TreaToss Project Report

John Kenny, Donghun Lee and Luke Seybold
ME 102B: Mechatronics Design, Fall 2022
Date: 12/6/2022

1. Opportunity

The TreaToss Pet Food Distributor is a device which allows for automated pre-proportioned pet treat distribution. Smart pet devices are becoming a rapidly growing niche of commerce throughout the world as technology advances and pet owners seek to give their pets a life of comfort. Smart devices, such as the pet food distributor, have become increasingly in demand. We have focused on the opportunity to allow pet owners to distribute pet treats in a controlled manner at the push of a button. Handling pet treats can be an undesirable task for many pet owners and the ability to distribute a batch of treats to their animal at the push of a button, while keeping their hands clean is the exact purpose of our device.

2. High-level strategy

The high-level strategy of this device was to create a treat bowl for uniformly sized dog treats and design a rotating mechanism which allows these treats to exit their initial holding place through the pull of gravity. The treats will fall in groups of (1-6) and land in the designated pet reception area. Then, expectant pets will be able to consume these treats straight from the platform on which they land. The device is controlled by a potentiometer which has four setting levels. With each setting level, the distribution wheel makes a 90 degree turn, allowing one plate hole to overlap with the distribution chute of the bowl. The settings are as follows: potentiometer at the first threshold one treat holder passes distribution chute (90^o) and about 1-2 treats are released. At the second threshold, two treat holders pass (180^o) the distribution chute and approximately (2-4) treats are distributed. At the third threshold, three treat holders pass (270^o) the distribution chute and approximately (3-6) treats are distributed. And finally, at the fourth stage, all four treat holders pass (360^o) the distribution chute and approximately (4-7) treats are distributed. The device will be run on a single DC motor and the actuation will be the result of a button being pressed. One press will allow for one batch of treats to be released at the current setting of the potentiometer. Initially, the device was intended to launch treats, one by one. The device was also intended to be controlled remotely, to allow for away pet owners to interact with their animals when remote. But the decision to confine the device to a locally actuated design where treats are dropped for pets to consume was the final result. The justification for this simplification came from the realization that whether the treat was launched or dropped is of no difference to the recipient, only that the treat was distributed. The feeding mechanism which administers the treats was modified from the original ‘sweeping arm’ idea to a holed plate which, when aligned, allowed treats to drop through the open section. The modification here was that the number of treats is inconsistent but allows for a more smooth and reliable distribution. Also, calculated forces became more complex with a rotating arm, given the randomness of the treats in the bowl. With a flat plate design, the treats need only to fall to the bottom of the plate and settle into one of the many holes, which will eventually rotate and distribute them.

