

Construcción de un Muñeco Animado Controlado por Motor de Pasos

Descripción General:

El proyecto consiste en armar y construir un muñeco animado que gira y aumenta su velocidad de acuerdo a la cantidad de monedas que se le coloquen. El sistema debe tener dos modos de funcionamiento: automático y manual.

- Modo Manual: Mediante el uso de un teclado matricial, se puede configurar la velocidad máxima de la rueda giratoria, la cantidad máxima de monedas, y si va a ser alimentado con monedas de un dólar o de diez centavos.
- Modo Automático: El muñeco gira de acuerdo a la velocidad configurada y a la cantidad de dinero depositado en el muñeco.

El sistema debe mostrar en un módulo ht16k33 la velocidad actual, la cantidad de monedas colocadas y el valor en dólares depositados en el muñeco, con una pausa de un segundo entre cada información.

Requisitos Funcionales:

1. El sistema debe permitir la configuración de la velocidad máxima de la rueda giratoria en modo manual.
2. El sistema debe permitir la configuración de la cantidad máxima de monedas en modo manual.
3. El sistema debe permitir la configuración del tipo de moneda (dólar o diez centavos) en modo manual.

4. En modo automático, el muñeco debe girar de acuerdo a la velocidad configurada y a la cantidad de dinero depositado.

5. El sistema debe mostrar la velocidad actual en el módulo ht16k33.

6. El sistema debe mostrar la cantidad de monedas colocadas en el módulo ht16k33.

7. El sistema debe mostrar el valor en dólares depositados en el módulo ht16k33.

8. La información mostrada en el módulo ht16k33 debe cambiar con una pausa de un segundo entre cada dato.

Requisitos No Funcionales:

1. El sistema debe ser confiable y tener un tiempo de inactividad mínimo.

2. El sistema debe ser fácil de usar y configurar.

3. El sistema debe ser seguro para los usuarios.

4. El sistema debe ser duradero y capaz de soportar el uso continuo.

5. El sistema debe tener una respuesta rápida al insertar monedas y al configurar mediante el teclado matricial.

6. El sistema debe ser compatible con monedas de diferentes denominaciones (dólar y diez centavos).

Lista de Materiales:

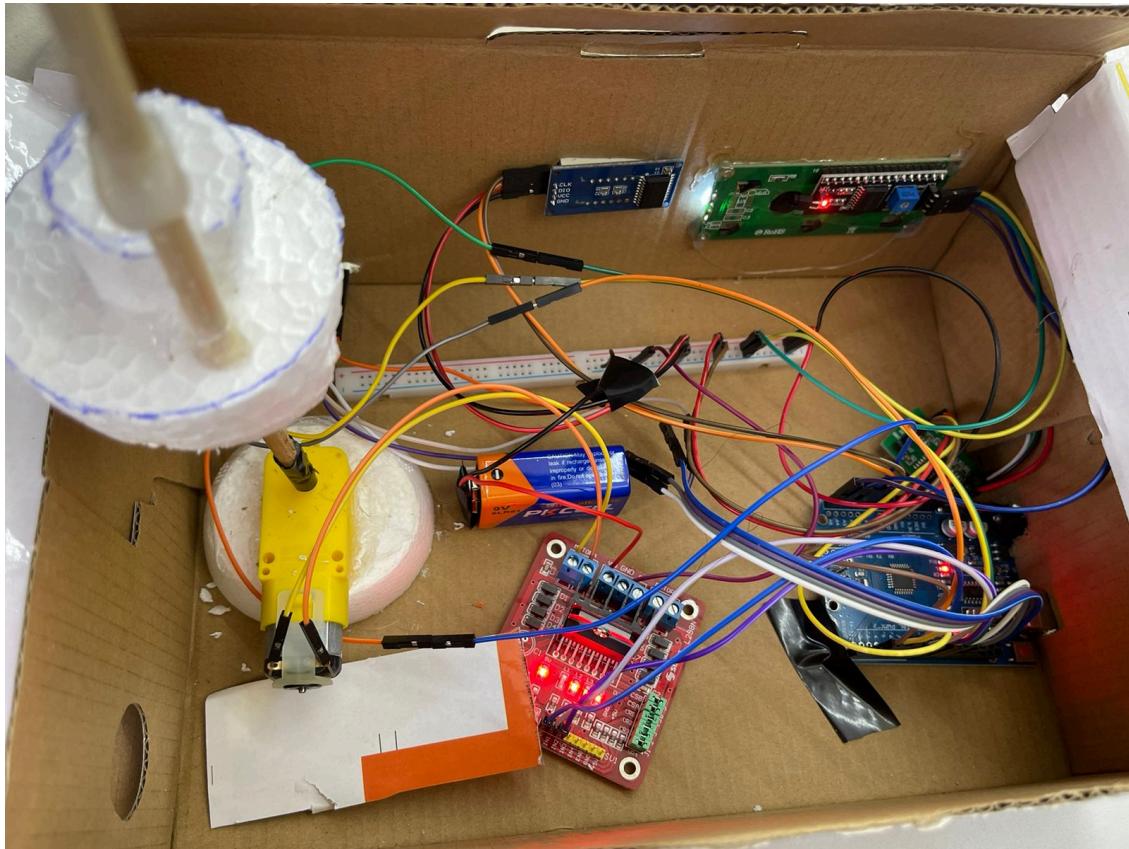
1. Muñeco animado

2. Motor de pasos (en este caso, se usará un motor DC)

3. Controlador de motor de pasos (L298N)

4. Teclado matricial
5. Módulo ht16k33
6. Microcontrolador Arduino
7. Sensor de monedas (celda de carga y módulo HX711)
8. Fuente de alimentación
9. Cableado y conectores
10. Estructura y soporte para el muñeco y componentes
11. Display para el módulo ht16k33
12. Carcasa para protección de componentes electrónicos Estos son los elementos esenciales y las especificaciones para la construcción del muñeco animado controlado por motor de pasos.

A continuación, se presenta una pequeña documentación sobre el proyecto “Muñeco animado controlado por motor DC” de MICROCONTROLADORES 1 del segundo semestre.

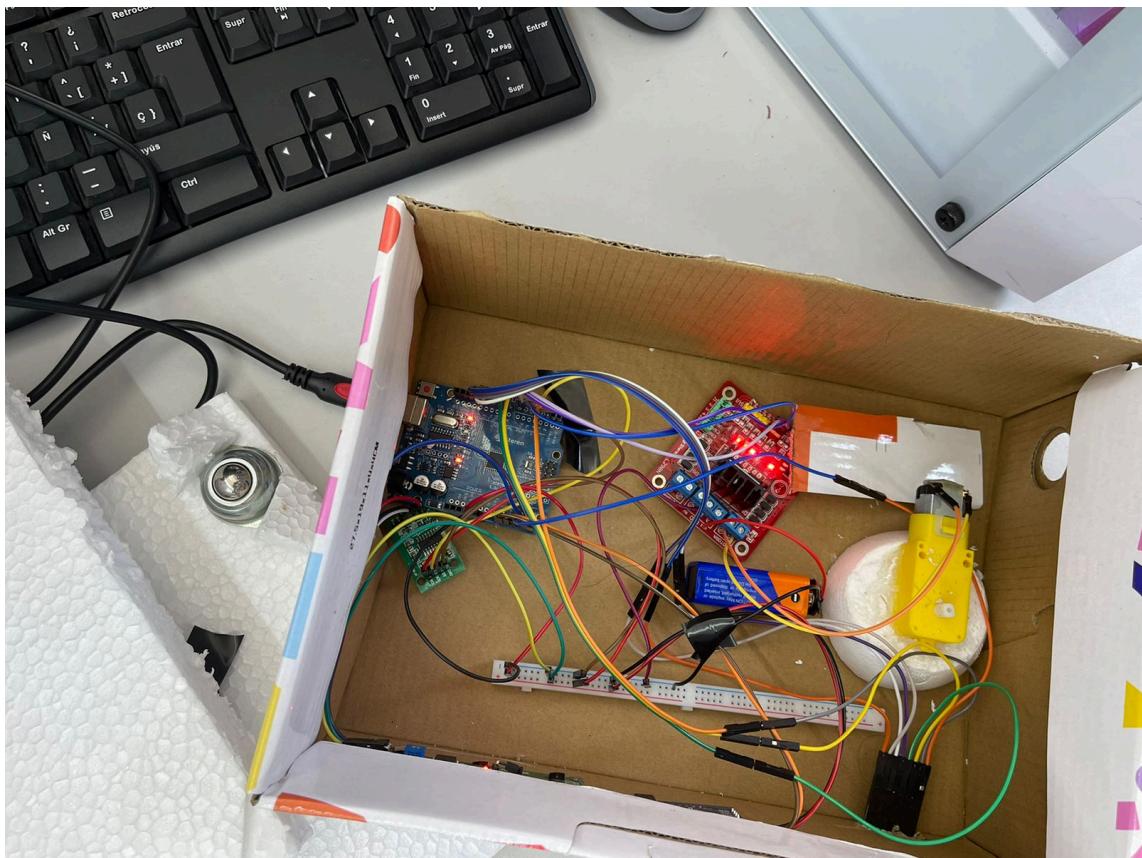


Los componentes mostrados de derecha a izquierda son:

Arduino (placa de desarrollo), módulo HX711 (encima del arduino, color verde) pantalla LCD I2C (en la pared de la caja, color verde), módulo HT16K33 (en la pared de la caja, color azul), batería de 9v, módulo L298N (color rojo), motor DC.

No apreciables en la imagen: Keypad 4x4 y celda de carga 1kg.

