

Chocolatera

1st Tony Santiago Montes Buitrago
Universidad de los Andes
Ingeniería Eléctrica y Electrónica
 Bogotá, Colombia
 t.montes@uniandes.edu.co

2nd Julian David Parra Forero
Universidad de los Andes
Ingeniería Eléctrica y Electrónica
 Bogotá, Colombia
 j.parraf @uniandes.edu.co

3rd Santiago Iván Pardo Morales
Universidad de los Andes
Ingeniería Eléctrica y Electrónica
 Bogotá, Colombia
 s.pardom@uniandes.edu.co

4rd Fernando Andrés Álvarez López
Universidad de los Andes
Ingeniería Eléctrica y Electrónica
 Bogotá, Colombia
 f.alvarezl@uniandes.edu.co

Resumen—En el siguiente instructivo se explicará el montaje de la chocolatera, además del código desarrollado en la tarjeta Arduino Uno, junto con el desarrollo de la aplicación para celular en AppInventor y el funcionamiento de todo el sistema en general. Para ello, se presentarán imágenes de las diferentes etapas de montaje y se comentará sobre el código desarrollado tanto del sistema como de la aplicación.

Index Terms—Chocolatera, codificación, incorporación, instrucciones y montaje.

I. MATERIALES:

Material	Costo Unitario	Unidades
Arduino UNO	\$50.000	1
Celda de Carga	\$17.500	1
Modulo HX711	\$5.500	1
Módulo L293D	\$13.800	1
Mini Bomba de 6V	\$6.900	1
Módulo A4988D	\$6.300	1
Acrílico	\$10.000	1
Tornillos y arandelas	\$18.800	4
Manguera 6mm de diámetro	\$6.100	1
Motor Nema 17	\$35.000	1
Mini Bomba de 6V	\$8.200	1
Acrílico y Corte	\$25.600	1
Tuerca 1/4"	\$1.618	16
Varillas 1/4"	\$3.240	8
Jumpers	\$16.000	1
Hélice	\$4.500	1
Módulo A4988D	\$7.000	1
Varilla y Acople para motor Nema 17	\$41.000	1
Tornillos 3m	\$100	2

II. MONTAJE Y CÓDIGO DE ARDUINO

II-A. Sistema balanza

II-A1. Montaje y pines: La balanza y su módulo tienen 4 pines, dos para alimentación, un pin de CLOCK y un pin de

DATA. Estos dos son los pines que se conectan al Arduino y se definen en el código como **BA_DOUT** y **BA_CLK**.

II-A2. Calibración Escala: Inicialmente, se debe importar la siguiente librería para la celda de carga: **HX711.h**

Esta librería permite obtener de manera más eficiente las medidas de la celda de carga (balanza), ajustar la escala de funcionamiento y tarar (obtener el peso de desfase de la celda), con el fin de obtener la medida más precisa posible.

La primera vez que se utilice la balanza, se debe realizar una calibración de la escala de la celda, para determinar por qué factor se multiplica el valor arrojado por la celda. Para esto se puede emplear el siguiente código:

```

12 void CalibracionBalanza(void)
13 {
14     Serial.println("CALIBRACIÓN DE LA BALANZA");
15     Serial.println(" ");
16     delay(100);
17     Serial.println("No ponga ningún objeto sobre la balanza");
18     Serial.println(" ");
19     balanza.set_scale(); //Ajusta la escala a su valor por defecto que es 1
20     balanza.tare(20); //El peso actual es considerado "Tara".
21     delay(500);
22     Serial.print("...destarando...");
23     delay(200);
24
25     Serial.println(" ");
26     Serial.println(" ");
27     Serial.print("Coloque un peso de ");
28     Serial.print("Peso_conocido");
29     Serial.println(" ");
30     Serial.println("Presione el botón.");
31     while (DigitalRead(BtnInput)) {} // Esperar a que se oprima el botón
32     antDebounce(BtnInput);
33     Serial.println("Espera...");
34
35     Promedio = balanza.get_units(100); //Obtiene el promedio de las mediciones análogas según valor ingresado
36     Escala = Promedio / Peso_conocido; // Relación entre el promedio de las mediciones análogas con el peso conocido en gramos
37     Serial.println("Escala:");
38     Serial.println(" ");
39     Serial.print(" ");
40     Serial.println("Peso el vaso");
41     for (int i = 3; i > 0; i--) {
42     {
43         Serial.print("... ");
44         Serial.print(i);
45         delay(1000);
46     }
47
48     Serial.println("\nEspera...");
49
50     float PesoVaso = balanza.get_units(20);
51     Serial.print("El vaso tiene un peso de ");
52     Serial.print(PesoVaso/Escala);
53     Serial.println(" ");
54
55     balanza.tare(100);
56
57     balanza.set_scale(Escala); // Ajusta la escala correspondiente
58 }

```

Figura 1. Código Arduino: Calibración de la balanza

En este código debes reemplazar la variable "Peso_conocido" por el peso de un objeto que conozcas que puedas poner sobre la balanza; cuando corras el código, pon dicho objeto sobre la balanza, observa y anota el valor de "Escala".

