Food Experimentation Device (Adaptation)

Bohórquez Lab 2016

Summary:

The Food Experimentation Device (FED) is a Arduino-based system designed for the precision feeding of mice. It is composed of an Arduino Pro, Adafruit Motor Shield, Adafruit Data Logging Shield, two photo-interrupter circuits, two motor circuits and an assortment of 3D printed parts. This adaptation of the FED is unique from its ancestor in that it can accommodate two mice at one time (with a set of pellet wells, motors and food silos). Future modifications will take advantage of this capability using a video feed or transponder system (coming soon).

Parts:

- (1) Arduino Pro: https://www.sparkfun.com/products/10915
- (1) Header pins for the Arduino Pro: https://www.sparkfun.com/products/10007
- (1) Header pins for the motor shield: https://www.sparkfun.com/products/11417
- (1) Header pins for the datalogging shield: https://www.sparkfun.com/products/11417
- (1) 7 segment display: https://www.sparkfun.com/products/11442
- (2) Photo Interrupter: https://www.sparkfun.com/products/9299
- (2) Photo Interrupter board: https://www.sparkfun.com/products/9322

- Adafruit
  - (1) Datalogging shield with on-board RTC: https://www.adafruit.com/products/1141
  - (1) SD card: https://www.adafruit.com/products/102
  - (2) Stepper motor: https://www.adafruit.com/products/858
  - (1) Motor shield *v2*: https://www.adafruit.com/product/1438
  - (1) LiPo boost circuit and recharger: https://www.adafruit.com/products/2465
  - (1) 6600 mAh LiPo: https://www.adafruit.com/products/353
- 3D printed parts (see supporting documents)
  - (1) Motor housing

- (2) Motor mounted pellet dispensing disk:
- (2) Food silo
- (2) Food silo lid
- (2) Pellet well
- (1) Back
- (1) Cover
- (1) Base

Construction Diagrams:

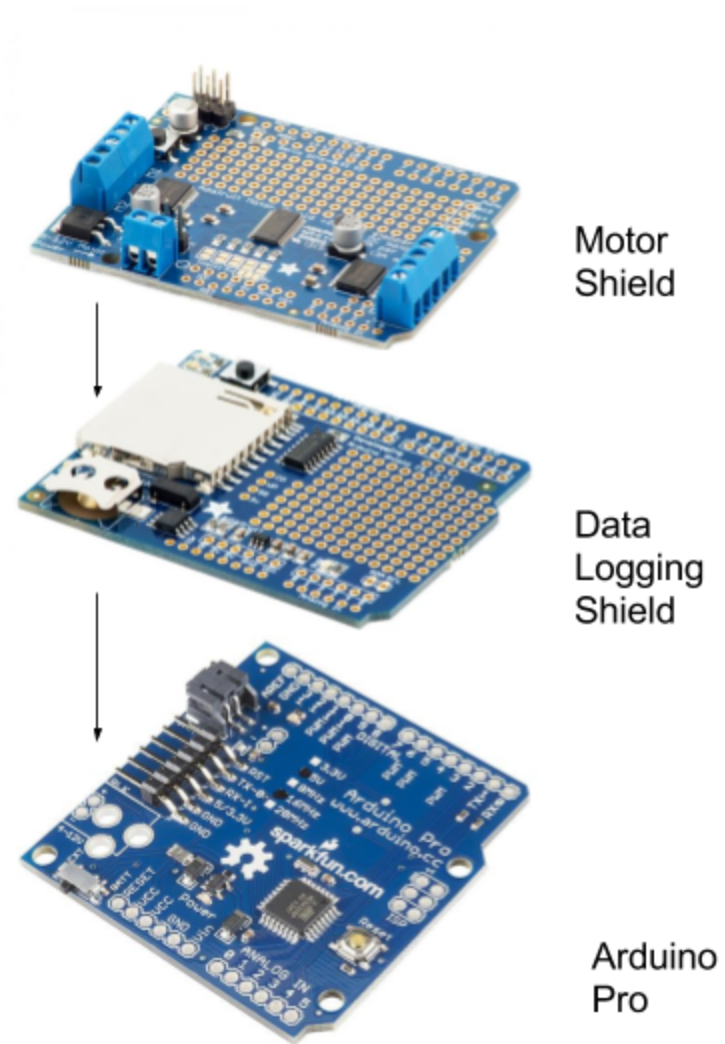


Figure 1: Stacking the boards

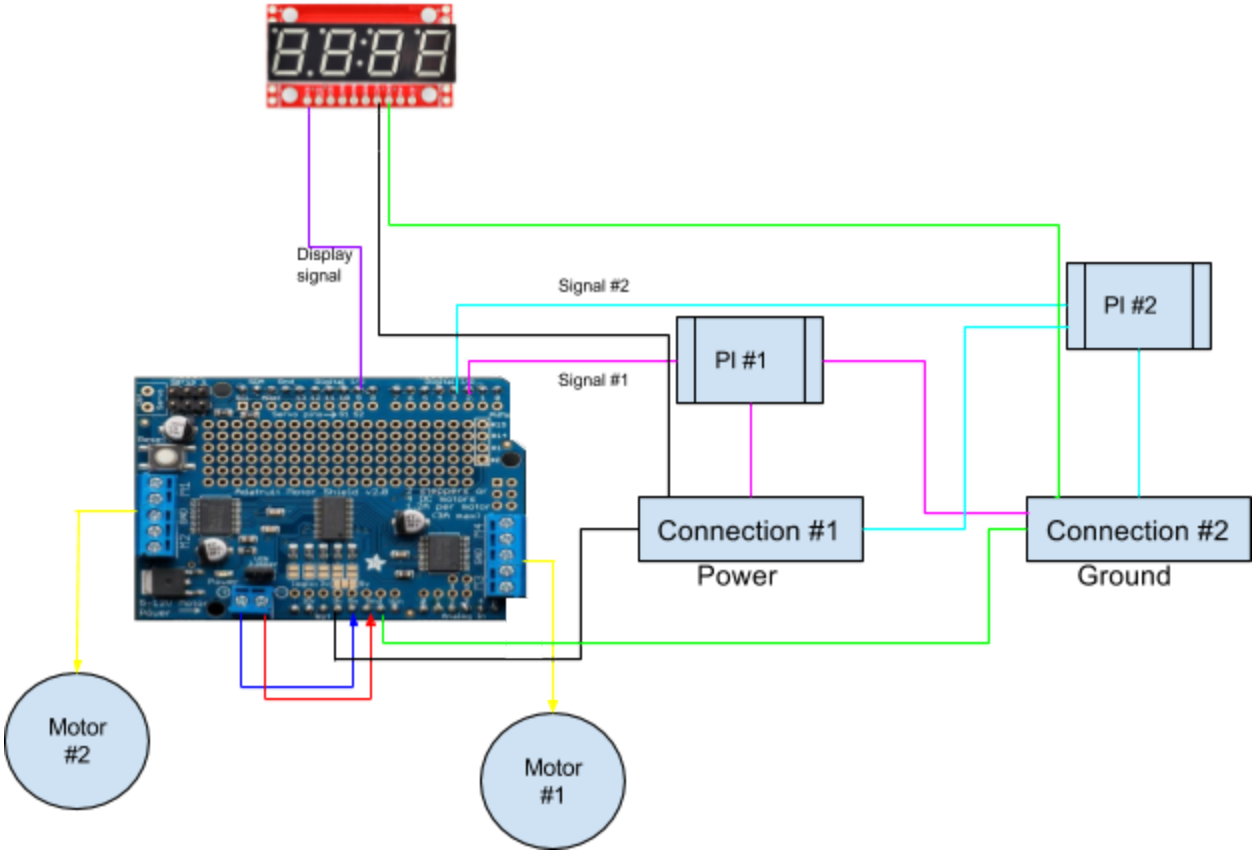


Figure 2: Outline of Integrated Circuitry

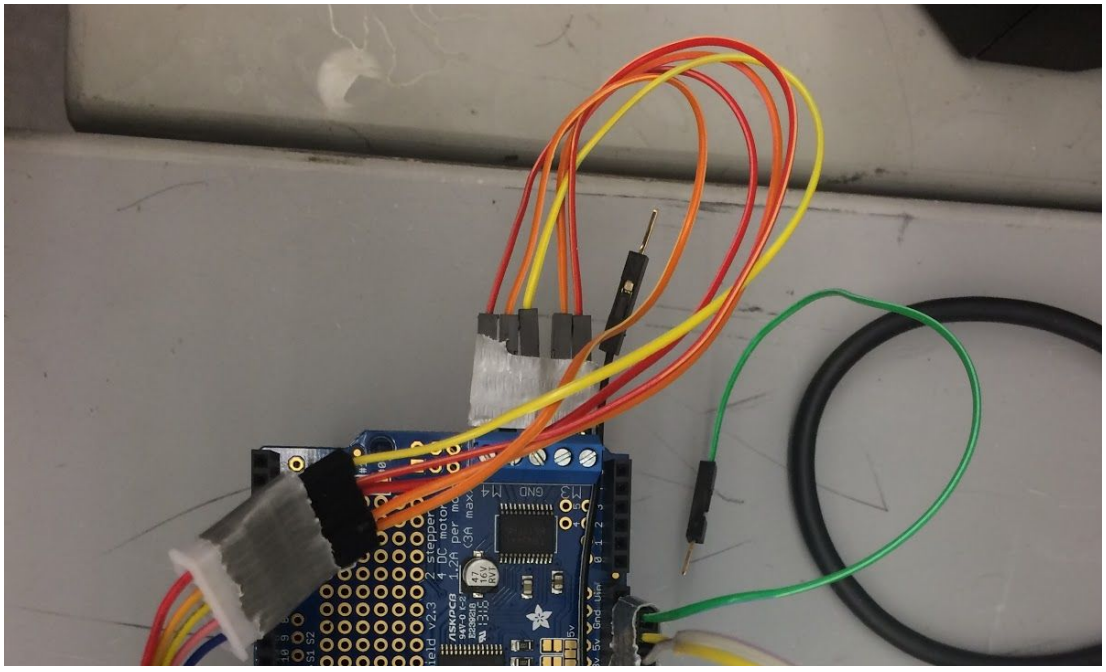


