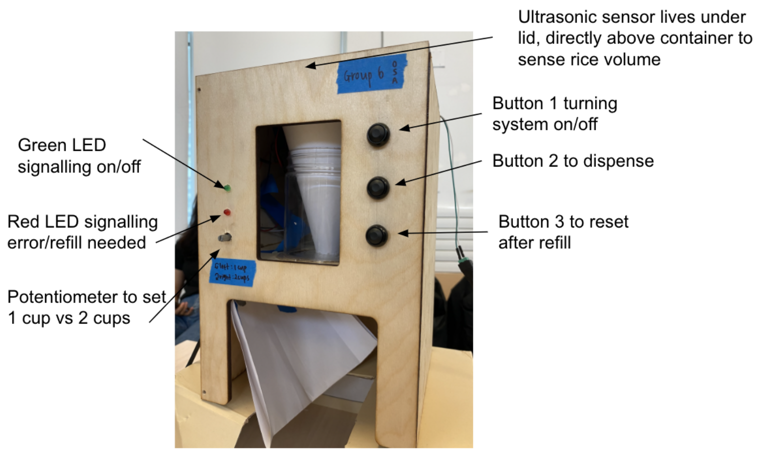
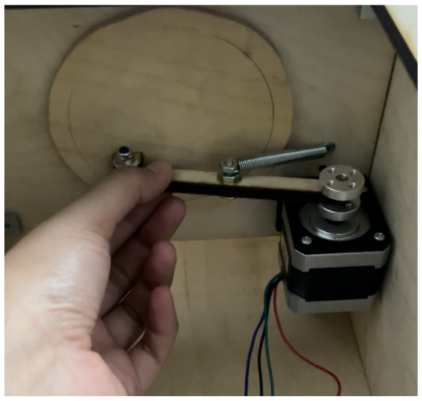

Ultrasonic sensor lives under lid, directly above container to sense rice volume
Button 1 turning system on/off
Button 2 to dispense
Button 3 to reset after refill
Green LED signalling on/off
Red LED signalling error/refill needed
Potentiometer to set 1 cup vs 2 cups
Group 6

Figure 2: Actuation Mechanism



The trap door is attached to a linkage that is turned via a stepper motor. It is programmed to then rotate by 90 degrees for a predetermined amount of time

<u>Function-Critical Decisions</u>

The main function-critical decisions revolved heavily around the dispensing mechanism and actuation. To actuate the trap door between its dispensing and holding positions required a motor—we decided on a stepper motor due to its stable stationary position. Stepper motors provide a holding torque that is sufficient enough to hold the trap door closed against a full load of rice. We can also easily control the position by commanding the number of steps that results in the door being held open at a desired angle. A closed loop control is not needed, and we can spare the expense of an encoder.

To spec out the proper sized motor, it is important to properly define the loads that the motor would see, then add a small factor of safety to account for manufacturing errors. The load case that would result in failure of the assembly would be when the system is fully loaded with rice in a stationary closed position. Although the system only dispenses 1 or 2 cups of rice, we decided that choosing the motor to hold a 2.5 cup load would give us a large enough safety margin as well as a margin to allow for a larger storage container in the assembly. This load is to be distributed evenly across the 5” (127mm) wide platform. A static analysis was performed on an equivalent cantilever beam fixed at one end to replicate the platform (beam) and motor shaft (fixed end).

Assumptions:

- 2.5 cups maximum holding capacity (~500g)
- 500g distributed load
- 5” (127mm) long cantilever beam
- FBD as follows:

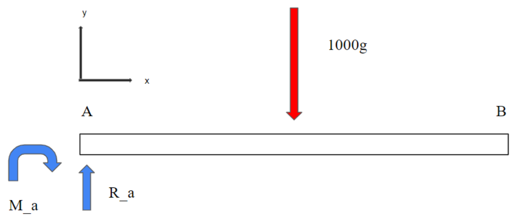


Figure 3: Motor Load FBD

Force and Moment Equivalencies:

$$\sum F_y = -500 + R_A[g] = 0$$

$$\rightarrow R_A = 500g$$

$$\sum M_A = -500 * (127/2) + M_A = 0$$

$$\rightarrow M_A = 31750g * mm = 0.31Nm$$

The force calculations tell us that our motor requires 0.31Nm of torque capability and a 500g (4.9N) radial load capacity. According to the datasheet for a NEMA 17 stepper motor, we fall just within the torque capability and well within the max radial load of 28N. We feel comfortable choosing this motor as the 2.5 cup load has a built in safety margin (refer to Appendix A, Figure 6).