Posterior a esto, se puede utilizar dicho valor de "Escala" para medir el peso de la balanza de manera precisa; en el caso que se enseñará en el código a continuación, este valor fue de -1308.5.

II-A3. Variables: A continuación se describen las variables que se utilizan para el funcionamiento de la balanza:

```

19 /* ----- VARIABLES MODULADAS ----- */
20 // Balanza
21 float pesoConocido = 200; // SOLO para hallar la escala de la balanza
22 float escala = -1308.5; // Escala de la balanza (después de calibrar)
23 float pesoVaso = 41; // Peso real del vaso
24 float toleranciaVaso = 3;
25 int numMedidas = 1; // Número de medidas para promediar (entre más, más lento)
26
27 /* ----- VARIABLES DEL CÓDIGO ----- */
28 // Balanza
29 HX711 balanza(BA_DOUT, BA_CLK); // Balanza
30 float pesoRecipiente = 0; // Peso del recipiente
31 float pesoActual; // Peso medido por la balanza
32
33
34
35
36
37

```

Figura 2. Código Arduino: Variables de funcionamiento de la balanza

Las "Variables del código" se modificarán a lo largo del programa, mientras que las "Variables moduladas" son valores constantes que definen el comportamiento del sistema; estas son:

- pesoConocido: Variable utilizada anteriormente para calibrar la balanza.
- escala: Valor de la escala de la celda, encontrada en la calibración.
- pesoVaso: Peso real del vaso que se va a poner sobre la balanza comúnmente (este valor se utiliza para saber si la balanza está midiendo correctamente).
- toleranciaVaso: Valor (en gramos) alrededor del peso del vaso, que es aceptado.

■ numMedidas: Número de medidas promediadas para determinar el valor medido por la balanza. En este caso solo se utiliza una medición, para que el código funcione más rápido.

II-A4. Funcionamiento: Ahora bien, para la medición de la balanza y para tarar la balanza y verificar que esté en el rango correcto, se definen las funciones calibracionBalanza y medidaBalanza:

Nota: La variable "BT" corresponde a la comunicación serial por Bluetooth **II-B**; para una primera implementación de solo la balanza, se podría eliminar.

```

183 // Función de calibración de la balanza
184 void calibracionBalanza() {
185     while (true) {
186         BT.println("||CALIBRANDO BALANZA||");
187         Serial.println("||CALIBRANDO BALANZA||");
188         BT.println("No ponga ningún objeto sobre la balanza");
189         Serial.println("No ponga ningún objeto sobre la balanza");
190         delay(500);
191         balanza.tare(20); // Tara sin el recipiente
192
193         BT.println("\nColoque el recipiente sobre la balanza");
194         Serial.println("\nColoque el recipiente sobre la balanza");
195         for (int i = 0; i < 3; i++) {
196             delay(1000);
197             BT.println(i);
198             Serial.println(i);
199         }
200
201         pesoRecipiente = balanza.get_units(20);
202         BT.print("\nPeso del recipiente: ");
203         Serial.print("\nPeso del recipiente: ");
204         BT.print(pesoRecipiente/escala);
205         Serial.print(pesoRecipiente/escala);
206         BT.printIn(" g");
207         Serial.printIn(" g");
208
209         if ((pesoRecipiente - escala - toleranciaVaso) > pesoVaso || (pesoRecipiente/escala + toleranciaVaso) < pesoVaso) {
210             BT.println("Calibración erroneaVolviendo a medir");
211             Serial.println("Calibración erroneaVolviendo a medir");
212             delay(4000);
213         } else {
214             balanza.tare(100);
215             balanza.set_scale(escala);
216             break;
217         }
218     }
219 }
220
221
222
223 // Función de medición de la balanza (actualiza la variable pesoActual directamente)
224 void medidaBalanza(int n) {
225     if ((now - lastTimeBalanza) >= intervalPrintBalanza) {
226         BT.print("Peso actual: ");
227         BT.print(pesoActual);
228         BT.println(" g");
229         lastTimeBalanza = now;
230     do {
231         pesoActual = balanza.get_units(n); // Entrega el peso actualmente medido en gramos
232     } while (pesoActual > 1000); // if peso > 1000, se mide de nuevo
233     }
234 }
235

