Automated Pour-over Coffee Machine

Evan Borzilleri

December 2020

1 Product Description

Like most college students, I love coffee. For the past two years, I have used the George Howell pour over method[1] to brew my coffee every morning. In short, this brewing method consists of pouring water at 368.7K over the grounds for 15 seconds in a circular or spiral motion, then letting it rest for another 15 seconds. This process continues for three minutes, with an additional 30 seconds of rest at the end for the coffee to fully draw down into the brewing carafe. However, since I typically make coffee in the morning, I am not fully alert and constantly get distracted, causing me to lose track of where I am in the brewing cycle. The Howell method is very precise, and any deviations can cause the brew to taste terrible (overextraction or underextraction). Therefore, I wanted to make a device that automated the pour over process to ensure I could have a quality cup of coffee every morning. Automated pour over machines exist, but cost hundreds of dollars. My design incorporates parts commonly found in the homes of engineers to create a machine that achieves similar result.

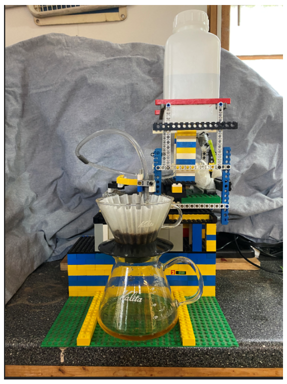

Figure 1: The automated pour-over machine

[1] `https://www.georgehowellcoffee.com/brew-guide/kalita-185/`

2 Materials Manufacturing

Due to the COVID-19 pandemic, my manufacturing and procurement abilities were limited. I don't have access to any rapid protyping machines, such 3D printers, laser cutters. I also live on an island, so shipping currently takes several weeks. Since I am back at my childhood home for the rest of the semester, I chose Legos as the main prototyping material. Using Legos allows me to quickly create and iterate my designs, while also providing sufficient structural support. In addition, Legos have tight tolerances, so all mechanisms can interface and function smoothly. Other electrical and mechanical components were sourced from the provided lab kit and my personal collection of parts.

3 Electro-mechanical Details

3.1 Brewing Arm

The brewing arm, as shown in figure 2, consists of a simple four-bar linkage attached to a continuous servo motor (Parallax futaba[2]) via a shaft coupler I manufactured. When turned on, the mechanism moves the water tube in a circular motion over the grounds, brewing the coffee. The speed of the arm can be controlled a potentiometer on the breadboard. It is not meant to be changed often, and is therefore located inside the machine.

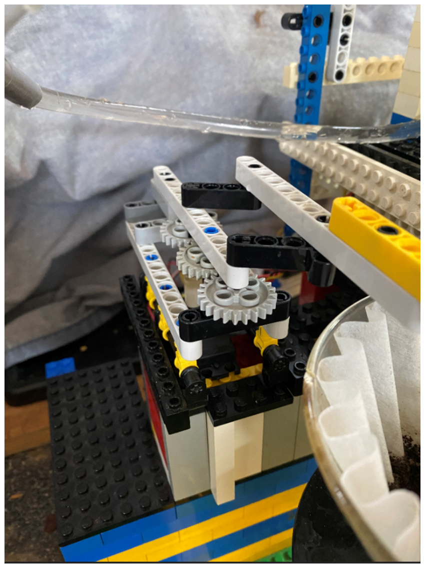

Figure 2: The brewing arm mechanism

[2]https://www.parallax.com/product/parallax-continuous-rotation-servo/

3.2 Valve Control

A high torque, metal geared servo motor (HS-645MG[3]) was used to open and close the water tank valve. The itself valve is a simple lever valve with a barb fitting on the end, originally from an old bandsaw coolant system. Ideally, I would use a solenoid valve to control the flow of water, but I did not have one on hand and could not find one that would ship in time. Interfacing the servo with the valve handle was the most difficult part of the design process. The handle requires a decent amount of torque, has no obvious attachment points, and is not parallel to valve housing, but instead 20 degrees off the plane, as shown in figure 4. My solution was to glue a Lego piece to the handle then use a strut with ball joints to connect them with another lever arm. The ball joints on the strut to allow it connect the lever arm and the valve handle, even when they are not planar. The lever is then rotated back forth using a gear-eqsue connector, that then interfaces with the servo shaft via a coupler. The angular values for opening and closing valve were determined experimentally.