3. Assembly

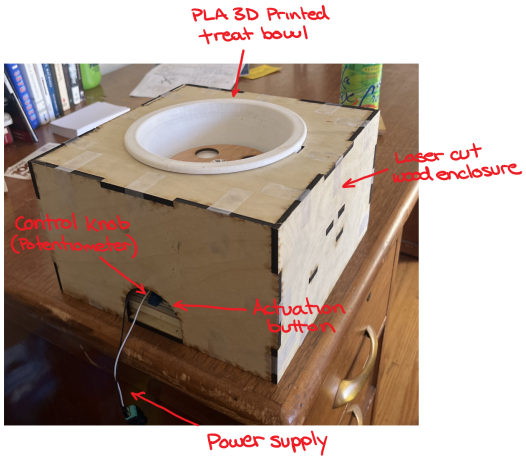


Figure 1: Full assembly

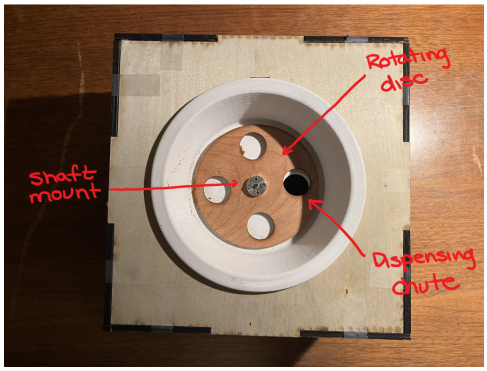


Figure 2: Dispensing mechanism

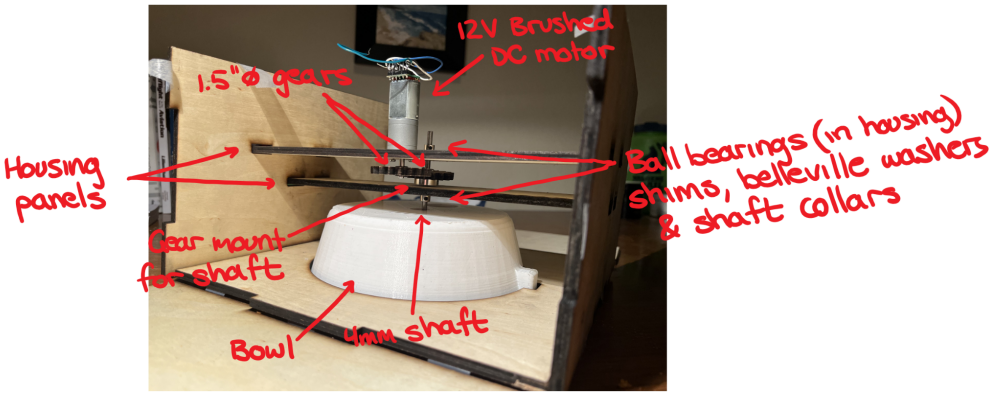


Figure 3: Transmission system

4. Function-critical decisions

When faced with designing a feeding mechanism that would allow for controlled distribution of treats, the first function-critical design decision was how to turn something which has one degree of freedom into something which distributes pet treats. Many of the initial design concepts lead to bunching and blocking of the distribution hole by the treats. An arm mechanism will push treats towards a hole, but there is no controlling if other treats will fall out of this hole when the arm is not covering it. The design which came as a solution to this problem was a horizontal plate at the base of a bowl with a hole that, when aligned with the distribution chute, allowed for treats to fall through its opening. This design solved the problem of free-falling treats and allowed for a controlled distribution of treats. Next, in order to allow for more frequent distributions with less movement by the transmission, more treat holders were added to the plate, to allow one 90º rotation to distribute one portion of treats. This method also ensures consistency — when the number of treats in the bowl gets low, there are enough holes to ensure that blank distributions will not occur.

4.1. Calculations:

Some calculations necessary to ensure the device will run on a 12V motor are performed on the rotational plate in the device. Since this component is the only moving part beyond the transmission gears, we focus our physical analysis on it.

1 cup of kibble ~ 0.2 kg

Radius of plate: $r_{plate} = 7cm$

Plate mass: $m_{plate} = \rho v = 0.034\ kg$

Torque outside plate: $\tau_0 = 2[r_{plate}(m_{plate} + m_{kibble})] = 1.434\ kg \cdot cm$

Torque inside plate: $\tau_i = \frac{d_z}{d_0}\frac{d_i}{d_z}\tau_0 = 1.434\ kg \cdot cm$

Stall torque: $\tau_{stall} = \frac{\tau_i}{0.6} = 7.17\ kg \cdot cm$

Therefore, a $16\ kg \cdot cm$ rotated motor, RPM does not need to be high for the device to run.