```

Figura 3. Código Arduino: Función calibracionBalanza

```

222 // Función de medición de la balanza (actualiza la variable pesoActual directamente)
223 void medidaBalanza(int n) {
224     if ((now - lastTimeBalanza) >= intervalPrintBalanza) {
225         BT.print("Peso actual: ");
226         BT.print(pesoActual);
227         BT.println(" g");
228         lastTimeBalanza = now;
229     do {
230         pesoActual = balanza.get_units(n); // Entrega el peso actualmente medido en gramos
231     } while (pesoActual > 1000); // if peso > 1000, se mide de nuevo
232     }
233 }
234

```

Figura 4. Código Arduino: Función medidaBalanza

Podemos observar que solo se imprimirá el peso de la balanza cada cierto intervalo de tiempo; esto se hace de esta forma ya que la acción de "Imprimir por Serial" tanto en consola como por Bluetooth es una tarea que toma cierto tiempo para el Arduino, por lo cuál para tareas más rápidas es mucho más eficiente no imprimir todo el tiempo.

II-B. Sistema Bluetooth

II-B1. Montaje y pines: Inicialmente, se deben importar las siguientes librerías para la comunicación serial por Bluetooth: **SoftwareSerial.h**

El módulo Bluetooth posee 6 pines de los cuales se utilizarán únicamente 4, 2 de alimentación, un pin TX (transmisión de datos) y un pin RX (recepción de datos).

Estos dos pines se definirán como los pines **BT_RX** y **BT_TX** y ¡Deben ir conectadas de manera opuesta a como se definirán en el código!; por ejemplo, si se conecta el pin TX del módulo bluetooth al pin 10, en el código ese será el pin BT_RX.

Esto se hace de esta forma debido a que la transmisión del Arduino será la recepción del módulo y viceversa.

II-B2. Variables: Posteriormente, se define la variable BT que corresponde al Serial bluetooth:

```
62
63 // Modulo Bluetooth
64 SoftwareSerial BT(BT_RX, BT_TX);
65
```

Figura 5. Código Arduino: Variable BT para comunicación bluetooth

II-B3. Funcionamiento: El serial del módulo bluetooth funciona de manera casi idéntica al serial de Arduino, por lo cuál para mayor simplicidad y eficiencia, se utilizarán los métodos `println` y `print` para imprimir en el serial.

De igual manera, el módulo se debe inicializar como un serial de arduino con una tasa de baudios (baud rate); comúnmente el valor utilizado es 9600: `BT.begin(9600);`

II-C. Sistema bomba

II-C1. Montaje y pines: Para el montaje de la minibomba se debe conectar al puente H y este al Arduino, con esto se puede modular el PWM de la minibomba para dosificar agua. La minibomba se utilizan dos pines del puente H para el control de la minibomba; uno hacia adelante (pin **MB_FRONT**) y otro hacia atrás (pin **MB_BACK**), en este caso solo se utilizará el pin hacia adelante.

II-C2. Variables: A continuación se describen las variables que se utilizan para el funcionamiento de la dosificación por minibomba:

```
26
27 // Minibomba
28 float pesoOffset = 1.5; // El peso anterior desde el cual se apaga la minibomba
29 float pwmFuncionamiento = 255; // % (0-255) del PWM de funcionamiento de la minibomba
30 int densidad = 1; // Densidad del líquido a dosificar (g/mL)
31
32 // Minibomba
33 int firstTime = 1; // Para calcular el caudal de la minibomba (1 | 0)
34 int timeCaudal = 0; // Tiempo transcurrido para calcular el caudal (minibomba)
35 float caudal = 0; // Caudal de la minibomba (mL/s)
```

Figura 6. Código Arduino: Variables de funcionamiento de la minibomba

Las variables del código permiten calcular el caudal e identificar cuándo empieza la dosificación; por otra parte, existen ciertas variables modificables que son constantes en el código, estas son:

- **pesoOffset:** Valor en gramos desde el cuál se apagará la minibomba. Esto es para conseguir una mayor precisión, ya que cierta cantidad de agua se queda en la manguera cuando se apaga la minibomba.

- **pwmFuncionamiento:** El PWM (valor entre 0 y 255) al que funciona la minibomba, en este caso se estableció en 255 que corresponde a un PWM del 100 %.
- **densidad:** Densidad del líquido a dosificar (en este caso 1, que es la densidad en g/mL del agua).