Figure 3: Motor Wiring

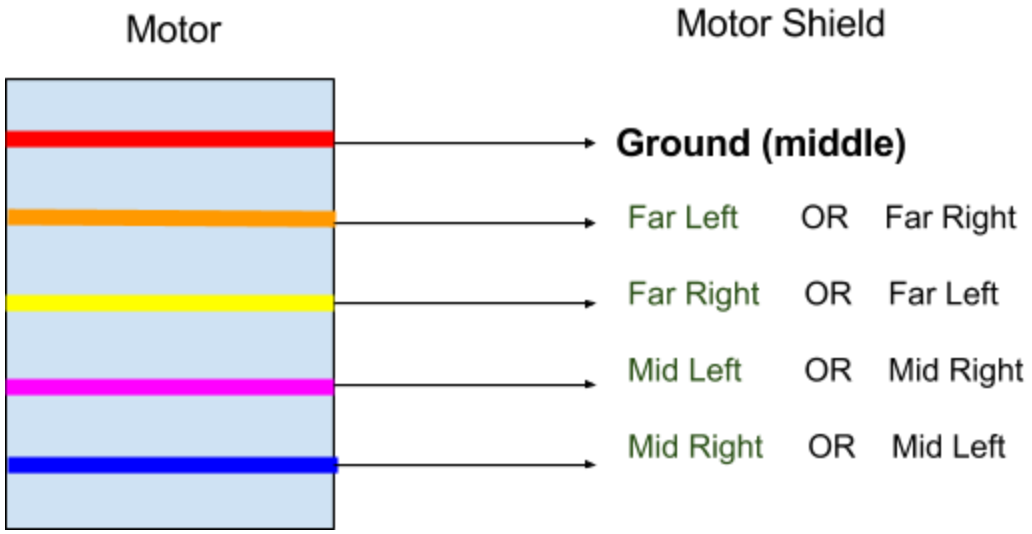
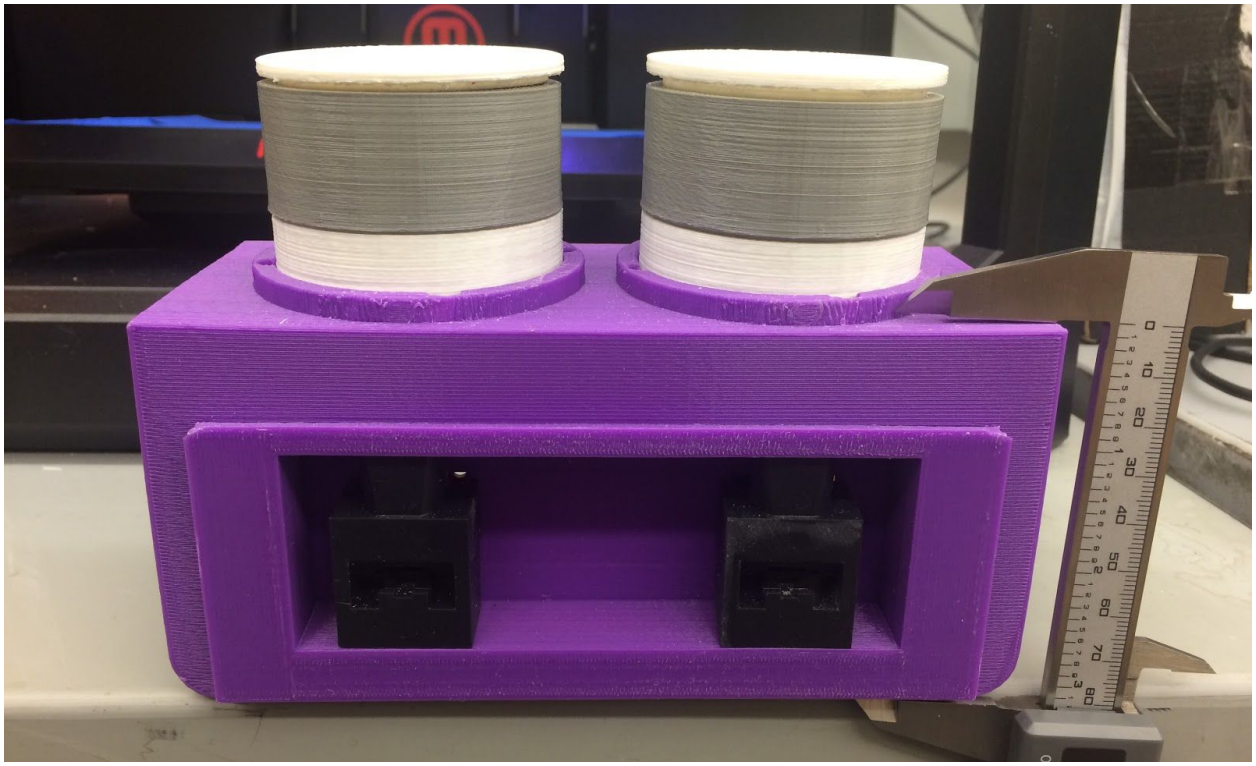


Figure 4: Motor Wiring Diagram

Finished Product:



Code:

(Largely derived from open source code)

```
#define PHOTO_INTERRUPTER_PIN_1 2
#define PHOTO_INTERRUPTER_PIN_2 3
const int PHOTO_INTERRUPTER_PINS[] = { PHOTO_INTERRUPTER_PIN_1,
PHOTO_INTERRUPTER_PIN_2 };

//defining serial display pin (pin 9)
#define DISPLAY_SERIAL_RX_PIN 255 //255=null
#define DISPLAY_SERIAL_TX_PIN 9
SoftwareSerial LEDserial = SoftwareSerial(DISPLAY_SERIAL_RX_PIN, DISPLAY_SERIAL_TX_PIN);

//reference definitions for time conversions
const long day2 = 86400000; // 86400000 milliseconds in a day
const long hour2 = 3600000; // 3600000 milliseconds in an hour
const long minute2 = 60000; // 60000 milliseconds in a minute
const long second2 =  1000; // 1000 milliseconds in a second

RTC_DS1307 RTC; //real time clock (on SD shield)

//setting up all the variables that will be used for couting pellets**
//        and storing data on IR photo-interrupter states
int PIStates[] = { 1, 1 };
int lastStates[] = { 1, 1 };
int lastTime[] = { 0, 0 };
int pelletCount[] = { 0, 0 };
bool pelletCounted[] = { false, false };
bool countMe[] = { false, false };

//setting up variables for dealing with the SD card
#define FILENAME "PelletData.csv"
const int CS_pin = 10;
SdFat SD;
File dataFile;

//setting up motors and motor variables**
const int MOTOR_STEPS_PER_REVOLUTION = 513;  //number of "steps" the motor takes in order
to make a full revolution.  Typically one more than a power of two
const int STEPS_TO_INCREMENT = 48; //steps it travels each time it turns
Adafruit_MotorShield gMotorShield = Adafruit_MotorShield();
```

```
Adafruit_StepperMotor *motor1 =
gMotorShield.getStepper(MOTOR_STEPS_PER_REVOLUTION,1);
Adafruit_StepperMotor *motor2 =
gMotorShield.getStepper(MOTOR_STEPS_PER_REVOLUTION,2);

//initialization, run when the program first starts
void setup() {
 //power saving stuff
 for (byte i=2; i <= 20; i++)
 { pinMode(i, INPUT_PULLUP); }
 ADCSRA = 0;  //disable ADC
 power_adc_disable(); //ADC converter
 power_timer1_disable(); //Timer 1
 power_timer2_disable(); //Timer 2

 //serial monitor initialization
 Serial.begin(9600);
 Serial.println(F("Starting up..."));

 //display initialization and setup
 LEDserial.begin(9600);
 pinMode(DISPLAY_SERIAL_TX_PIN, OUTPUT);
 setDisplayBrightness(64);
 updateDisplay();

 //SD card pin setup
 pinMode(CS_pin, OUTPUT); //CS pin
 pinMode(SS, OUTPUT);

 //motor shield stuff
 gMotorShield.begin();
 motor1->setSpeed(15); //setting speed of motor 1**
 motor2->setSpeed(15); //setting speed of motor 2**

 //SD card init stuff
 Wire.begin();
 RTC.begin();
 delay(250);

RTC.adjust(DateTime(__DATE__, __TIME__));  //<--used for resetting the clock if necessary**

 //checking if clock is running
 if (!RTC.isrunning()) {
```

```
  Serial.println(F("RTC is NOT running!"));
  RTC.adjust(DateTime(F(__DATE__), F(__TIME__)));
 }
 else {
  Serial.println(F("RTC init sucessful"));
  Serial.print(F("Current time: "));
  Serial.println(currentTime());
 }

 //checking if SD card is present
 if (!SD.begin(CS_pin)) {
  Serial.println(F("Card failed, or not present"));
  while (1); //stop
 }
 Serial.println(F("Card initialized."));

 //checking if SD card file saving works
 dataFile = SD.open(FILENAME, FILE_WRITE);
 if (!dataFile) {
  Serial.println(F("Error opening datalog.txt"));
  while (1);
 }
 else {
  dataFile.println(F("Time,Pellet #1 Count,Pellet #2 Count")); //header for SD card file
  dataFile.close();
  logData();
 }
 delay(500);

 //setup photointerrupter vals
 for (int i = 0; i < 2; i++)
 { PIStates[i] = digitalRead(PHOTO_INTERRUPTER_PINS[i]); lastStates[i] = PIStates[i]; }
}

//saving data to SD card:
void logData() {
 power_twi_enable();
 power_spi_enable();

 String time = currentTime();
 dataFile = SD.open(FILENAME, FILE_WRITE);
 if (dataFile) {
  Serial.println(F("File successfully written..."));
```

```
   Serial.println(time);

   //printing to SD card (each comma tells the excel doc to go to the next cell to the right)**
   dataFile.print(time);
   dataFile.print(",");
   dataFile.print(pelletCount[0]);
   dataFile.print(",");
   dataFile.println(pelletCount[1]);
   dataFile.close();
  }

  power_twi_disable();
  power_spi_disable();
}

//main loop - what runs the actual program
void loop() {
  bool a = updateState(0);
  bool b = updateState(1);
  if (!a  && !b)
  { enterSleep(); }
  delay(100);
}

//main function for updating each of the IR sensor values and responding**
bool updateState(int inputNum)
{
  PIStates[inputNum] = digitalRead(PHOTO_INTERRUPTER_PINS[inputNum]); //reading in IR
sensors

  if (PIStates[inputNum] == 1  & PIStates[inputNum] != lastStates[inputNum]) { //if currently
empty but wasn't empty last time
    //pellet taken
    if (!pelletCounted[inputNum])
    {
      countMe[inputNum] = true;
      pelletCounted[inputNum] = true;
    }
  }
  else if (PIStates[inputNum] == 1) { //if still empty, same as last time
    if (countMe[inputNum] && lastTime[inputNum] >= 8) {
      pelletCount[inputNum]++; //counts pellet
      Serial.println(F("Pellet taken"));
```