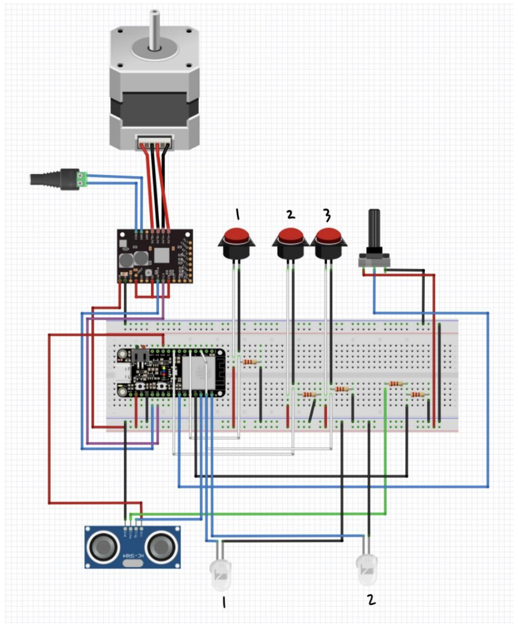


Figure 4: Final Circuit Diagram

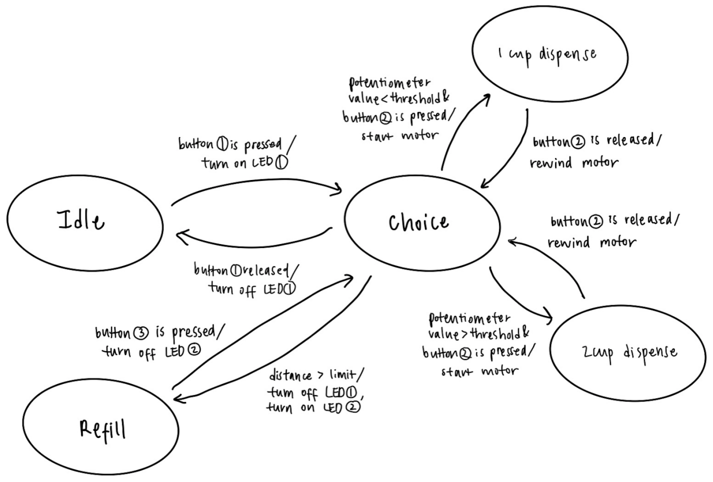

Idle
Choice
1 cup dispense
2cup dispense
Refill
button ① is pressed / turn on LED ①
button ① released / turn off LED ①
potentiometer value < threshold & button ② is pressed / start motor
button ② is released / rewind motor
button ② is released / rewind motor
potentiometer value > threshold & button ② is pressed / start motor
button ③ is pressed / turn off LED ②
distance > limit / turn off LED ①, turn on LED ②

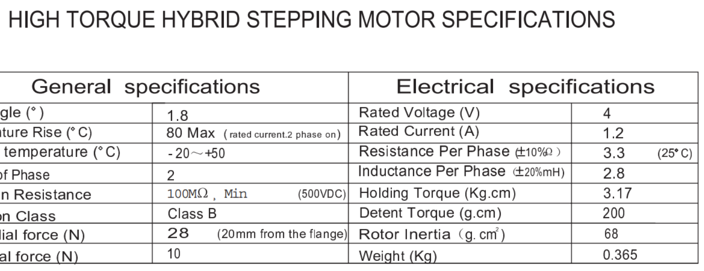
Figure 5: Final State Transition Diagram

<u>Advice for Future Students</u>

The biggest difficulty we had was to actuate the motor. While we wanted to resort with the parts within our kit, we figured that the stepper motor would be more appropriate. We had to independently learn the concepts and apply it. There was also a lot of time spent debugging, integrating debounce, and making a solid code; however, coding step-by-step with the state diagram helped us finalize the code.