II-C3. Funcionamiento: Para el funcionamiento de la minibomba, se definió un código en el loop y una función para imprimir el estado y calcular el caudal de la minibomba:

```
122 /*
123 * ----- MINIBOMBA -----
124 if (modeVol) {
125     printStatusMinibomba();
126     // ENCENDIDA
127     if (onOff > 0) {
128         if (firstTime) {
129             //timeCaudal = now; // Reiniciar el tiempo para medir el caudal
130             firstTime = 0;
131             BT.println("Encendiendo minibomba!");
132             analogWrite(MB_FRONT, pwmFuncionamiento * onOff);
133         }
134
135     medidaBalanza(numMedidas);
136
137     if (!(pesoDeseado >= (pesoActual + pesoOffset))) {
138         onOff = 0;
139         analogWrite(MB_FRONT, 0);
140     }
141     // APAGADA
142     else {
143         medidaBalanza(5);
144         medidaBalanza(15);
145         if (pesoDeseado >= (pesoActual + pesoOffset)) {
146             firstTime = 1;
147             onOff = modeMix ? 0.65 : 0.75; // Se enciende la minibomba al 75% de su PWM
148         }
149     }
150     else {
151         BT.println("La dosificación ha finalizado");
152         modeVol = 0;
153         modeChanged();
154     }
155 }
156 }
```

Figura 7. Código Arduino: Código loop de la minibomba

```
235 // Función de impresión del estado de la minibomba
236 void printStatusMinibomba(void) {
237     if (onOff > 0) {
238         if ((now - lastTimeMinibomba) >= intervalPrintMinibomba) {
239             BT.println("Minibomba ENCENDIDA");
240             BT.print("PWM: ");
241             BT.print((pwmFuncionamiento * onOff)/2.55);
242             BT.println("%");
243             lastTimeMinibomba = now;
244         }
245     } else {
246         BT.println("Minibomba APAGADA");
247
248         caudal = 1000*(pesoActual/densidad)/(now - timeCaudal);
249         // (ms/s) * (g / (g / mL)) / (ms) = (1/s) * (mL) = mL/s
250         BT.print("Caudal: ");
251         BT.print(caudal);
252         BT.println(" mL/s");
253     }
254 }
255 }
```

Figura 8. Código Arduino: Función `printStatusMinibomba`

De igual manera, el estado de la minibomba solo se imprime cada cierto tiempo para optimizar el código, y en el código loop se utiliza el método de control On-Off para dosificar la cantidad exacta de volumen con cierta modificación de que se puede volver a iniciar a dosificar cuando se observe que no se dosificó el volumen completo.

II-D. Sistema motor

II-D1. Montaje y pines: Para el montaje del motor paso a paso y su encoder, se puede detallar de manera más precisa el modelo circuitual [III](#).

Principalmente, para su funcionamiento (luego de realizar las conexiones necesarias del motor al driver), el motor se maneja mediante 2 pines principales: Uno de dirección y uno de pasos. En este caso, no es relevante para el proyecto controlar la dirección del motor, por lo cuál se definirá únicamente el pin **MP_STEP**.

Por otra parte, el encoder, que es el dispositivo encargado de medir la velocidad en RPM del motor, posee dos canales para indicar la velocidad del motor (Canal A y Canal B) que se definirán en el código como **MP_ENCA** y **MP_ENCB**.

II-D2. Variables: Para el funcionamiento del motor y del encoder, se definieron en el código las siguientes variables:

```

46 // Motor PaP
47 float onOff = 1; // % (0-1) del PWM del motor PaP
48 volatile int encoderCount = 0; // Contador de ticks del encoder
49 double rpm; // RPM del motor PaP, medida por el encoder
50 double delayRpm; // Delay calculado (en us) para ir a los RPM deseados
51
52 int modeMix = 0; // Modo de mezclado (0: APAGADO | 1: ENCENDIDO)
53 float rpmDeseado; // RPM deseado a mezclar (se especifica por consola)
54
55

```

Figura 9. Código Arduino: Variables del motor y el encoder

En donde `encoderCount` permite medir el número de pulsos del encoder y `rpmDeseado` y `delayRpm` se encargan de controlar que el motor vaya a una velocidad específica.

II-D3. Funcionamiento: La velocidad en RPM del motor se controla mediante la función `delayMicroSeconds`, que tiene el código por cierta cantidad de microsegundos y permite así definir cuántas revoluciones queremos lograr.

Para obtener la transformación $\text{RPM} \rightarrow \text{delay}$, se debe primero caracterizar el motor y hacer una gráfica del número de RPM que se obtienen para cierto delay; luego de obtener dicha gráfica se hace una linealización y se saca la inversa de dicha linealización (puede ser cuadrática, exponencial, lineal, etc).