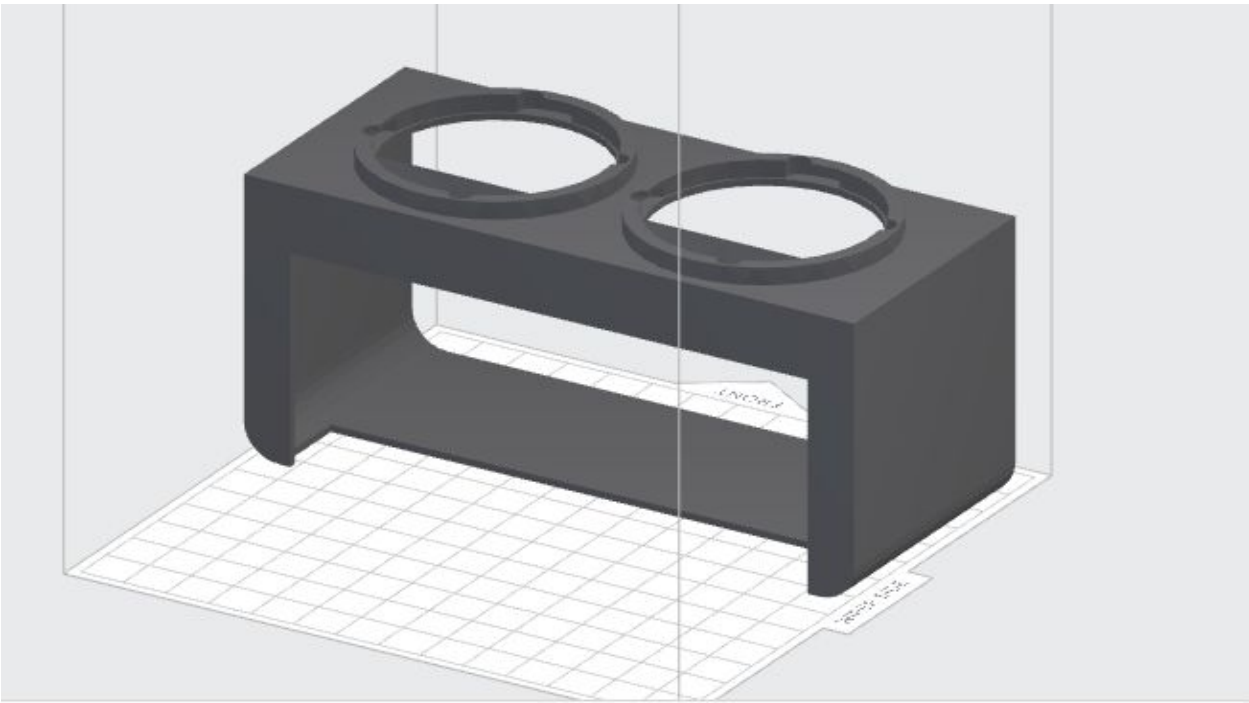
```
    Serial.print(F("Pellets of type #"));
    Serial.print(String(inputNum + 1));
    Serial.print(F(" taken: "));
    Serial.println(String(pelletCount[inputNum]));

    logData();
    updateDisplay();
    lastTime[inputNum] = 0;
    countMe[inputNum] = false;
   }
   //need to replace pellet
   if (lastTime[inputNum] >= 8) {
    Serial.print(F("Replacing pellet #"));
    Serial.print(String(inputNum + 1));
    Serial.println(F("..."));
    moveMotor(inputNum);
    lastTime[inputNum] = 6;
    pelletCounted[inputNum] = false;
    delay(200);
   }
   else
   { delay(500); lastTime[inputNum] += 1; }
  }

  else if (PIStates[inputNum] == 0 & PIStates[inputNum] != lastStates[inputNum]) { //if currently
has pellet but didn't have pellet last time
   //pellet just replaced
   Serial.print(F("Pellet #"));
   Serial.print(String(inputNum + 1));
   Serial.println(F(" replaced"));
   lastTime[inputNum] = 0;
  }

  else  { //if PIStates == 0 //if still has pellet, same as last time
   //pellet still there, do nothing
   lastStates[inputNum] = PIStates[inputNum];
   lastTime[inputNum] = 0;
   return false;
  }
  lastStates[inputNum] = PIStates[inputNum];
  return true;
}
```

```
//code for moving motors
void moveMotor(int motorNum)
{
   const int backMotion = 6; //fraction of the motion for which to move backwards (6 -> 1/6th)**
   power_twi_enable();
   if (motorNum == 0) //for moving first motor
   {
    motor1->step(STEPS_TO_INCREMENT/backMotion,BACKWARD,DOUBLE);
    motor1->step(STEPS_TO_INCREMENT,FORWARD,DOUBLE);
    motor1->release();
   }
   else if (motorNum == 1) //for moving second motor
   {
    motor2->step(STEPS_TO_INCREMENT/backMotion,BACKWARD,DOUBLE);
    motor2->step(STEPS_TO_INCREMENT,FORWARD,DOUBLE);
    motor2->release();
   }
   power_twi_disable();
}


//-------------------------anything after this probably shouldn't be changed----------------------

//code for entering sleep mode
void enterSleep()
{
 Serial.println(F("Going to sleep.")); delay(50);
 power_usart0_disable(); //Serial (USART)
 sleep_enable();

 attachInterrupt(digitalPinToInterrupt(PHOTO_INTERRUPTER_PIN_1), pinInterrupt, RISING);
 attachInterrupt(digitalPinToInterrupt(PHOTO_INTERRUPTER_PIN_2), pinInterrupt, RISING);
 delay(100);

 set_sleep_mode(SLEEP_MODE_PWR_DOWN);
 cli();
 sleep_bod_disable();
 sei();
 sleep_cpu();
 sleep_disable();
}

//code for what to do when leaving sleep mode, run when the interrupt is triggered
```

