Automatic Cheese Grater: Final Project Report

G1: Steph Akakabota, Maylee Tan, Kyler Yeum, Xavier Macalalad

May 14, 2025

1 Opportunity

Preparing fresh grated cheese at home is time-consuming, and commercial automatic graters are prohibitively expensive ($200+). Our device offers an affordable, push-button solution for fast, high-quality cheese grating, making fresh meals more accessible to everyday users.

2 High-Level Strategy

Our automatic cheese grater streamlines cheese preparation by integrating two motorized systems:

- **Lead Screw Mechanism:** A 24V motor with encoder feedback advances a lead screw to apply controlled pressure to the cheese block.
- **Rotary Grater:** The original hand crank is replaced with a 24 V motor, which automates the rotation of the grating drum.

Operation: User loads cheese → button press → screw advances (30k encoder counts) → grater spins → 5s hold → screw retracts.

3 Integrated Physical Device

Our device integrates a lead screw mechanism for cheese compression and a rotary grater for shredding. Key actuators, sensors, and moving parts are labeled in Fig. 1a. The compact design enables efficient operation and easy access for maintenance.

4 Function-Critical Decisions

4.1 Load Analysis

Material Basis: Used non-fat mozzarella shear strength ($\tau = 140\,\text{kPa}$) as a conservative estimate for cheddar (no published data), with 2× safety factor:

$$\tau_{\text{design}} = 2 \times 140 = 280\,\text{kPa}$$

Contact Area: Accounting for 50% blade spacing reduction in grating surface:

$$A_{\text{eff}} = 0.5 \times (0.025\,\text{m} \times 0.050\,\text{m}) = 0.000\,625\,\text{m}^2$$

$$F = \tau A = 280 \times 10^3 \times 0.000625 = \boxed{175\,\text{N}}$$

4.2 Lead-Screw Torque

For Tr8x8 screw ($l = 8\,\text{mm}$) with $\eta = 0.30$ efficiency:

$$T = \frac{Fl}{2\pi\eta} = \frac{175 \times 0.008}{2\pi \times 0.30} \approx \boxed{0.74\,\text{N m}}$$

4.3 Lead Screw and Encoder Calculation

A Tr8x8 lead screw (8 mm/rev) with a 64 CPR encoder and 150:1 gearbox yields:

$$\text{Counts per rev} = 64 \times 150 = 9600$$

$$\text{Counts per mm} = \frac{9600}{8} = 1200$$

Thus, moving 25 mm requires $25 \times 1200 = 30{,}000$ encoder counts.

4.4 Motor Torque-Speed Analysis

The torque-speed relationship for our 24V Pololu gearmotor is:

$$T = \frac{k}{R}(V - k\omega) \quad \begin{aligned} &T = \text{torque (N·m)},\\ &V = 24\,\text{V},\\ &\omega = 7.13\,\text{rad/s (68 RPM)},\\ &k = 0.207\,\text{N·m/A},\\ &R = 8.89\,\Omega. \end{aligned}$$

Key operating points:

- **No-load:** 68 RPM (7.13 rad/s) @ 0.071A
- **Stall:** 0.560 N·m @ 2.7A

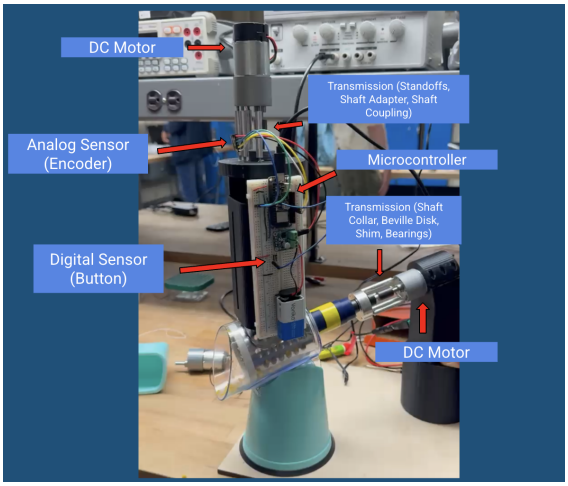
$$R = \frac{24}{2.7} \approx \boxed{8.89\,\Omega},$$
$$k = \frac{0.560}{2.7} \approx \boxed{0.207\,\text{N·m/A}}$$

Safe operation: While rated for 24V, we operate at 9 V (37.5% of max voltage) to limit speed to ∼25 RPM and prevent gearbox overload.