Appendix

A. Stepper Motor Data Sheet


HIGH TORQUE HYBRID STEPPING MOTOR SPECIFICATIONS

General specifications		Electrical specifications	
Step Angle (°)	1.8	Rated Voltage (V)	4
Temperature Rise (°C)	80 Max (rated current.2 phase on)	Rated Current (A)	1.2
Ambient temperature (°C)	-20～+50	Resistance Per Phase (±10%Ω)	3.3 (25°C)
Number of Phase	2	Inductance Per Phase (±20%mH)	2.8
Insulation Resistance	100MΩ, Min (500VDC)	Holding Torque (Kg.cm)	3.17
Insulation Class	Class B	Detent Torque (g.cm)	200
Max.radial force (N)	28 (20mm from the flange)	Rotor Inertia (g. cm^2)	68
Max.axial force (N)	10	Weight (Kg)	0.365

Figure 6: Pololu Stepper Motor Datasheet

B. Bill of Materials

Part	Functionality	Quantity	UOM	Cost Tot.	Already Have
Plywood - 1/4" x 24" x 48"	Housing/Enclosure	2	EA	$26.64	
Plastic Container 5 ⅛" x 7" x 11"	Rice Container	2	EA	$4.10	
Plywood 1/4" x 12" x 24"	Dispense Mechanism (Platform + Linkage)	2	EA	$3.34	
ESP32 Arduino	Microcontroller, Power Source	1	EA	N/A	o
Ultrasonic Sensor	To detect rice level/amount	1	EA	N/A	o
LED	Alerts if rice container is empty	2	EA	N/A	o
Button	Press to dispense	3	EA	$16.00	o
Potentiometer	For desired amount to dispense	1	EA	N/A	o
Pololu Stepper Motor (NEMA 17)	To actuate dispensing mechanism	1	EA	$21.95	
Pololu Stepper Motor Driver	To drive/control stepper motor	1	EA	$11.95	
Rice	Prototyping, Functionality Check	1	5-LB Bag	$6.89	o

Wood Glue	Adhesive for enclosure setup and mounting container to enclosure	1	1	$5.20	
Handle	Handle for user to open and refill rice	1	EA	$3.80	
Hinges	Enclosure Opening and Dispense Doors	4	EA	$8.56	
¼ - 20 Screws	Mounting handles and linkages	8	25 pack	$10.45	o
¼ - 20 Hex Nuts	Mounting Handles and linkages	8	100 pack	$5.56	o
Shaft Collar	Linkage held in place	1	EA	$5.11	