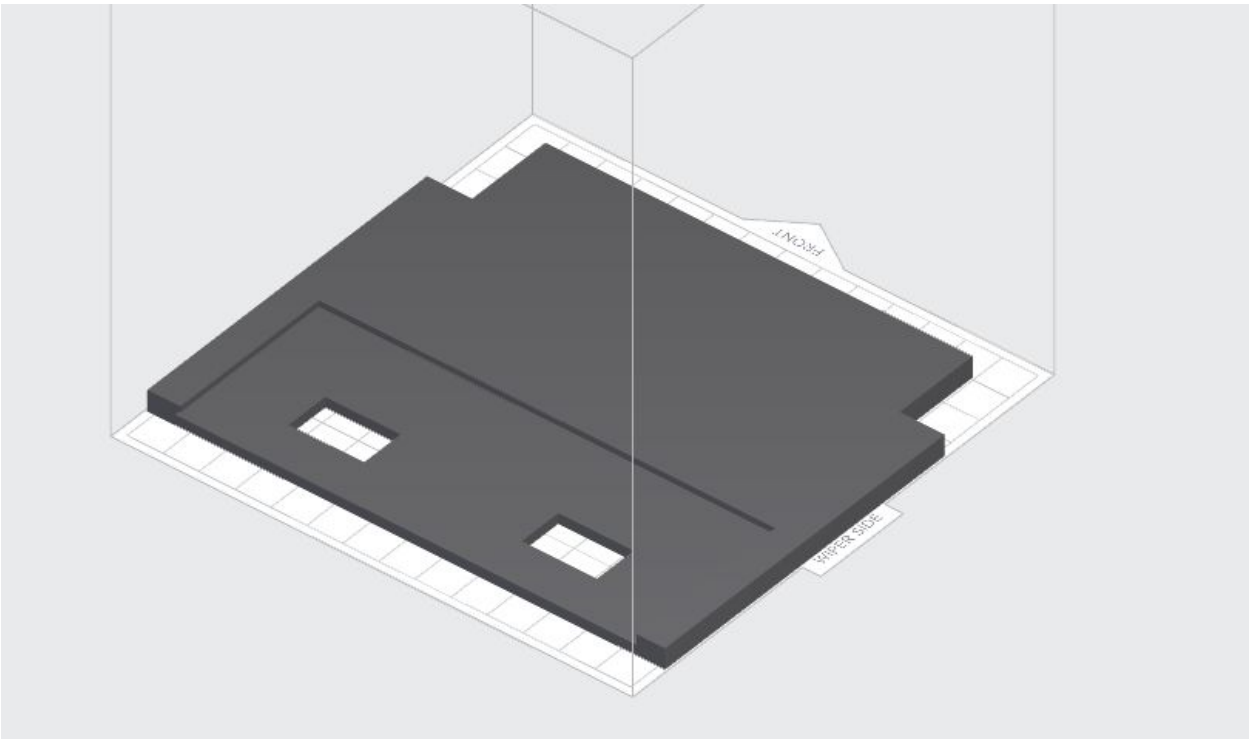
```
void pinInterrupt(void)
{
 detachInterrupt(digitalPinToInterrupt(PHOTO_INTERRUPTER_PIN_1));
 detachInterrupt(digitalPinToInterrupt(PHOTO_INTERRUPTER_PIN_2));
 sleep_disable(); //disabling sleep
 power_usart0_enable(); //re-enabling serial
 Serial.println(F("Exiting sleep."));
 delay(300);
}

//updating the LCD monitor display
void updateDisplay()
{
 clearDisplay();
 delay(2);
 if (pelletCount[0] % 100 < 10)
 { LEDserial.write(0x79); LEDserial.write(1); }
 LEDserial.print(String(pelletCount[0]));
 if (pelletCount[1] % 100 < 10)
 { LEDserial.write(0x79); LEDserial.write(3); }
 LEDserial.print(String(pelletCount[1]));
}

//next few are all methods for editing display
void clearDisplay()
{ LEDserial.write(0x76); }

//brightness: 0 to 255
void setDisplayBrightness(byte value)
{
 LEDserial.write(0x7A);
 LEDserial.write(value);
}

void setDisplayValues(String value)
{ LEDserial.print(value); }

//returns time from clock
String currentTime()
{
 DateTime datetime = RTC.now();
 String year = String(datetime.year(), DEC);
 String month = String(datetime.month(), DEC);
```

```
 String day  = String(datetime.day(),  DEC);
 String hour  = String(datetime.hour(),  DEC);
 String minute = makeDigit(datetime.minute());
 String second = makeDigit(datetime.second());
 return (month + "/" + day + " " + hour + ":" + minute + ":" + second);
}

//formats numbers from clock
String makeDigit(int i)
{
 if (i < 10)
 { return ("0" + String(i, DEC)); }
 else
 { return String(i, DEC); }
}
```