Desde otra perspectiva:

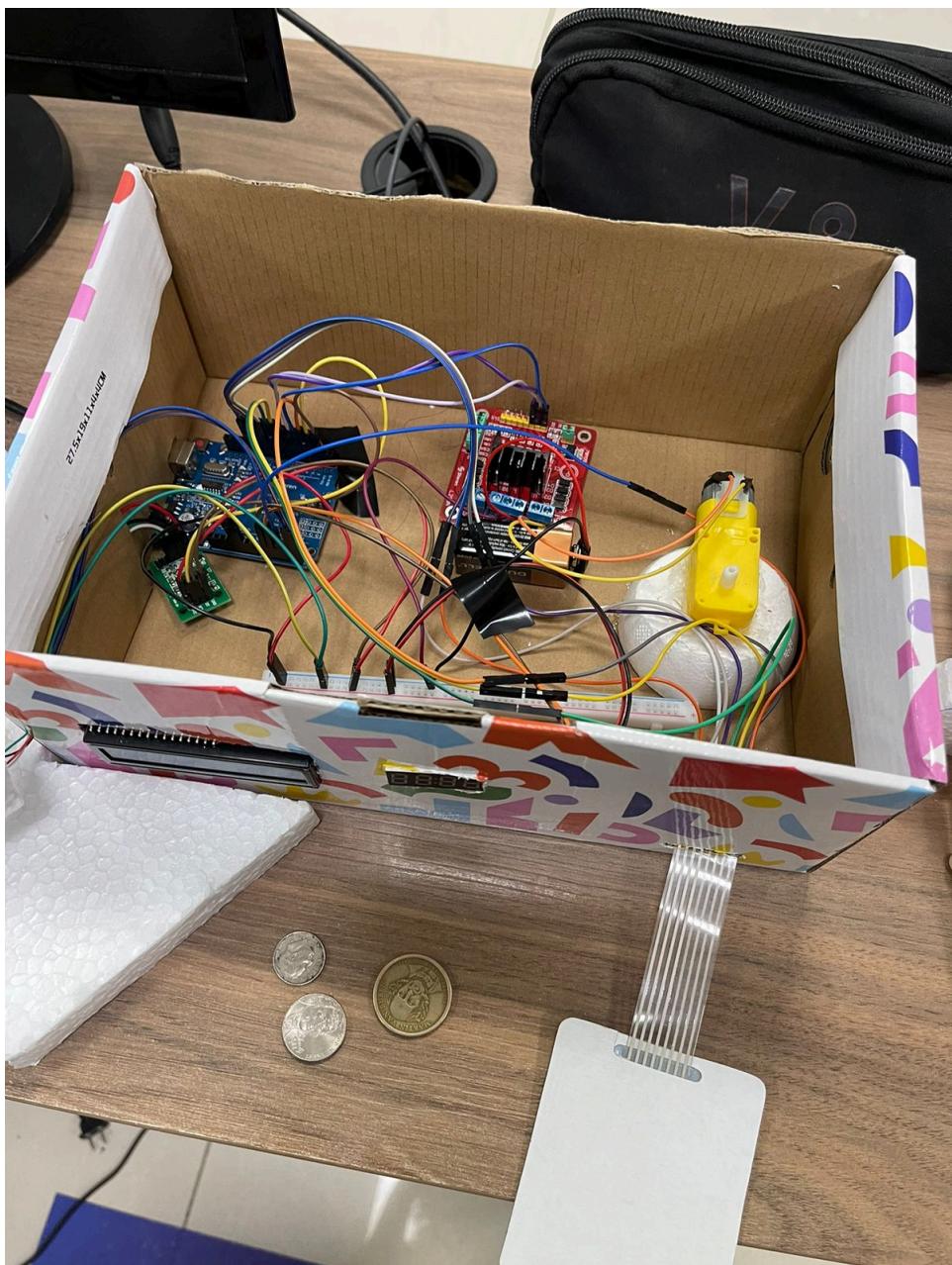


Se aprecia mejor el módulo HX711 (color verde), a la izquierda se encuentra la plataforma de peso (balanza). En la esquina inferior derecha se pueden observar los cables conectados al Keypad 4x4.

Más imágenes de referencia:



Se aprecia la celda de carga y su debido posicionamiento.



Se aprecia el Keypad 4x4 y su debido posicionamiento, además de las monedas usadas para el proyecto.