A continuación se muestra el código del motor en el loop y las funciones del encoder que permiten conocer las RPM a las que va el motor:

```

156
157 /* ----- MOTOR ----- */
158 if (modeMix) {
159     printRpmMotor();
160     // Si no está encendido el modo de volumen, igual toca mostrar el peso
161     medidaBalanza(numMedidas);
162
163     digitalWrite(MP_STEP, HIGH);
164     delayMicroseconds(delayRpm);
165     digitalWrite(MP_STEP, LOW);
166     delayMicroseconds(delayRpm);
167 }

```

Figura 10. Código Arduino: Código loop del motor

```

95 pinMode(MP_STEP, OUTPUT);
96 pinMode(MP_ENCB, INPUT_PULLUP);
97
98 attachInterrupt(digitalPinToInterrupt(MP_ENCB), countPulses, CHANGE);
99
100
179
180 // Función de conteo de ticks del encoder
181 void countPulses(void) {
182     encoderCount++;
183 }

```

Figura 11. Código Arduino: Conteo de pulsos del encoder

```

256 // Función de impresión de los RPM del motor
257 void printRpmMotor(void) {
258     if ((now - lastTimeMotor) >= intervalPrintMotor) {
259         rpm = ((double) encoderCount/960)*60;
260         BT.print("RPM: ");
261         BT.println(rpm);
262         encoderCount = 0;
263         lastTimeMotor = now;
264     }
265 }
266
267

```

Figura 12. Código Arduino: Función `printRpmMotor`

II-E. Sistema Temperatura

II-E1. Montaje y pines: Inicialmente, se deben importar las siguientes librerías para el sensor de temperatura: [OneWire.h](#) y [DallasTemperature.h](#)

El sensor de temperatura posee 3 pines de funcionamiento, 2 de alimentación y uno de datos, que es el único que se utiliza para obtener la información de la temperatura. Este pin corresponde al pin **TMP**.

II-E2. Variables: Se definieron las siguientes variables para el manejo de variables del sensor de temperatura:

```

55 // Temperatura
56 OneWire oneWireObjeto(TMP);
57 DallasTemperature sensorDS18B20(&oneWireObjeto);
58 float tempActual; // Temperatura actual del líquido
59
60 int modeTemp = 0; // Modo de medir temperatura (0: APAGADO | 1: ENCENDIDO)
61
62

```

Figura 13. Código Arduino: Variables del sensor de temperatura

II-E3. Funcionamiento: El funcionamiento del sensor de temperatura es bastante sencillo; se utilizarán los métodos integrados de las librerías para obtener la temperatura constantemente.

En el código loop lo único que se hace es imprimir la temperatura y el peso de la balanza (siempre se debe imprimir el peso de la balanza).

```

169 /*
170 * ----- TEMPERATURA -----
171 if (modeTemp) {
172     medidaBalanza(numMedidas);
173     printTemp();
174 }
175

```

Figura 14. Código Arduino: Código loop del sensor de temperatura

```

267 // Función de impresión de la temperatura
268 void printTemp(void) {
269     if ((now - lastTimeTemp) >= intervalPrintTemp) {
270         sensorDS18B20.requestTemperatures();
271
272         BT.print("Temperatura sensor 1: ");
273         BT.print(sensorDS18B20.getTempCByIndex(0));
274         BT.println(" C");
275
276         lastTimeTemp = now;
277     }
278 }
279 }
```

Figura 15. Código Arduino: Función printTemp

II-F. Sistema Completo

Para el funcionamiento completo de todos los sistemas en conjunto, se definió un algoritmo que funciona mediante el reconocimiento de comandos, similar al concepto de "CLI". Este algoritmo admite ciertos comandos con parámetros y ciertos sin parámetros. Todos los comandos admitidos se presentan a continuación:

- vol X: Dosifica X mililitros mediante la minibomba.
- mix X: Mezcla a X revoluciones por minuto (rpm) con el motor paso a paso.
- temp: Iniciar a medir temperatura.
- bal: Vuelve a calibrar la balanza.
- off Y: Apaga el proceso Y, donde Y puede ser "vol", "mix" o "temp".
- stop: Detiene todos los procesos actuales.

Así mismo, se verifica en el código loop cuáles procesos están encendidos actualmente para ejecutarlos; esto permite que se ejecuten concurrentemente múltiples procesos, como mezclar, dosificar, medir temperatura y medir el peso al mismo tiempo.

Es importante aclarar que dado que el Arduino cuenta con un único procesador, realmente no ejecuta tareas concurrentemente; por el contrario el código se optimizó al máximo posible de modo que con un único Arduino sea posible que se ejecuten los procesos de manera "simultánea".