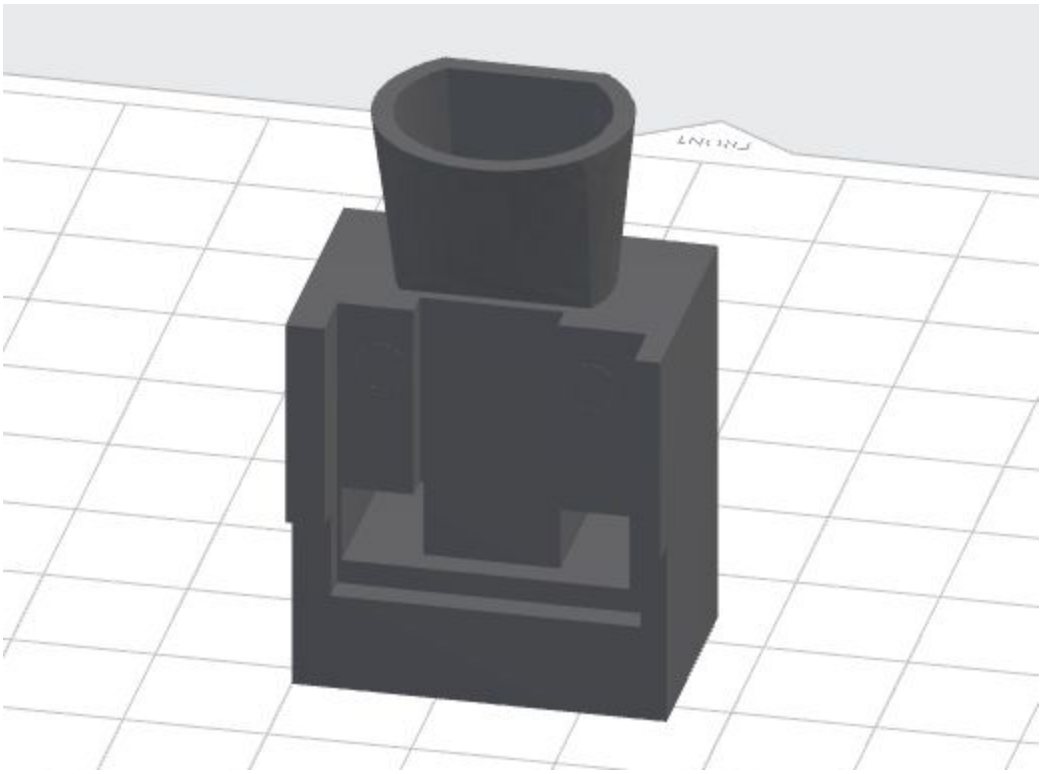
3D Printed Parts:



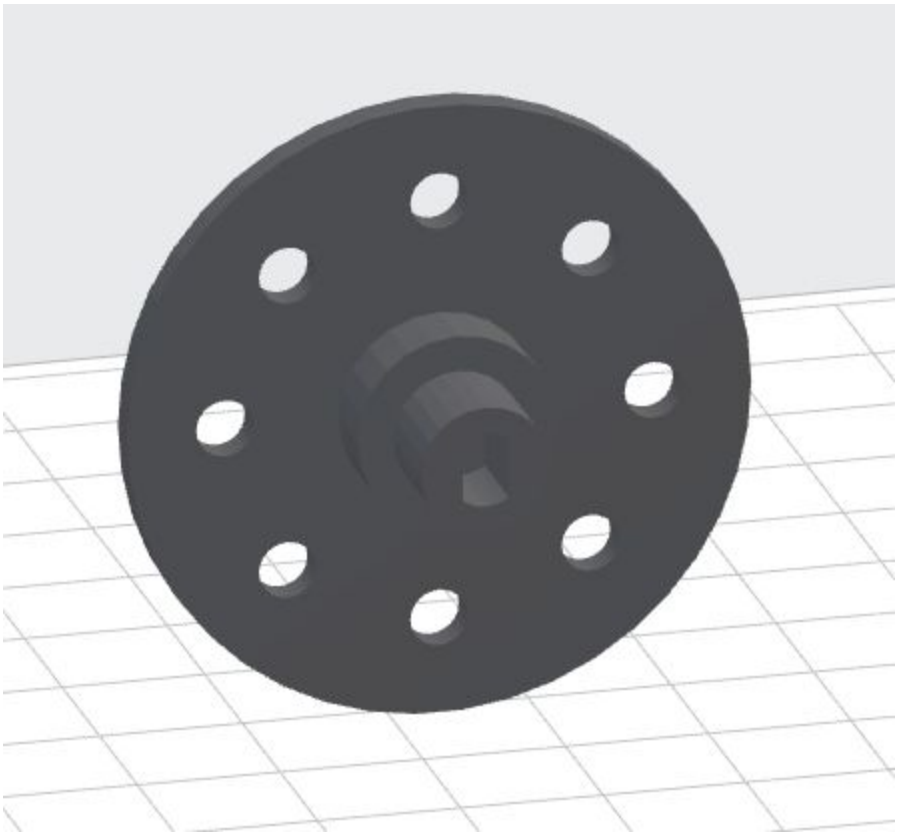
Part 1: Pellet Dispenser Casing



Part 2: Front Cover



Part 3: Pellet Funnel



Part 4: Motor Attachment

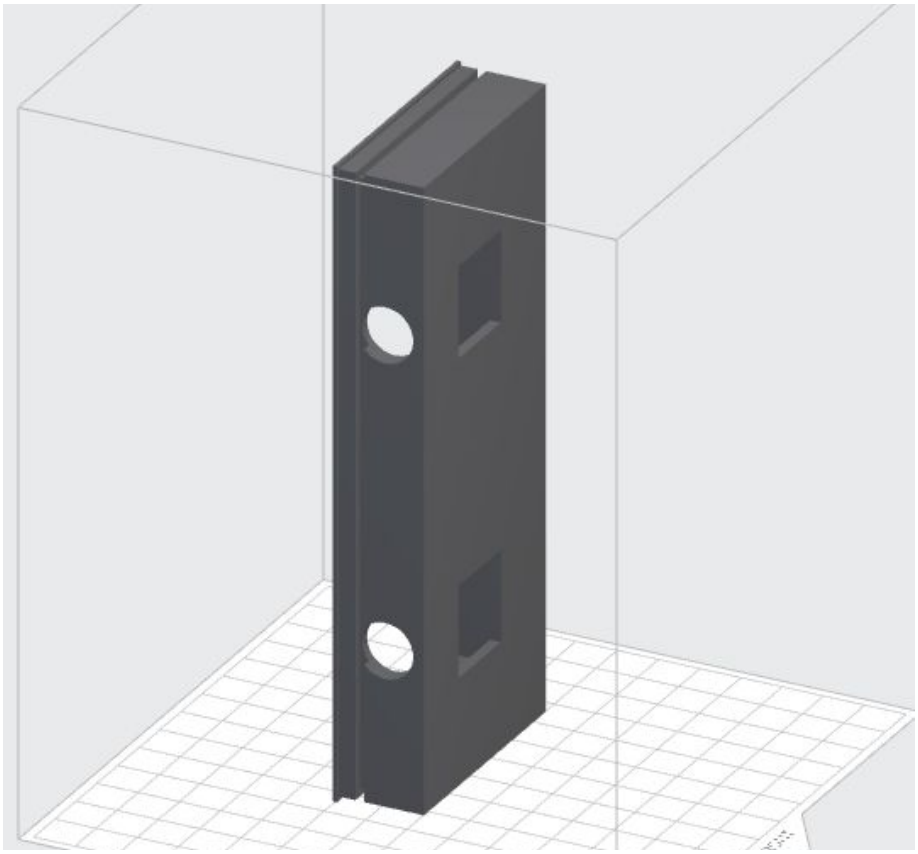


Figure 5: Front Shelf

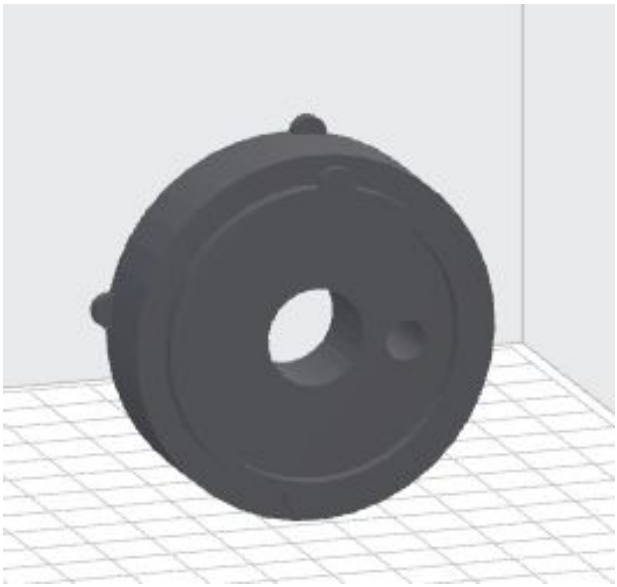


Figure 6: Food Well Mount

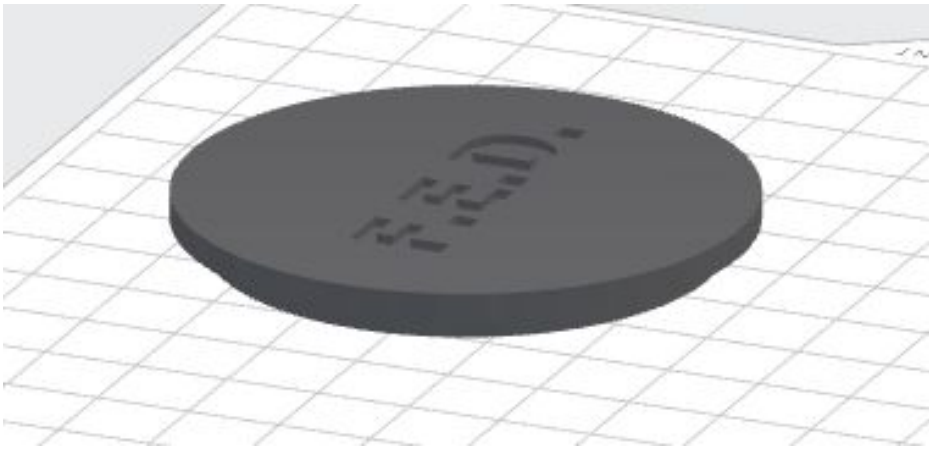


Figure 7: FED Lid

(Missing Resource) Figure 8: Food Silo
Silo between FED Lid and Food Well Mount

(Missing Resource) Figure 9: Back Cover
Back cover, hides exposed circuitry