5. Circuit diagram and state transition diagram

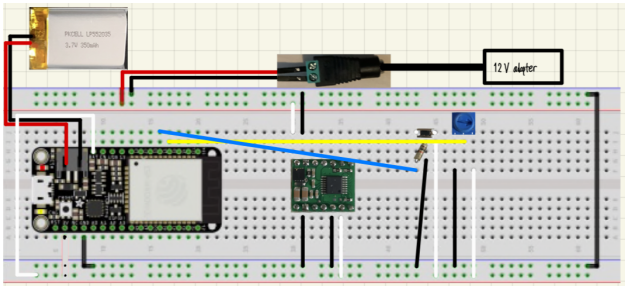


Fig **X**: Circuit diagram

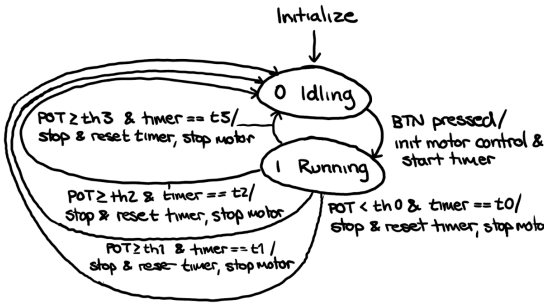


Fig. **Y**: State transition diagram

6. Final reflection

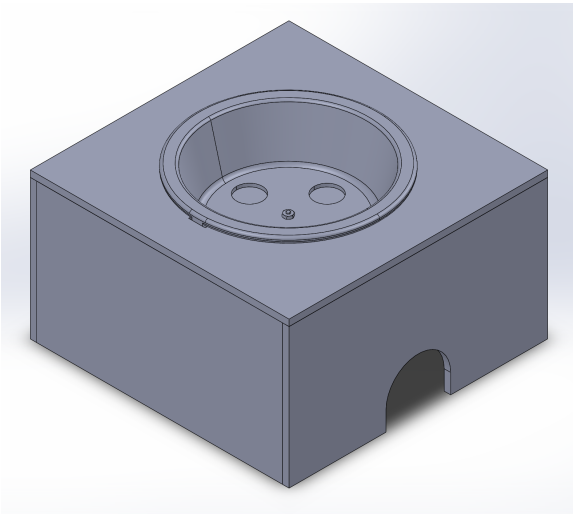
To future ME102B engineers, there is no such thing as a bad idea. When you workshop ideas, listen to each member of the group. If there are no ideas, make up nonsense because that openness will allow for the best and most feasible ideas to surface. When each member of the group has an open mind and no agenda, the project runs smoothest. Take time to meet and discuss openly how robust the project should be and what each member of the group's goals for the project and the class are. Something that worked well in our group was the openness to new innovation and project scope adjustments as the project progressed. Each one of us would say we wish we could have dedicated more time to the project and attempted to implement those initial features into the final product, but the goal of producing a complete and working device should take precedent. Overall, enjoy getting to work collaboratively on a project with other engineers and take pride in your final outcome, no matter what that may be.

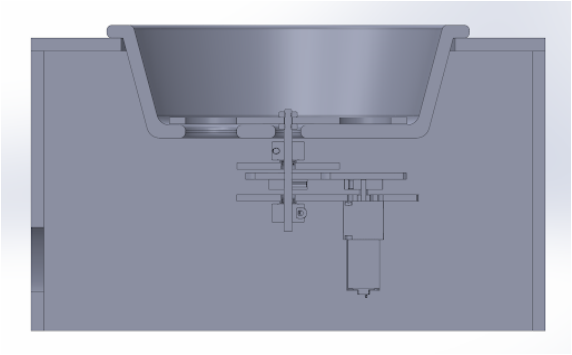
Appendices

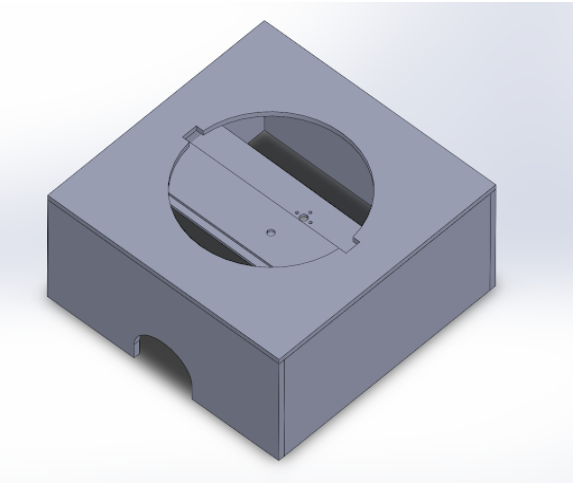
A. Bill of Materials

Part	Description	Quantity	Cost	Source
1	Ultimaker Filament: PLA	3	$ 6.00	Jacobs Hall
2	488:1 Metal Gearmotor 20Dx46L mm 12V CB with Extended Motor Shaft	1	$ 29.95	Pololu
3	Magnetic Encoder Pair Kit for 20D mm Metal Gearmotors, 20 CPR, 2.7-18V	1	$ 8.95	Pololu
4	Universal Aluminum Mounting Hub for 4mm Shaft, #4-40 Holes	2	$ 4.25	Pololu
5	Assembled Adafruit HUZZAH32 – ESP32 Feather Board - with Stacking Headers	1	-	Adafruit
6	Plywood - 1/4" x 18" x 30"	2	$ 6.25	Jacobs Hall
7	Swpeet 80Pcs 1K-500K Ohm Potentiometer Assortment Kit	1	-	Swpeet
8	Low-Carbon Steel Rod, 4 mm Diameter, 1 Foot Long	1	$ 2.42	McMaster-Carr
9	Flanged Ball Bearing	2	$ 9.53	McMaster-Carr
10	18-8 Stainless Steel Round Shim	2	$ 3.40	McMaster-Carr
11	Belleville Disc Spring	2	$ 1.39	McMaster-Carr
12	Machine Screw: #4-40, 5/16″ Length, Phillips	4	$ 0.35	Pololu
13	Set Collar 4.0mm Shaft Hole Diameter	2	$ 4.35	Aexit
14	40PCS L Bracket Corner Brace Sets	8	$ 1.12	SEANSDA
15	Steel Hex Nut, Zinc-Plated, 10-32 Thread Size	16	$ 0.14	McMaster-Carr
16	316 Stainless Steel Hex Drive Flat Head Screw, 82 Degree Countersink Angle, 10-32 Thread Size, 3/4" Long, packs of 10	16	$ 0.21	McMaster-Carr
17	#4-40 UNC Stainless Steel Machine Screws, 230PCS Flat Head Phillips Bolts Screws Nuts	1	$ 15.42	Amazon
18	DRV8874 Single Brushed DC Motor Driver Carrier	1	$ 9.95	Pololu
19	Cylewet 12Pcs 1A 250V AC 2 Pins SPST Momentary Mini Push Button Switch Normal Open	1	-	Cylewet