Conexiones:

- Arduino
 - Pin 2 -> DT, módulo HX711
 - Pin 3 -> SCK, módulo HX711
 - Pin 4 -> 3era columna de Keypad 4x4
 - Pin 5 -> ENA, módulo L298N
 - Pin 6 -> IN1, módulo L298N
 - Pin 7 -> IN2, módulo L298N
 - Pin 8 -> 2da columna de Keypad 4x4
 - Pin 9 -> 1era columna de Keypad 4x4
 - Pin 10 -> 4ta fila de Keypad 4x4
 - Pin 11 -> 3era fila de Keypad 4x4
 - Pin 12 -> 2da fila de Keypad 4x4
 - Pin 13 -> 1era fila de Keypad 4x4
 - Pin A0 -> CLK, módulo HT16K33
 - Pin A1 -> DIO, módulo HT16K33
 - Pin A2 -> 4ta columna de Keypad 4x4
 - Pin A4 -> SDA, pantalla LCD I2C
 - Pin A5 -> SCL, pantalla LCD I2C
- HX711
 - -> E+, Cable rojo de la celda de carga
 - -> E-, Cable negro de la celda de carga
 - -> A+ Cable verde de la celda de carga
 - -> A-, Cable blanco de la celda de carga
- L298N

- Motor A (El orden es irrelevante, pero ambos conectores van al motor DC en sus contactos + y -).

Todos los VCC y GND van conectados a un segmento de protoboard y son alimentados por el arduino, excepto el módulo L298N que es directamente alimentado con una batería de 9V. Considerar que todos los componentes deben tener una tierra común para evitar dañarlos.

En el caso particular del módulo L298N y la batería de 9v:

- VMS -> Batería 9v (+)
- GND -> Tierra común (segmento de protoboard)
- Batería 9v (-) -> Tierra común (segmento de protoboard)

Lógica implementada:

El proyecto presenta dos modos descritos anteriormente, en el modo automático (activado por defecto al alimentar al proyecto) el muñeco trabaja según el peso de la cantidad de monedas en la balanza.

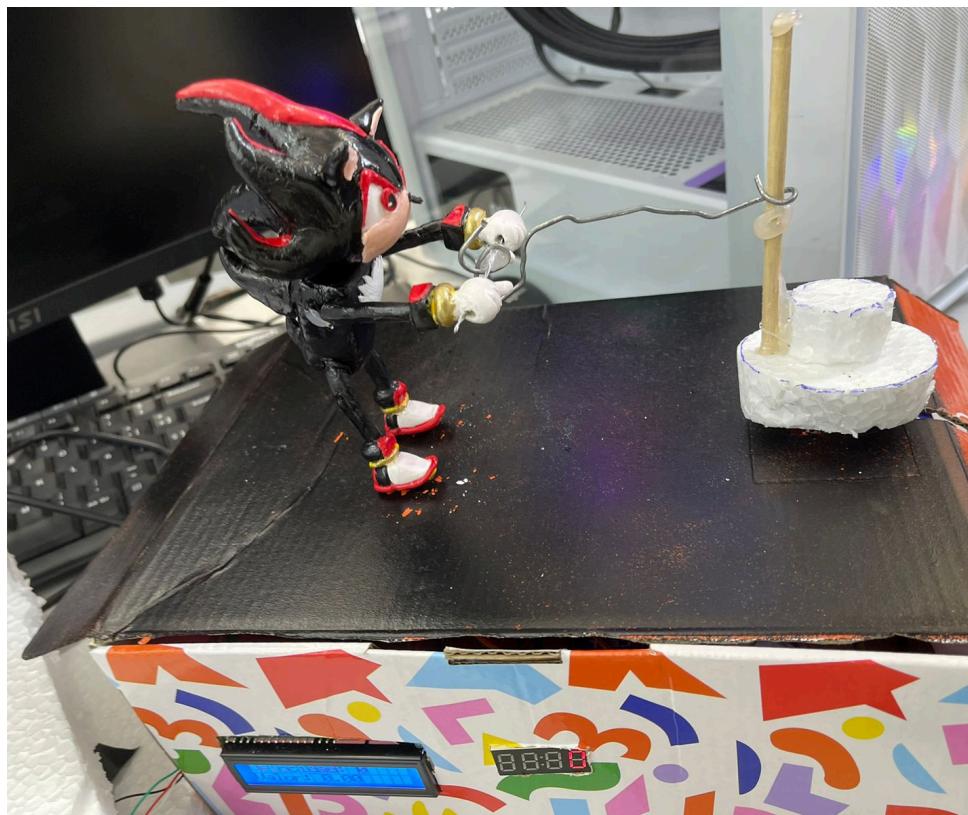
En el modo manual (activado por la tecla “#”), el muñeco trabaja según la configuración ingresada por el usuario: Velocidad máxima (configurado con tecla “A”, ingresar un valor al leer “VLim” en la pantalla), Cantidad máxima de monedas (configurado con tecla “B”, ingresar un valor al leer “CLim” en la pantalla y luego pulsar “#” para confirmar) y tipo de moneda (“C” para 10 centavos y “D” para dólar).

En caso de llegar al límite establecido, el proyecto deja de aceptar nuevos valores/cambios hasta que los parámetros (monedas) den un resultado dentro del límite o se cambie a otro modo (para volver al modo automático, pulsar la tecla “*”).