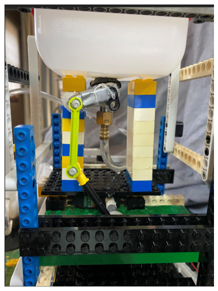

Figure 3: The valve control mechanism for the brewing machine

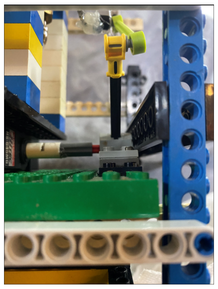

Figure 4: A side view of the valve actuation mechanism

[3]`https://hitecrcd.com/products/servos/sport-servos/analog-sport-servos/hs-645mg/product`

4 Circuit

The circuit for this machine is relatively straight forward. The servo motors are connected to the microcontroller (ESP32) and controlled via PWM signals. They are powered by the 5V power supply provided in the lab kit. A potentiometer is then used to control the speed of the brewing arm, and push button turns the brewing cycle on and off. A circuit diagram can be found in figure 5.

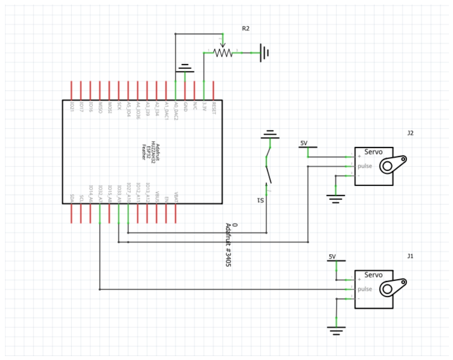

Figure 5: A circuit diagram for the auto-brewing machine

5 Finite State Machine

A diagram of the Finite State Machine (FSM) can be found in figure 6. The machine has three main states: **Brewing: Pouring**, **Brewing: Waiting**, and **Not Brewing**. Elapsed time is measured via the microcontroller timers and the button presses are detected via an interrupt. For safety reasons, the tank valve can never be opened while the machine is not brewing and automatically closes on startup and finish.

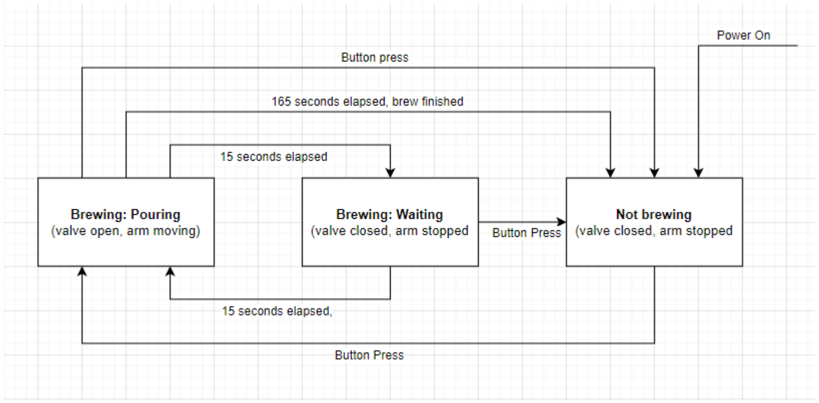

Figure 6: An FSM diagram for the auto-brewing machine

6 Appendix

6.1 Future Modifications

As this design is a rough, functional prototype, several modifications can be made to refine performance. For example, a solenoid valve should be used instead of a lever valve to make opening and closing simpler. An X-Y gantry system could also be used to make the pouring motion smoother and more configurable. Such a system would also allow for a spiraling motion of the brew tube, which is optimal for pour-over brewing. Lastly, a heating element could be added to the water tank, to further automate the brewing process. However, while these modification would refine performance, the machine still effectively automate the George Howell pour over method.