El código completo de Arduino se puede encontrar en este enlace de GitHub:

<https://github.com/t-montes/Chocolatera-Instrumentacion>

III. MONTAJE

A continuación se propone que se hagan las siguientes conexiones eléctricas en este montaje, en él se muestra todas las conexiones de los componentes que tienen relación directa con el Arduino

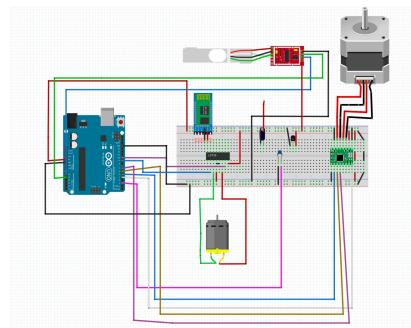


Figura 16. Imagen visual del montaje en físico

En las siguientes imágenes se muestra el montaje físico de la figura 16, en las cuales se puede ver integrado el sistema balanza-bomba de agua-motor-temperatura y IoT.

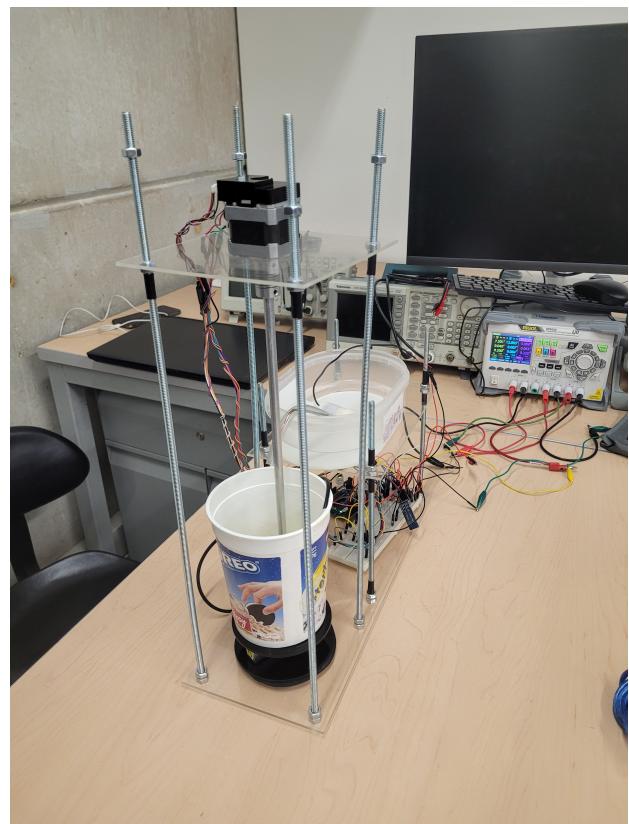


Figura 17. Imagen visual del montaje en físico

IV. APLICACIÓN

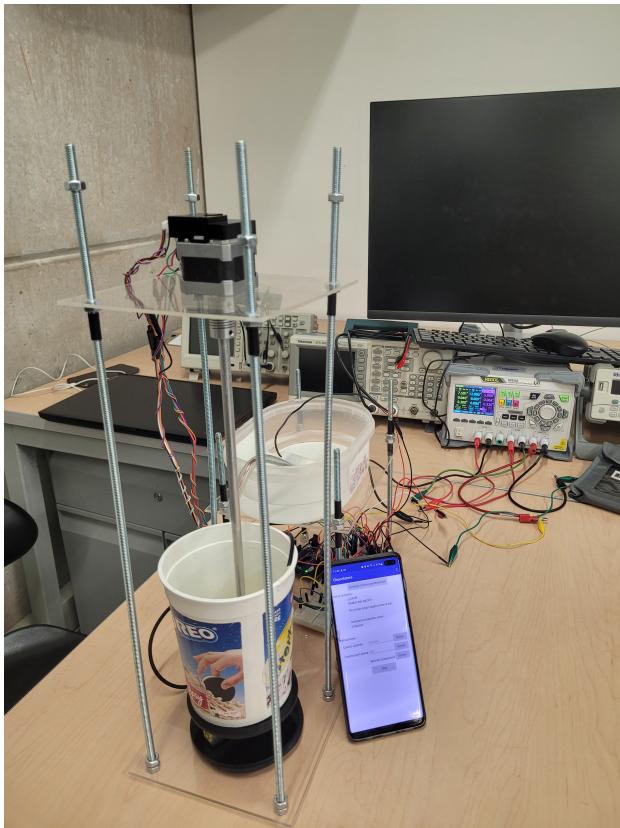
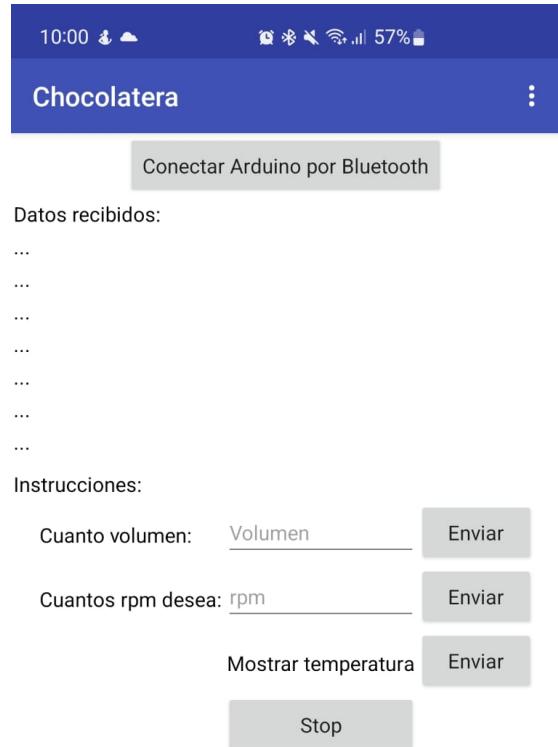