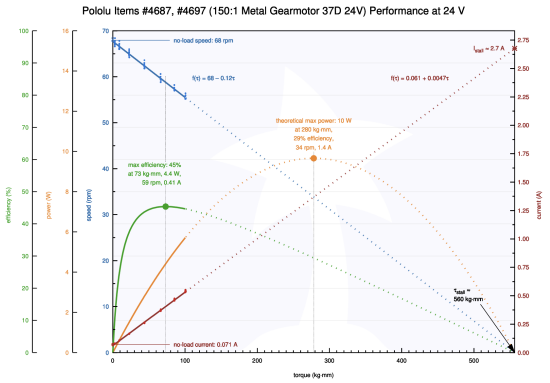
5 Reflection for Future Students

Defining our state machine early and verifying motor requirements with quick calculations made both hardware selection and software development much smoother. However, we underestimated the time and difficulty of precision machining tasks such as tapping aluminum standoffs, so we recommend designing to facilitate fabrication and ordering backup components ahead of time.

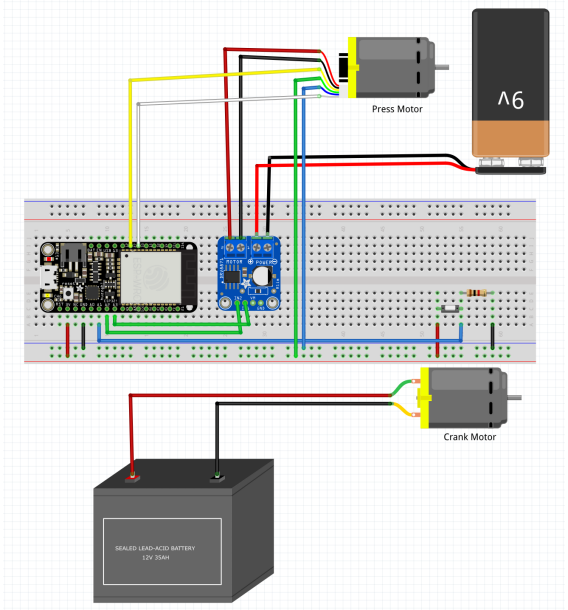
6 Figures



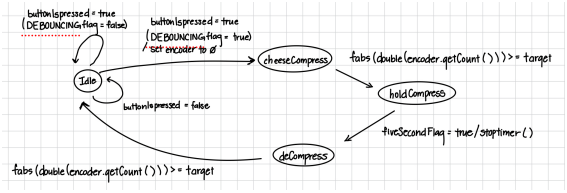
(a) Labeled diagram of the fully assembled cheese grater.



(b) Torque-Speed performance graph.



(c) Updated circuit diagram.



(d) State transition diagram.

Figure 1: Overview of system: (a) labeled assembly, (b) torque-speed graph, (c) circuit diagram, (d) state transition diagram.

7 Appendix A: Bill of Materials

Component	Description	Qty	Cost ($)
Electromechanical Components			
Pololu Gearmotor	150:1 Metal Gearmotor with Encoder	2	114.55
DRV8871 Motor Driver	Adafruit breakout board	2	16.54
Lab Motor	DC motor from lab kit	1	0.00
ESP32	Microcontroller	1	0.00
Mechanical Hardware			
X Home Cheese Grater	For grating drum and base	1	27.55
Tr8×8 Lead Screw	200mm with brass nut	1	13.88
Aluminum Rod	1/4" × 3 ft solid round rod	1	6.03
M3 × 18mm screws	Mounting hardware	33	16.94
McMaster-Carr Components			
Step-Down Shaft Adapter	3/8" to 1/4" diameter	1	46.68
Set Screw Shaft Coupling	1/4" diameter	2	25.10
Shaft Collar	1/4" clamping	2	11.34
Belleville Disc Springs	0.319" ID, pack of 10	1	5.00
Stainless Steel Ring Shim	0.01" thick, 1/4" ID	2	17.26
Stainless Steel Ball Bearing	Flanged, R4 trade number	2	12.84
Other Components			
Ultrasonic Sensor	RCWL distance measurement	1	0.00
PLA Filament	3D printing material	-	0.00
12V Power Adapter	DC power supply	1	24.68
	Total Project Cost		**377.80**