6.2 Additional figures

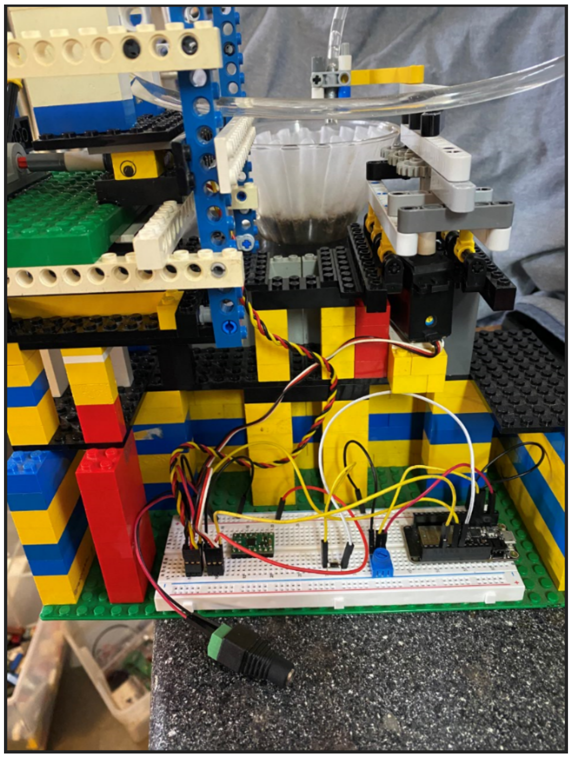

Figure 7: A rearview of the pour-over machine

6.3 Code

```
#include <Servo.h>

//#include <ESP32Servo360.h>
#include <ESP32Servo.h>
#include <analogWrite.h>
//#include <tone.h>
#include <ESP32Tone.h>
#include <ESP32PWM.h>

hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

Servo valve_servo;
Servo arm_servo;

//Input Pins
const int valve_servo_pin = 33;
const int button_pin = 27;
const int arm_servo_pin = 32;
const int pot_pin = A0;

//Loop and callback function variables
int val;
int speed_val = 90;
float last_press = 0;
volatile int valve_counter = 0;
volatile bool valve_state = false;
bool finished = false;
volatile bool brew_on = false;
volatile bool timer_active = false;



//Timer interrupt
void IRAM_ATTR pourControl() {
  //Check if 165 seconds has elapsed
  if (valve_counter == 11) {
    //Close valve and stop brewing
    valve_state = false;
    brew_on = false;
    valve_counter = 0;
  }
  //Start pouring if not pouring
  if ((valve_counter % 2) == 0) {
    valve_state = true;
    valve_counter++;
  } else {
    //Stop pouring othwerise
    valve_state = false;
    valve_counter++;
  }



}

//Button press interrupt
void isrButtonPress() {
  //Debounce detection
  if (millis() - last_press > 100) {
    last_press = millis();
    valve_state = false;
    //If not brewing, start brewing
    if (!brew_on) {
      brew_on = true;
    } else {
      //stop brewing
      brew_on = false;
    }
  }
}


void setup() {
  Serial.begin(9600);

  //Timer object creation
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &pourControl, true);
  timerAlarmWrite(timer, 15000000, true);

  //Button setup
  pinMode(button_pin, INPUT_PULLUP);
  attachInterrupt(button_pin, isrButtonPress, RISING);

  //Servo setup
  valve_servo.attach(valve_servo_pin, 500, 2400);
  arm_servo.attach(arm_servo_pin, 500, 2400);
  //Allocate timers for servos so they dont conflict with timer
      ↪ interrupt
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);

  //Close the valve on startup
  valve_servo.write(75);

  //delay to allow valve to close
  delay(5000);
}

void loop() {
  //Read in state of pot
  val = analogRead(pot_pin);

  //57 is max speed, 97 is stopped
  speed_val = map(val, 0, 4095, 50, 97);


  //Prevent interrupts from interferring with turning timer on
      ↪ and off
  portENTER_CRITICAL(&timerMux);
  if (brew_on) {
    //Serial.println("Brewing");
    //If brewing and timer not active, start timer
    if (!timer_active) {
      timerAlarmEnable(timer);
      timer_active = true;
    }
  } else {
    valve_state = false;
    //Serial.println("Not Brewing");
    //If not brewing and timer active, disable timer

    if (timer_active) {
      timerAlarmDisable(timer);
      timer_active = false;
      valve_counter = 0;
    }
  }
  portEXIT_CRITICAL(&timerMux);

  if (valve_state) {
    //open valve and move arm
    valve_servo.write(155);
    arm_servo.write(speed_val);
    Serial.println("Pouring");

  } else {
    //close valve and stop arm
    valve_servo.write(75);
    arm_servo.write(97);
    Serial.println("Closed");
  }

}
```