Figura 18. Imagen visual del montaje en físico y aplicación en celular

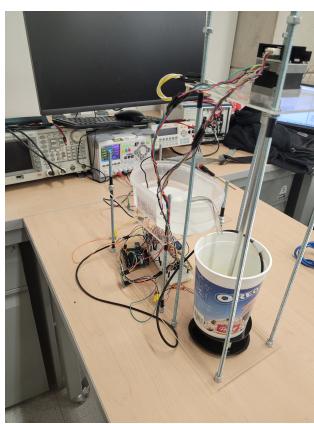


Figura 19. Imagen visual del montaje en físico

Figura 20. Imagen visual de la aplicación Chocolatera

La aplicación desarrollada con MIT App Inventor presenta una característica llamada "Bluetooth Client" que facilita la conexión con el módulo HC-05. Se ha integrado un botón específico en la interfaz de la aplicación para establecer la conexión con dicho módulo. Una vez establecida la conexión, el usuario puede enviar comandos personalizados, como ajustar el volumen deseado o la velocidad de rotación del motor en rpm. Además, se ha implementado un botón de parada que interrumpe todas las funciones, a excepción del pesaje proporcionado por la balanza. Esta funcionalidad adicional brinda al usuario un control preciso y flexible sobre el sistema, permitiéndole detener las operaciones según sus necesidades. Asimismo, en la parte superior de la interfaz se muestra la información suministrada por todos los sensores conectados a través del Arduino. La comunicación entre la aplicación y el

Arduino se realiza mediante el módulo HC-05, que transmite la información a través del puerto serial del Arduino.

En la siguiente imagen se evidencia como se configuro cada botón y cada elemento de la aplicación:

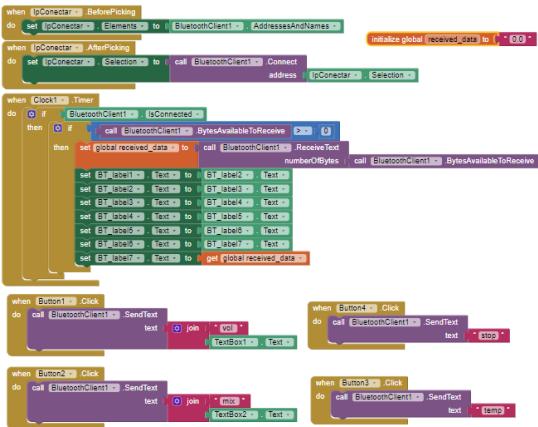


Figura 21. Configuración de la aplicación Chocolatera

Se puede evidenciar en la figura 21 se utiliza el bloque "Clock" para generar eventos periódicos en un intervalo de tiempo específico, ya que es necesario enviar información desde el módulo HC05 a la aplicación de manera regular. Esto permite que, en intervalos programados, el Arduino envíe actualizaciones sobre el estado del sistema a la aplicación, brindando al usuario la posibilidad de monitorear su funcionamiento en tiempo real.

V. FUNCIONAMIENTO

V-A. Sistema balanza

Para medir el peso del líquido en el recipiente utilizando la balanza, se siguió un procedimiento específico. Primero, se pesó el recipiente junto con el sensor de temperatura, sumando aproximadamente 40 gramos al peso total. Luego, se realizó una tara de la balanza sin ningún objeto encima y luego con el recipiente colocado, de modo que la balanza mostrara exclusivamente el peso del recipiente. Una vez realizada la tara, el sistema se estableció en 0 gramos, lo que permitió obtener una medición precisa del peso del líquido agregado al recipiente. Durante todo el proceso, la balanza se mantuvo activa para registrar los cambios de peso, aunque se implementaron intervalos en los que se realizaban pesajes menos frecuentes para evitar afectar el funcionamiento de la mini bomba y el motor. De esta manera, se logró un control y registro adecuados del peso en diferentes etapas del proceso, proporcionando información precisa sobre la cantidad de líquido presente en el recipiente.