<u>C. CAD Model</u>

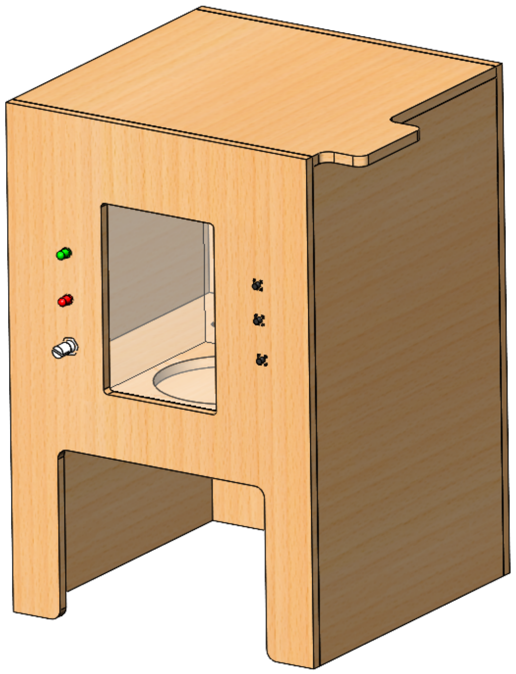


Figure 7: 3D CAD Model of Final Product

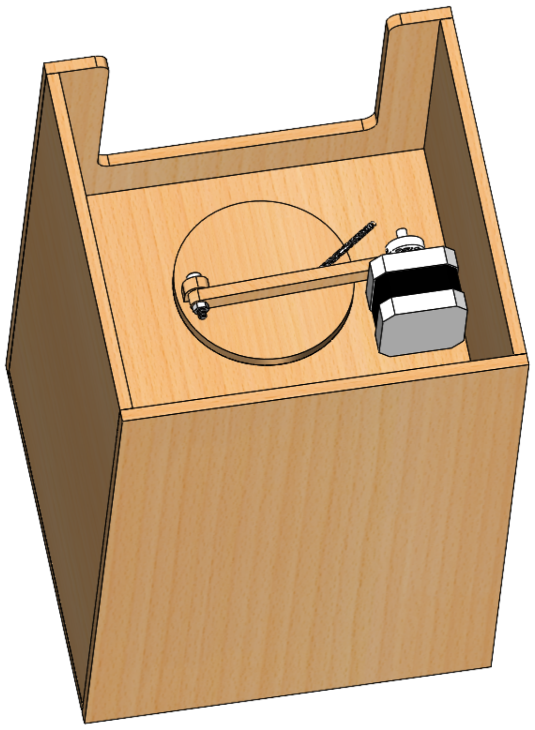


Figure 8: Bottom View of the Trap Door and Motor

D. Code

```
#include <Arduino.h>
#include "BasicStepperDriver.h"

//Define constants ------------------------------------
#define BTN 15  // declare the button ED pin number
#define BTN2 12
#define BTN3 14
#define LED_PIN1 17
#define LED_PIN2 21
#define DIR 26 //A0
#define STEP 25 //A1
#define POT 4
#define trigPin 16
#define echoPin 19
#define SOUND_SPEED 0.034
#define CM_TO_INCH 0.393701
#define MOTOR_STEPS 200
#define RPM 10
#define MICROSTEPS 1

BasicStepperDriver stepper(MOTOR_STEPS, DIR, STEP);

//PWM properties
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int pwmChannel = 0;
const int resolution = 8;
int MAX_PWM_VOLTAGE = 255;
float distanceCm;
//Setup variables ------------------------------------
volatile bool buttonIsPressed = false;
volatile bool button2IsPressed = false;
volatile bool button3IsPressed = false;
volatile bool interruptCounter = false;    // check timer interrupt
volatile bool EnoughRice = false;
```

```
//Timer Variables
int totalInterrupts;   // counts the number of triggering of the alarm
hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;


int state = 0;
int distance;
long duration;
float refillduration;
float refilldistance = 1;

//Initialization ------------------------------------
void IRAM_ATTR onTime() {
  portENTER_CRITICAL_ISR(&timerMux);
  interruptCounter = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux);
  timerStop(timer);
}
void IRAM_ATTR isr() {
  buttonIsPressed = true;
}
void IRAM_ATTR isr2() {
  button2IsPressed = true;
}
void IRAM_ATTR isr3() {
  button3IsPressed = true;
}
void TimerInterruptInit() {  //The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer = timerBegin(0, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics / sec
  timerAttachInterrupt(timer, &onTime, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(timer, 1000000, true);          // sets how many tics will you count to trigger the interrupt
  timerAlarmEnable(timer); // Enables timer
}



void setup() {
  // put your setup code here, to run once:
  distance = analogRead(POT);
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input

  pinMode(POT, INPUT);
  pinMode(LED_PIN1, OUTPUT);
  pinMode(LED_PIN2, OUTPUT);
  pinMode(BTN3, INPUT);
  pinMode(BTN2, INPUT);
  pinMode(BTN, INPUT);  // configures the specified pin to behave either as an input or an output
  attachInterrupt(BTN3, isr3, RISING);
  attachInterrupt(BTN2, isr2, RISING);
  attachInterrupt(BTN, isr, RISING);  // set the "BTN" pin as the interrupt pin; call function named "isr" when the interrupt is triggered; "Rising" means triggering interrupt when the pin goes from LOW
  Serial.begin(115200);
  TimerInterruptInit();
  timerStop(timer); // ADDED
  digitalWrite(LED_PIN1, LOW);


  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);

  // attach the channel to the GPIO to be controlled
  //ledcAttachPin(BIN1, ledChannel_1);
  // ledcAttachPin(BIN2, ledChannel_2);

  stepper.begin(RPM, MICROSTEPS);

}
```

```
//Main loop ------------------------------------
void loop() {

switch (state) {
  case 0: //case 0 checks if Button 1 is pressed. If it is pressed, then the green LED light turns on and state switches to 1. Prints System is now ON, turn Dial to dispense 1 or 2 aa
    if (CheckforButtonPress()) {
      led1_on();
      buttonIsPressed = false;
      state = 1;
      Serial.println("System On, turn dial to dispense 1 or 2 cups!");
      Serial.println("Press Button 2 to Dispense Rice");
    }
    break;
  case 1:  // case 1 checks if button 1 (turn off button) is clicked. If yes, then turn off system and return to case 0. If not, continues to next check.
    distance = analogRead(POT);
    Serial.println(refilldistance);
    //Serial.print("Distance (inch): ");
    //Serial.println(refilldistance);
    if (CheckforButtonPress() == true) {
      led1_off();
      buttonIsPressed = false;
      state = 0;
    }

    if (NeedmoreRice() == true) { // This if checks if rice needs refill. If needs refill, state switches to 4.  If rice does not need refill, continue to next check.