El diseño y materiales del muñeco, balanza y caja del proyecto son de libre elección

Proyecto construido:

El proyecto ya armado y en operación debería tener un aspecto similar a este:



Código:

Para el correcto funcionamiento del proyecto, se considera el siguiente código:

```
int IN1 = 6;
int IN2 = 7;
int ENA = 5;
int VELOCIDAD;

//MANEJADOR DE MOTOR

#include <HX711.h>

#define DT 2
#define SCK 3

//MANEJADOR DE CELDA DE CARGA

HX711 celda;

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
//MANEJADOR DE LCD

#include <TM1637Display.h>

#define CLK 14
#define DIO 15

TM1637Display display(CLK, DIO);
```

```

//MANEJADOR DE HT16 4SEG

#include <Keypad.h>

//Config del teclado

const byte filas = 4; //4 filas

const byte columnas = 4; //4 columnas

// Definir el layout de las teclas

char teclas[filas][columnas] = {

{'1','2','3','A'}, {'4','5','6','B'},

{'7','8','9','C'}, {'*','0','#','D'}

};

// Pines de filas y columnas

byte pinesFilas[filas] = {13,12,11,10};

// conecta a pines de filas

byte pinesColumnas[columnas] = {9,8,4,16};

// conecta a pines de columnas

//Crear objeto de la clase Keypad

Keypad teclado = Keypad(makeKeymap(teclas), pinesFilas,
pinesColumnas, filas, columnas);

//MANEJADOR DE KEYPAD

const uint8_t modo[] = {

SEG_A | SEG_C | SEG_E , //M

SEG_D | SEG_E | SEG_G | SEG_C , //O

```

```
SEG_D | SEG_E | SEG_G | SEG_C | SEG_B, //d
SEG_D | SEG_E | SEG_G | SEG_C }; //o

//Texto unico para mostrar en display de 4 segmentos

void setup() {
    Wire.begin(); // Para el cableado
    //Inicializar el LCD
    lcd.begin(16,2);

    lcd.backlight(); //Enciende la retroiluminación del LCD

    //Escribe texto en la primera linea del LCD
    lcd.setCursor(0,0); //A donde apunto, posicion en el LCD
    lcd.print("- - - SOMBRA");
    delay(1000);

    lcd.setCursor(0,1); //A donde apunto, posicion en el LCD
    lcd.print("ESPÍN - - - ");
    delay(1000);

    display.setBrightness(1);
    display.clear();

    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(ENA, OUTPUT);

    pinMode(13, OUTPUT);
    Serial.begin(9600);
```

```

    Serial.println("Balanza con celda de carga: ");

    celda.begin(DT, SCK);

    celda.setScale(926.f); //para calibrar con una moneda de
5g (5ctvs)

    celda.tare(); //calibrar la balanza

    Serial.println(celda.get_units(30)); //Valor obtenido en
gramos, el 30 es el promedio de 30 lecturas

    lcd.clear(); //Se usa para limpiar la pantalla
    lcd.setCursor(0,0); //A donde apunto, posicion en el LCD
    lcd.print("Esperando...");
    display.setSegments(modo, 4, 0);
    delay(1000);

}

/*
- Consideraciones -

```

El código debe permitir dos modos; manual y auto

Manual: Config la velocidad máxima de la rueda giratoria, la cantidad máxima de monedas y el tipo de moneda.

Auto: Gira de acuerdo a la cantidad de dinero / peso.

El código debe mostrar la siguiente info:

Velocidad actual: Niveles que van del 0 (sin movimiento) a 6
(máxima velocidad).

[LCD] Si se llega al máx. mostrar máx

Cantidad de monedas colocadas: Cuántas monedas hay en la plataforma, si se registra un peso mayor a 2.2 gramos, se sabe

que la cantidad aumentó, independientemente del tipo de moneda.

[4-SEG] Si se llega al máx. mostrar máx

Valor en dólares del dinero depositado: \$0.00 según la cantidad de monedas.

Para el modo manual esto es sencillo porque cada moneda tiene el mismo valor.

En el modo auto, dependiendo del aumento del peso registrado, se asigna un valor (1.6g a 2.5g == \$0.1), (5.3g a 8.4g == \$1.0).

Se debe esperar 5 segundos luego de asignar una nueva moneda contada para evitar rebotes y para considerar un peso nuevo como

moneda nueva, la diferencia debe ser la descrita arriba.

[LCD]

Cada segundo se debe mostrar esta info.