B. CAD







C. Code

```
#include <Arduino.h>
#define BIN_1 26
#define BIN_2 25
#define BTN 15
#define LED_PIN 13
#define POT 32 // declare potentiometer

//Setup variables -----------------------------------
const int freq = 5000;
const int pwmChannel = 0;
const int resolution = 8;
volatile bool buttonIsPressed = false;
int state = 1;
int MAX_PWM_VOLTAGE = 75;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;

int motor_PWM;

// POT properties
int readVal = 0;
int ledVal = 0;

//Setup timer for debounce
volatile bool debounceT = false; //check timer interrupt
hw_timer_t *timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

//Initialization timer for debounce ----------------------
void IRAM_ATTR onTime() {
  portENTER_CRITICAL_ISR(&timerMux);
  debounceT = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux);
}

//Initialization -----------------------------------
void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
  buttonIsPressed = 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, 50000, true);         // sets how many tics will you count to trigger the interrupt
  timerAlarmEnable(timer); // Enables timer
} // For debounce
```

```
long previousMillis = 0;
unsigned long curr;
long interval_1 = 2000;
long interval_2 = 2*interval_1;
long interval_3 = 3*interval_1;
long interval_4 = 4*interval_1;

extern volatile unsigned long timer0_millis;

void setup() {
  pinMode(BTN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  attachInterrupt(BTN, isr, RISING);

  Serial.begin(115200);

  ledcSetup(ledChannel_1, freq, resolution);
  ledcAttachPin(BIN_1, ledChannel_1);

  ledcSetup(ledChannel_2, freq, resolution);
  ledcAttachPin(BIN_2, ledChannel_2);

  TimerInterruptInit(); // Initiates timer interrupt for debounce
}
```

```
void loop() {
  delay(100);
  switch (state) {
    case 1:
      if (CheckForButtonPress() == true) {
        BtnResponse();
        led_on();
        motor_on();
        state = 2;
      }
      break;

    case 2:
      Serial.println(curr);
      timeOut();
      if (CheckForButtonPress() == true) {
        BtnResponse();
        led_off();
        motor_off();
        state = 1;
      }
      break;
  }
}

bool CheckForButtonPress() {
  if (buttonIsPressed == true && debounceT == true){
    return true;
  }
  else {
    return false;
  }
}

void led_on() {
  digitalWrite(LED_PIN, HIGH);
}

void led_off() {
  digitalWrite(LED_PIN, LOW);
}

void motor_on() {
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcWrite(ledChannel_1, MAX_PWM_VOLTAGE);

  ledcAttachPin(BIN_2, ledChannel_2);
  ledcWrite(ledChannel_2, LOW);
}

void motor_off() {
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcWrite(ledChannel_1, LOW);

  ledcAttachPin(BIN_2, ledChannel_2);
  ledcWrite(ledChannel_2, LOW);
}
```

```
// Service Funtions
void BtnResponse() {
  curr = millis();
  Serial.println(curr);

  readVal = analogRead(POT); // read pot value
  ledVal = map(readVal, 0 , 4095, 0, 255);

  Serial.println(ledVal);

  timerStart(timer);
  buttonIsPressed = false; // reset btn flag to false

  timerStop(timer);
  timerRestart(timer);
}

void timeOut() {
  Serial.println(curr);
  if(ledVal >= 195) { // pass 4 holes
    if(millis() - curr >= interval_4){
      led_off();
      motor_off();
      state = 1;
    }
  }
  else if(ledVal < 195 && ledVal >= 130) { // pass 3 holes
    if(millis() - curr >= interval_3){
      led_off();
      motor_off();
      state = 1;
    }
  }
  else if(ledVal < 130 && ledVal >= 65) { // pass 2 holes
    if(millis() - curr >= interval_2){
      led_off();
      motor_off();
      state = 1;
    }
  }
  else if(ledVal < 65) { // pass 1 holes
    if(millis() - curr >= interval_1){
      led_off();
      motor_off();
      state = 1;
    }
  }
}
```