TREN ARDUINO: PRÓXIMA PARADA, ¡INNOVACIÓN!

Laura Canchano Vargas
Nerea Díaz Rovira
Lorena Triviño Plaza

ÍNDICE

ÍNDICE..... 2
INTRODUCCIÓN..... 3
DESCRIPCIÓN DEL PROYECTO..... 3
FASES DE IMPLEMENTACIÓN..... 3
1. Planificación inicial..... 3
2. Construcción física..... 4
3. Montaje Electrónico..... 4
MATERIALES Y COSTO..... 6
PROGRAMACIÓN Y AUTOMATIZACIÓN..... 7
CÓDIGO..... 7
CASOS DE USO..... 14
PROBLEMAS Y SOLUCIONES..... 14
CONCLUSIÓN..... 16

INTRODUCCIÓN

En el proyecto llevado a cabo, desarrollamos una maqueta funcional que constituye la simulación de una ruta ferroviaria que cuenta con 2 paradas y melodías para 3 casos de uso diferentes. El sistema está basado en una placa Arduino y combina tanto componentes electrónicos como decorativos para crear un entorno urbano. En cada estación el tren deberá parar un tiempo considerable, permitiendo la subida y bajada de pasajeros, así como el cruce de estos hacia el otro lado. En estas paradas se activan funciones específicas como melodías, detención automática y simulación de iluminación nocturna. Además el sistema incluye una alarma en caso de detección de fuego.

El objetivo principal ha sido integrar los elementos arquitectónicos y tecnológicos, combinando la automatización con la programación básica para lograr la correcta simulación de un sistema ferroviario inteligente,

DESCRIPCIÓN DEL PROYECTO

La maqueta está compuesta por:

- Un tren de juguete monitorizado que se desplaza de forma autónoma.
- Dos estaciones con plataformas y barreras móviles.
- Diversas casas y comercios que representan el entorno urbano.
- Farolas automáticas que se encienden al anochecer, controladas por un sensor LDR.
- Elementos de decoración como árboles, un lago, caminos…

FASES DE IMPLEMENTACIÓN

1. Planificación inicial

El proyecto comenzó con la compra del tren eléctrico y sus vías. A partir de sus dimensiones exactas, realizamos un esbozo de la maqueta ajustado para organizar la ubicación de los elementos sobre la base. De igual manera elaboramos una lista con las acciones que queríamos que el tren realizara para comprar los elementos electrónicos necesarios para ello.

2. Construcción física

Hemos empleado diversos materiales:

- **Base:** hemos usado una tabla de madera contrachapada forrada con papel y césped sintético.
- **Estación principal:** construida con cartón y forrada.
- **Fijación de vías:** las vías del tren se encuentran fijadas a la base mediante tornillos para evitar desplazamientos durante el movimiento del tren.
- **Fijación de las estaciones:** las estaciones del tren las hemos fijado mediante una pistola de silicona.
- **Decoración:** se añadieron casas, árboles, caminos, un lago…para ambientar la maqueta.

3. Montaje Electrónico

Hemos instalado y empleado una diversidad de componentes electrónicos:

- **Placa Arduino UNO**
  - Función principal: Actúa como el cerebro de toda la maqueta. Es un microcontrolador programable que recibirá información de los sensores y, basándose en el código que le carguemos, tomará decisiones y enviará señales a los actuadores (como el motor del tren o las luces).
  - En el proyecto: La Arduino UNO será responsable de leer los datos de los sensores, interpretar las órdenes para el movimiento del tren, controlar la velocidad a través del driver de motor, activar el zumbador para emitir sonidos.

- **Protoboard**
  - Función principal: Es una placa con orificios y conexiones internas que permite ensamblar y probar circuitos electrónicos de forma temporal sin necesidad de soldar.
  - En el proyecto: La protoboard será fundamental durante la fase de desarrollo para conectar el Arduino UNO con los sensores, el driver y el zumbador.

- **Zumbador Pasivo**
  - Función principal: Es un componente que genera sonido cuando se le aplica una señal eléctrica oscilante. A diferencia de un zumbador activo (que solo emite un tono fijo), un zumbador pasivo requiere que el microcontrolador (Arduino) genere la frecuencia del tono, permitiendo reproducir diferentes notas y melodías simples.
  - En el proyecto: Se utilizará para emitir diferentes melodías o tonos que indiquen el estado del tren.