El teclado matricial tiene la siguiente configuración:

- Números del 0 al 9.
 - Botón A: Config Velocidad máxima para modo manual.
 - Botón B: Config Cantidad máxima de monedas para modo manual.
 - Botón C: Usar monedas de 10ctvs.
 - Botón D: Usar monedas de dólar.
 - Botón #: Activar modo manual.
 - Botón *: Activar modo auto.
- */

```
const uint8_t AUTO[] = {  
    SEG_A | SEG_B | SEG_C | SEG_E | SEG_F | SEG_G, //A  
    SEG_E | SEG_C | SEG_D, //U  
    SEG_D | SEG_E | SEG_F | SEG_G, //t  
    SEG_D | SEG_E | SEG_G | SEG_C }; //O
```

```
const uint8_t manu[] = {  
  
SEG_A | SEG_C | SEG_E , //M  
  
SEG_A | SEG_B | SEG_C | SEG_E | SEG_F | SEG_G, //a  
  
SEG_C | SEG_G | SEG_E , //n  
  
SEG_E | SEG_C | SEG_D }; //u
```

```
const uint8_t maxv[] = {  
  
SEG_D | SEG_F | SEG_E, //L  
  
SEG_F | SEG_E, //i  
  
SEG_A | SEG_C | SEG_E , //m  
  
SEG_B | SEG_D | SEG_F | SEG_C | SEG_E }; //v
```

```
const uint8_t maxc[] = {  
  
SEG_D | SEG_F | SEG_E, //L  
  
SEG_F | SEG_E, //i  
  
SEG_A | SEG_C | SEG_E , //m  
  
SEG_A | SEG_D | SEG_E | SEG_F }; //C
```

```
const uint8_t ctvs[] = {  
  
SEG_D | SEG_E | SEG_G , //c  
  
SEG_D | SEG_E | SEG_F | SEG_G , //t  
  
SEG_B | SEG_D | SEG_F | SEG_C | SEG_E , //v  
  
SEG_A | SEG_C | SEG_D | SEG_F | SEG_G }; //s
```

```
const uint8_t dolr[] = {  
  
SEG_D | SEG_E | SEG_G | SEG_C | SEG_B, //d  
  
SEG_D | SEG_E | SEG_G | SEG_C , //o  
  
SEG_D | SEG_F | SEG_E, //L
```

```

SEG_G | SEG_E }; //o

bool manual = false; //Indica el modo, si está en true es
manual, si está en false es automático

int velmax = 0; // [Va del 0 al 6]

int cantmax = 0;

bool dolar = false; //Indica el tipo de moneda, si está en
true es dólares, si está en false es monedas de 10ctvs.

float anteriorGramaje = 0; // Almacena el gramaje anterior

int totalmonedas = 0; //contador de monedas

float valor = 0; //contador de valor

int nivel = 0; //Nivel de velocidad

void loop() {

delay(250);

char tecla = teclado.getKey(); //Obtener una tecla del
keypad.

//Nota: Sería bueno alternar la lógica para que una
pulsación del keypad siempre interrumpa el programa

//En el caso de este código, se debe presionar el botón
del keypad dentro de una ventada de tiempo específica para que se
registre el input.

delay(250);

if(tecla){

if (tecla == '*'){

//MODO AUTO

manual = false;

display.setSegments(AUTO,4,0);

```

```

    delay(500);

}

if (tecla == '#') {
    //MODO MANUAL

    manual = true;

    display.setSegments(manu, 4, 0);

    delay(500);

}

if ((tecla == 'A') && (manual == true)) {
    //CONF VEL

    display.setSegments(maxv, 4, 0);

    delay(100);

    CONFIGURARVEL();

    display.clear();

    display.showNumberDec(velmax, false); // Expect:
_301, extraido del sketch de ejemplo del módulo TM1637Display

    delay(500);

}

if ((tecla == 'B') && (manual == true)) {
    //CONF CANT

    display.setSegments(maxc, 4, 0);

    delay(100);

    CONFIGURARCANT();

    display.clear();

    display.showNumberDec(cantmax, false); // Expect:
_301, extraido del sketch de ejemplo del módulo TM1637Display

    delay(500);
}

```

```

    }

    if ((tecla == 'C') && (manual == true)) {

        //USAR 10CTVS

        dolar = false;

        display.setSegments(ctvs,4,0);

        delay(500);

    }

    if ((tecla == 'D') && (manual == true)) {

        //USAR DÓLARES

        dolar = true;

        display.setSegments(dolr,4,0);

        delay(500);

    }

}else{

    if(manual == false){ //AUTOMATICO

        float gramaje; //almacena el peso actual

        float gramuca = celda.get_units(30); // Esta variable
        sirve de validación / evitar rebotes.

        //Antes de registrar un valor en gramaje, gramuca asegura
        que sea un peso significativo, esto evita que en una baliza

        //Sensible, como la de espumafón, se inserten valores no
        deseados al mover ligeramente la plataforma.

        //Parámetro 30 que funciona similar a un delay, pero para
        el peso.

        if (gramuca <= 1)