V-B. Sistema Bluetooth

El sistema de comunicación Bluetooth se implementó utilizando el módulo HC-05, el cual es capaz de operar tanto como maestro como esclavo. Esta capacidad es importante ya que permite establecer una conexión bidireccional entre el sistema y un dispositivo móvil, como un teléfono celular. Al configurar

el HC-05 como maestro, se facilita el control del sistema desde la aplicación en el celular, permitiendo enviar comandos y recibir datos de forma efectiva. Por otro lado, cuando se configura como esclavo, el HC-05 puede recibir instrucciones y enviar información relevante al dispositivo móvil, lo que resulta útil para mostrar el estado actual del sistema en la interfaz de la aplicación.

V-C. Sistema bomba

Durante la operación de la bomba, cuando se encuentra en el estado de suministro de líquido y simultáneamente con el estado activo de la balanza (como se mencionó anteriormente), se utiliza un PWM del 100 % para asegurar un suministro constante de líquido sobre el recipiente. En este estado, la balanza realiza mediciones continuas para minimizar cualquier error en la cantidad de agua solicitada por el usuario. Sin embargo, cuando la bomba se encuentra en otro estado, el tiempo de pesaje de la balanza se limita para que no afecte los otros estados. Por esta razón, se ajusta el PWM al 75 % con el fin de obtener la mayor precisión posible en la cantidad de líquido deseada por el usuario

V-D. Sistema motor

El sistema de motor utilizado en el proyecto se basa en un motor Nema 17, con un rango de operación de velocidad que varía entre 60 y 120 rpm. Este rango fue determinado a través de pruebas experimentales, donde se obtuvo una curva de tendencia que se ajustaba de manera óptima a los datos recopilados, relacionando el tiempo de retardo en milisegundos y el número de revoluciones del motor. El motor pasó a paso funciona mediante la emisión de pulsos de retardo controlados por el Arduino, lo que nos permite desarrollar una función para regular la velocidad de rotación de acuerdo con las preferencias del usuario. Para medir las rpm en tiempo real, se utilizó un encoder H9700, el cual cuenta las vueltas realizadas por el motor y, a través de una fórmula específica, se obtienen los valores de rpm. Además, se empleó un driver A4988 conectado al Arduino, encargado de controlar el movimiento del motor de manera precisa. Estos componentes en conjunto permiten un control preciso de las rpm y el movimiento del motor, asegurando un funcionamiento eficiente y ajustable a las necesidades del proyecto.

V-E. Sistema Temperatura

Según la información obtenida del Datasheet de la sonda DS18B20, esta proporciona una lectura de temperatura con una precisión de 9 a 12 bits, lo que permite obtener mediciones detalladas de la temperatura del dispositivo. En el contexto del sistema en el que se utiliza, al obtener las mediciones de temperatura simultáneamente, se produce un ligero retardo en el proceso mientras se recopila el dato de temperatura del líquido en el recipiente. Durante este breve tiempo de retardo, el sistema se asegura de obtener de manera precisa y fiable la temperatura del líquido antes de continuar con las siguientes etapas o acciones del proceso.

VI. ANEXOS

Se va a anexar estos archivos en pdf para cortar en láser.

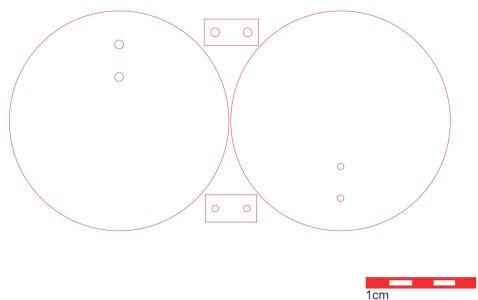


Figura 22. Imagen de la estructura de la balanza

VII. ANEXOS

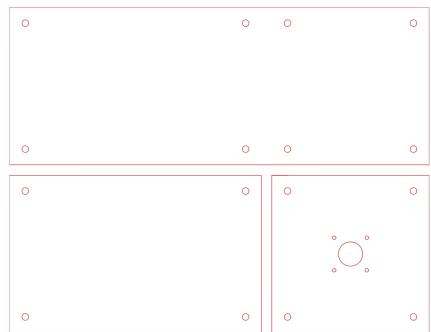


Figura 23. Imagen de la estructura de todo el sistema