Appendix B: CAD Images

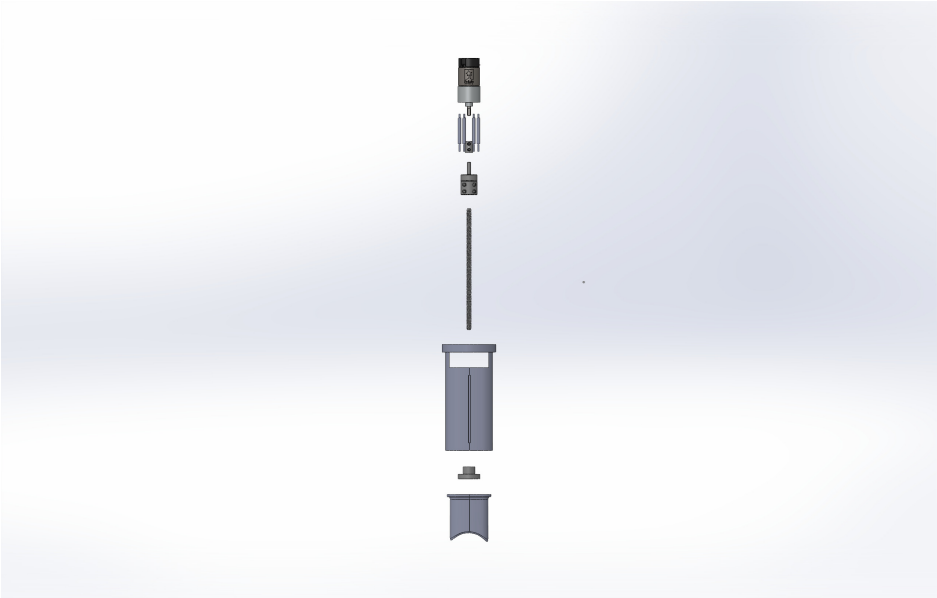


Figure 2: Exploded view of the lead screw assembly.

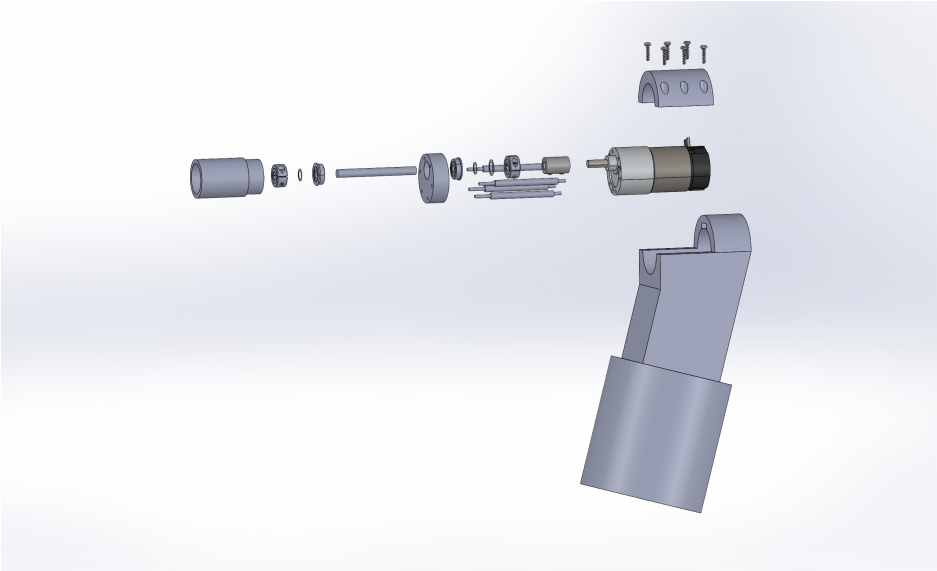


Figure 3: Exploded view of the crank assembly.