```

```
gramaje = 0; //Como gramaje no se actualiza a menos que entre en la validación, se hace una actualización excepción aquí para cuando no hay peso.
```

```
if (gramuca >= 1.2) // Se lee solo si el peso excede 2.4g. Nota: El valor de gramuca nunca se usa en el código.
```

```
gramaje = celda.get_units(10); //Valor obtenido en gramos, el 10 es el promedio de 10 lecturas
```

```
Serial.println("Valor (gramos): ");
```

```
Serial.println(gramaje, 1);
```

```
delay(500);
```

```
VELOCIDAD = ((gramaje / 5) * 26) + 125; // 5g para monedas de 5ctvs.
```

```
//El número 5 es arbitrario, se refiere a la unidad mínima de gramos para subir el nivel de velocidad.
```

```
//Como el motor de aficionado de dos lados solo acepta valores analógicos de 149~ a 255, lo dividi en 6 niveles (26 c/u).
```

```
//La operación gramaje/unidad mínima de gramos devuelve el nivel de velocidad, que se multiplica por 26 para "traducir" el valor de nivel a un valor analógico.
```

```
// + 125 es un simple offset, permite que el primer nivel comience en un valor "151", el segundo en un valor "177" y así sucesivamente.
```

```
Serial.println(VELOCIDAD);
```

```
nivel = (VELOCIDAD - 125) / 26; //Reobtenemos el nivel con la operación opuesta, también es posible obtenerlo directamente de (gramaje / 5).
```

```

if(VELOCIDAD < 150) {

    VELOCIDAD = 0; nivel = 0;} //Como los valores menores a
149~ no tienen suficiente poder para el motor, se mapea a 0 para
evitar consumo de batería y el zumbido.

if(VELOCIDAD > 255) {

    VELOCIDAD = 255; nivel = 6;} //Todo peso mayor al nivel 6
ya no aumenta la velocidad.

//VALORES DESDE 149 (APENAS FUNCIONAL) A 255 (MÁX VELOCIDAD),
es decir, 156 valores posibles

//156 / 6 = 26, es decir son 6 niveles de rotación de 26
valores c/u.

analogWrite(ENA, VELOCIDAD);

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

// Agrega monedas al contador si cumplen con la
condición, también las resta si es el caso opuesto.

if (gramaje - anteriorGramaje >= 2.2) {

    totalmonedas += 1;

}

else if (anteriorGramaje - gramaje >= 2.2) {

    totalmonedas -= 1;

}

if(totalmonedas < 0) //Evita valores negativos no
deseados en algunas ocasiones

totalmonedas = 0;

// Añade valor basado en un pequeño intervalo en gramaje

```

```

        if (gramaje - anteriorGramaje >= 1.7 && gramaje -
anteriorGramaje <= 2.6) {

            valor += 0.1; // Aumenta 0.1 si gramaje aumenta
entre 1.7 y 2.6 gramos

        } else if (anteriorGramaje - gramaje >= 0.7 &&
anteriorGramaje - gramaje <= 2.6) {

            valor -= 0.1; // Resta 0.1 si gramaje disminuye
entre 1.7 y 2.6 gramos

        }

        if (gramaje - anteriorGramaje >= 5.3 && gramaje -
anteriorGramaje <= 8.4) {

            valor += 1; //Aumenta 1 si gramaje aumenta entre 5.3
y 8.4 gramos

        } else if (anteriorGramaje - gramaje >= 5.3 &&
anteriorGramaje - gramaje <= 8.4) {

            valor -= 1; // Resta 1 si gramaje disminuye entre
5.3 y 8.4 gramos

        }

    }

/*

```

Para agregar más valores de monedas:

Define un intervalo de peso al rededor del peso

Ejemplo: Una moneda de 5ctvs pesa 5g. Define un intervalo entre 4.5 y 5.6 por ejemplo. Evita que toque los intervalos de peso de otras monedas

```

        if (gramaje - anteriorGramaje >= 4.5 && gramaje -
anteriorGramaje <= 5.6) {

            valor += 0.5 (de 5ctvs); //Aumenta 0.5 si gramaje
aumenta entre 4.5 y 5.6 gramos

```

```

        } else if (anteriorGramaje - gramaje >= 4.5 &&
anteriorGramaje - gramaje <= 5.6) {

            valor -= 0.5; // Resta 0.5 si gramaje disminuye
entre 4.5 y 5.6 gramos

        }

    */

if(valor < 0) //Evita valores negativos
valor = 0;

// Imprime el total de monedas y valor
Serial.print("Total monedas: ");
Serial.println(totalmonedas);
Serial.print("Valor: ");

Serial.println(valor, 1); // Imprime el valor con una
precisión de una decima.

display.clear();

display.showNumberDec(totalmonedas, false); // Expect:
_301

// Store the current gramaje as previous for the next
comparison

anteriorGramaje = gramaje;

//Mostrar nivel de velocidad

lcd.clear();

lcd.setCursor(0,0); //A donde apunto, posicion en el LCD

lcd.print("Velocidad: ");

if (nivel == 6){

    lcd.print("MAX");
}