      led1_off();
      led2_on();
      state = 4;
   }


     if (CheckforButtonPress2() == true) { // This if checks if button 2 (dispense button) is clicked. If yes, then check distance. If not, stay in state 1.
      //DOES THERE NEED TO BE A STATE ASSIGNMENT HERE?, WHAT STATE are we in, if none of these are true?
      if (CheckForDistancePot1() == true) { //If this is true, move to state 2 and dispense 1 cup.
      state = 2;
      button2IsPressed = true;
      }
```

```
     if (CheckforButtonPress2() == true) { // This if checks if button 2 (dispense button) is clicked. If yes, then check distance. If not, stay in state 1.
      //DOES THERE NEED TO BE A STATE ASSIGNMENT HERE?, WHAT STATE are we in, if none of these are true?
      if (CheckForDistancePot1() == true) { //If this is true, move to state 2 and dispense 1 cup.
      state = 2;
      button2IsPressed = true;
      }
```

```
      else if (CheckForDistancePot2() == true) { //If this is true, move to state 3 and dispense 1 cup
      state = 3;
      button2IsPressed = true;
      }
      break;
    }

    break;
   case 2: //dispenses 1 cup
       if(CheckforButtonPress2() == true) {
         Serial.println("1 cup of rice now dispensing");
         startMotorResponse1();
         rewindMotorResponse1();
         button2IsPressed = false;
         state = 1;
      }
    break;

    case 3: //dispenses 2 cups
    if (CheckForDistancePot2() == true) {
       if(CheckforButtonPress2() == true) {
         Serial.println("2 cups of rice now dispensing");
         startMotorResponse2();
         rewindMotorResponse2();
         button2IsPressed = false;
         state = 1;
      }
    }
    break;
    case 4: //if rice needs refill, then ask user to click button 3 to dismiss and return to main menu
    Serial.println("Refill needed, press Button 3 to dismiss error and return to main menu");
    if (CheckforButtonPress3() == true) { //"If Button 3 is clicked, turn off Alert light and return to state 1"
      led2_off();
      led1_on();
      state = 1;
      button3IsPressed = false;
    }
}

}

void Refill() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(trigPin, LOW);

  refillduration = pulseIn(echoPin, HIGH);
  distanceCm = refillduration * SOUND_SPEED/2;
  refilldistance = distanceCm * CM_TO_INCH;
  delay(1000);
}
// EVENT CHECKERS
bool NeedmoreRice() {
  Refill();
  if (refilldistance > 6.0) { //Change this Value when Sensor is mounted
    Serial.println("Need to refill");
    return true;
  }
  else {
    return false;
  }
}


bool CheckForDistancePot1() {
  Serial.println(distance);
  if (distance < 2048) {
    return true;
  }
  else {
    return false;
}
}
bool CheckForDistancePot2() {
  int distance = analogRead(POT);
  Serial.println(distance);
  if (distance > 2048) {
    Serial.println("2 cups dispensing");
    return true;
  }
  else {
    return false;
}
}

boolean CheckforButtonPress(){
  if (buttonIsPressed == true && timerStarted(timer) == false) {
    return true;
  }
  else {
    return false;
  }
  }

boolean CheckforButtonPress2(){
  if (button2IsPressed == true && timerStarted(timer) == false) {
    return true;
  }
  else {
    return false;
  }
```

```
boolean CheckforButtonPress3(){
  if (button3IsPressed == true && timerStarted(timer) == false) {
    return true;
  }
  else {
    return false;
  }
  }

void ButtonResponse(){
    Serial.println("Pressed!");
    buttonIsPressed = false;
    led1_on();
  }

boolean CheckforTimeDone() {
  if (interruptCounter) {
     return true;
  }
  else {
    return false;
    }
  }
void TimeDoneResponse() {
  if (interruptCounter) {
    portENTER_CRITICAL(&timerMux);
    interruptCounter = false;
    portEXIT_CRITICAL(&timerMux);
    timerStop(timer);
  }
}



// MOTOR and LED Response
void led1_on(){
  digitalWrite(LED_PIN1, HIGH);
}

void led1_off(){
  digitalWrite(LED_PIN1, LOW);
}

void led2_on(){
  digitalWrite(LED_PIN2, HIGH);
}

void led2_off(){
  digitalWrite(LED_PIN2, LOW);
}

void startMotorResponse1() {
  stepper.rotate(-80);
}

void rewindMotorResponse1() {
  stepper.rotate(80);
}

void startMotorResponse2() {
  stepper.rotate(-80);
  delay(2000);
}

void rewindMotorResponse2() {
  stepper.rotate(80);
  delay(2000);
}
```