Appendix C: Complete Code

```
// updated code:
// - Replaced delay(5000) in holdCompress with a 5-second timer interrupt
// - Replaced delay(1) in driveMotorForward and driveMotorBackward with a 1-second timer interr
// - Added button debouncing using timer interrupts

#include <ESP32Encoder.h>
#include <Adafruit_INA219.h>

ESP32Encoder encoder;
Adafruit_INA219 ina219;

//  Pin assignments
#define MOTOR1_IN1    26
#define MOTOR1_IN2    25
#define ENCODER1_IN1  27
#define ENCODER1_IN2  33
#define BTN           34
#define LED_PIN       13

//  Motion constants
#define COUNTS_PER_REV  9600.0     // encoder counts per full screw rev
#define LEAD_MM         8.0        // screw travel (mm) per rev
#define COUNTS_PER_MM   (COUNTS_PER_REV / LEAD_MM)

//  State-machine
enum State {
  idle,
  cheeseCompress,
  holdCompress,
  deCompress
};

// Timer Setup
volatile bool buttonIsPressed = false;
volatile bool DEBOUNCINGflag = false;


hw_timer_t* motorTimer = NULL;
portMUX_TYPE motorTimerMux = portMUX_INITIALIZER_UNLOCKED;
volatile bool oneSecondFlag = false;
volatile int secondsElapsed = 0;
volatile bool fiveSecondFlag = false;


hw_timer_t* debounceTimer = NULL;
portMUX_TYPE buttonMux = portMUX_INITIALIZER_UNLOCKED;
```

```
#define DEBOUNCE_TIME 50 // 50ms debounce time

//  ISR
void IRAM_ATTR btnIsr() {
  portENTER_CRITICAL_ISR(&buttonMux);
  buttonIsPressed = true;
  portEXIT_CRITICAL_ISR(&buttonMux);
  portENTER_CRITICAL_ISR(&motorTimerMux);
  DEBOUNCINGflag = false;
  portEXIT_CRITICAL_ISR(&motorTimerMux);
  timerStart(debounceTimer);
}

void IRAM_ATTR onDebounceTimer() {
  portENTER_CRITICAL_ISR(&motorTimerMux);
  DEBOUNCINGflag = true;
  portEXIT_CRITICAL_ISR(&motorTimerMux);
}

void IRAM_ATTR onTimer() {
  portENTER_CRITICAL_ISR(&motorTimerMux);
  oneSecondFlag = true;
  secondsElapsed++;
  if (secondsElapsed >= 5) {
    fiveSecondFlag = true;
  }
  portEXIT_CRITICAL_ISR(&motorTimerMux);
}

void startTimer() {
  secondsElapsed = 0;
  fiveSecondFlag = false;
  if (motorTimer) {
    timerEnd(motorTimer);
  }
  motorTimer = timerBegin(0, 80, true); // 80MHz/80 = 1MHz
  timerAttachInterrupt(motorTimer, &onTimer, true);
  timerAlarmWrite(motorTimer, 1000000, true); // 1 second
  timerAlarmEnable(motorTimer);
}

void stopTimer() {
  if (motorTimer) {
    timerEnd(motorTimer);
    motorTimer = NULL;
  }
  oneSecondFlag = false;
  fiveSecondFlag = false;
```

```
  secondsElapsed = 0;
}

//  Helper functions
void stopMotor() {
  analogWrite(MOTOR1_IN1, 0);
  analogWrite(MOTOR1_IN2, 0);
}

void driveMotorForward(int mm) {
  encoder.setCount(0);
  double target = mm * COUNTS_PER_MM;

  digitalWrite(MOTOR1_IN2, LOW);
  analogWrite (MOTOR1_IN1, 128);

  startTimer();
  while (fabs(double(encoder.getCount())) < target) {
    if (oneSecondFlag) {
      oneSecondFlag = false;

    }
  }
  stopTimer();
  stopMotor();
}

void driveMotorBackward(int mm) {
  encoder.setCount(0);
  double target = mm * COUNTS_PER_MM;

  digitalWrite(MOTOR1_IN1, LOW);
  analogWrite (MOTOR1_IN2, 128);

  startTimer();
  while (fabs(double(encoder.getCount())) < target) {
    if (oneSecondFlag) {
      oneSecondFlag = false;

    }
  }
  stopTimer();
  stopMotor();
}

bool CheckForButtonPress() {
  if (!buttonIsPressed) return false;
```

```
  buttonIsPressed = false;

  Serial.println("Button pressed!");
  return true;
}

//  Arduino setup / loop
void setup() {
  Serial.begin(115200);
  Serial.println("\n== DRV8871 Lead-screw Demo ==");

  // Pin modes
  pinMode(MOTOR1_IN1, OUTPUT);
  pinMode(MOTOR1_IN2, OUTPUT);
  pinMode(LED_PIN,   OUTPUT);

  pinMode(BTN, INPUT_PULLUP);

  // Setup debounce timer
  debounceTimer = timerBegin(1, 80, true);
  timerAttachInterrupt(debounceTimer, &onDebounceTimer, true);
  timerAlarmWrite(debounceTimer, DEBOUNCE_TIME * 1000, false);  // One-shot timer

  // Attach button interrupt
  attachInterrupt(BTN, btnIsr, FALLING);

  pinMode(ENCODER1_IN1, INPUT);
  pinMode(ENCODER1_IN2, INPUT);

  // Encoder
  ESP32Encoder::useInternalWeakPullResistors = puType::up;
  encoder.attachHalfQuad(ENCODER1_IN1, ENCODER1_IN2);
  encoder.setCount(0);

  Serial.println("Ready.");
}
// EVENT DRIVEN PROGRAM
int state = idle;

void loop() {
  switch (state) {
    case idle:
      Serial.println("Idle State.");
      stopMotor();
      if (CheckForButtonPress()) {
        state = cheeseCompress;
      }
```

```
      break;

    case cheeseCompress:
      Serial.println("Compressing 25mm");
      driveMotorForward(25);

      state = holdCompress;
      break;

    case holdCompress:
      Serial.println("Holding 5 s");
      startTimer();
      while (!fiveSecondFlag) {
        delay(1);
      }
      stopTimer();
      state = deCompress;
      break;

    case deCompress:
      Serial.println("Retracting 25 mm");
      driveMotorBackward(25);
      state = idle;
      break;
  }
}
```