```

```

}else{

    lcd.print(nivel);

}

lcd.setCursor(0,1); //A donde apunto, posicion en el LCD

lcd.print("Valor: ");

lcd.print(valor);

//digitalWrite(ENA, LOW);

celda.power_down(); //apagar hx711

delay(200);

celda.power_up();

}

if(manual == true){           ///MANUAAAAAAAL

float gramaje;

int unidad;

if(dolar == true){ //unidad minima de gramo para los niveles basado en dolares

    unidad = 8;

}

if(dolar == false){ //unidad minima de gramo para los niveles basado en 10 ctvs

    unidad = 2;

}

float gramuca = celda.get_units(30);

if (gramuca <= 1)

gramaje = 0;

```

```

if (gramuca >= 1.2)

gramaje = celda.get_units(10);

Serial.println("Valor MANUAL (gramos): ");
Serial.println(gramaje, 1);

delay(500);

VELOCIDAD = ((gramaje / unidad) * 26) + 125; // 5g para
monedas de 5ctvs

Serial.println(VELOCIDAD);

nivel = (VELOCIDAD - 125) / 26;

if(VELOCIDAD < 150) {

VELOCIDAD = 0; nivel = 0; totalmonedas = 0; valor = 0; }

if(VELOCIDAD > 255) {

VELOCIDAD = 255; nivel = 6; }

//VALORES DESDE 149 (APENAS FUNCIONAL) A 255 (MÁX VELOCIDAD),
es decir, 156 valores posibles

//156 / 6 = 26, es decir son 6 niveles de rotación de 26
valores c/u.

analogWrite(ENA, VELOCIDAD);

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

if (totalmonedas >= cantmax || nivel >= velmax) {

```

```

        // Si se llega a algún límite definido en el modo
manual, no se ingresa al proceso.

    Serial.println("Max lim!");

    lcd.setCursor(0,0); //A donde apunto, posicion en el
LCD

    lcd.print("-> Lim alcanzado");

    delay(500);

    return; // Reinicia el loop.

}

//Aumento el contador

if (gramaje - anteriorGramaje >= 2.2) {

    totalmonedas += 1;

}

else if (anteriorGramaje - gramaje >= 2.2) {

    totalmonedas -= 1;

}

if(totalmonedas < 0)

totalmonedas = 0;

if (dolar == true){ //Ya no se aumenta el valor en base
al peso porque se sabe que es uno fijo, asi que solo se multiplica
valor = totalmonedas * 1;

}

if (dolar == false){ //Lo mismo
valor = totalmonedas * 0.1;

}

```

```

if(valor < 0)

valor = 0;

Serial.print("Total monedas: ");
Serial.println(totalmonedas);

Serial.print("Valor: ");
Serial.println(valor, 1);

display.clear();

display.showNumberDec(totalmonedas, false);

//Esta validación en realidad nunca surge efecto, pero evita
que se actualice el gramaje si se llega a un límite

if ((nivel != velmax) && (totalmonedas != cantmax))
anteriorGramaje = gramaje;

//Mostrar nivel de velocidad

lcd.clear();

lcd.setCursor(0,0); //A donde apunto, posicion en el LCD
lcd.print("Velocidad: ");

if (nivel == velmax) {

lcd.print("MAX");

} else {

lcd.print(nivel);

}

lcd.setCursor(0,1); //A donde apunto, posicion en el LCD
lcd.print("Valor: ");

lcd.print(valor);

//digitalWrite(ENA, LOW);

```

```

        celda.power_down(); //apagar hx711
        delay(200);
        celda.power_up();
    }
}

String acumulador;

void CONFIGURARVEL() {
    char tecla;
    acumulador = "";
    velmax = 0;
    do {
        tecla = teclado.getKey();
        // Solo se actualiza si se pulsa una tecla
        if (tecla != NO_KEY) {
            acumulador = tecla;
            velmax = acumulador.toInt();
        }
    } while (velmax < 1 || velmax > 6);
}

void CONFIGURARCANT() {
    char tecla;
    bool salir = false;
    acumulador = "";
}

```

```

do {

    tecla = teclado.getKey();

    // Solo actualiza si se pulsa una tecla válida

    if (tecla != NO_KEY && tecla >= '0' && tecla <= '9') {

        acumulador += tecla; // Se hace un string del número

    }

    if (tecla == '#') {

        salir = true;

    }

    delay(50);

} while (salir == false);

cantmax = acumulador.toInt();

}

```

Nota: Observar con atención la operación del código, puede que existan diferencias en el funcionamiento final según la implementación que se le dé. (Por ejemplo, una calibración distinta de la celda de carga, uso de otros materiales, etc).

Referencias:

Celda de carga, calibración y módulo HX711:

<https://www.youtube.com/watch?v=Z01gjTLE5eE>

Módulo L298N y motor DC:

<https://www.youtube.com/watch?v=63aitq3KTaI>

Ejemplo de muñeco funcional:

https://www.pinterest.com/pin/10414642881425231/sent/?invite_code=1daa54d570d64cc3b30695be87ad77e8&sender=906912581120279533&sfo=1