- **Sensor LDR**
    - Función principal: Es una resistencia cuyo valor varía en función de la cantidad de luz que incide sobre ella. A más luz, menor resistencia, y viceversa.
    - En el proyecto: El sensor LDR se utilizará para detectar los niveles de luminosidad.

- **Driver de Motor**
    - Función principal: Es un circuito integrado o módulo diseñado para controlar la velocidad y dirección de motores de corriente continua (DC), como el que probablemente utiliza el tren eléctrico. Los microcontroladores como Arduino no pueden suministrar directamente la corriente que necesita un motor, ni manejar los voltajes requeridos. El driver actúa como un intermediario, recibiendo señales de control de bajo voltaje del Arduino y entregando la potencia necesaria al motor.
    - En el proyecto: Regulará la alimentación del motor del tren. El Arduino enviará señales PWM al driver para controlar la velocidad del tren de forma suave y precisa.

- **Transistor MOSFET**
    - Función principal: Es un tipo de transistor que actúa como un interruptor controlado electrónicamente. Puede manejar corrientes y voltajes significativamente mayores que los que puede proporcionar directamente un pin de salida de Arduino, y lo hace con muy poca pérdida de potencia.
    - En el proyecto: Si queremos controlar varias farolas LED, el MOSFET es ideal. El Arduino enviará una señal de bajo consumo al "gate" (puerta) del MOSFET, y este permitirá o bloqueará el paso de una corriente mayor desde una fuente de alimentación externa hacia las farolas LED.

- **Sensor Ultrasónico**
    - Función principal: Este sensor mide distancias emitiendo pulsos de sonido ultrasónico y midiendo el tiempo que tardan en rebotar en un objeto y regresar. Conociendo la velocidad del sonido, puede calcular la distancia al objeto.
    - En el proyecto: Montado en la parte delantera del tren puede detectar si hay un objeto o otro tren bloqueando el camino, permitiendo al Arduino detener o reducir la velocidad del tren para evitar colisiones.

Una vez finalizada la construcción física de la maqueta, iniciamos la fase electrónica. Para ello primero identificamos los 3 casos de uso que debía cubrir nuestro sistema de sonido: emitir una melodía mientras realiza el recorrido, emitir otra al parar por la detección de algún objeto y por último, en caso de detectar fuego, emitir un alarma de incendios.

Tras ello comenzamos con el diseño y montaje de los circuitos necesarios para el funcionamiento de zumbador pasivo.

De forma paralela se montó el circuito encargado de gestionar la luminosidad de la ciudad. Este se compone de farolas LED y un sensor de luz LDR. Dicho sensor está empleado para detectar niveles bajos de luminosidad, de tal manera que cuando se simula oscuridad las farolas se encienden automáticamente. Para reducir el uso de pines de la placa empleamos un transistor MOSFET que nos permitió el encendido de las farolas usando un solo pin.

Por otro lado, realizamos el montaje del sistema de control del motor con el driver TB6612FNG. El tren contaba con un motor incorporado, pero tuvimos que desmontarlo por completo y soldar cables más largos y fuertes para poder emplearlo en el driver.

MATERIALES Y COSTO

MATERIAL	PRECIO/EUROS
Driver TB6612FNG	9
Componentes electrónicos (Clemas, cables…)	7,60
Elementos decorativos (árboles, casas, piedras…)	19,33
Tren con vías + estación	16
Kit con Soldador	Regalo
Transistor MOSFET	3,90
Barras de Silicona	2,20
Total estimado	58,03

PROGRAMACIÓN Y AUTOMATIZACIÓN.

El código fue desarrollado en el entorno de programación Arduino IDE. La lógica de la programación se dividió en los siguientes módulos funcionales:

- **Control del tren:** a través del driver de motor regulamos la velocidad pudiendo circular el tren a menor velocidad y detenerse en las estaciones.
- **Gestión de sonido:** según el estado del tren el zumbador pasivo emite diferentes melodías que hemos almacenado como frecuencias.
- **Control de farolas:** el LDR detecta la baja luminosidad y enciende las luces al anochecer.
- **Detención automática:** el tren se detiene automáticamente en las estaciones correspondientes tras pasar un tiempo determinado.

CÓDIGO

```
#include <pitches.h>

// --- Pines Tren y Sensores ---
const int pinAIN1 = 6;
const int pinAIN2 = 8;
const int trigPin = 9;
const int echoPin = 10;
const int pinSensorLlamaAO = A0;
const int buzzerPin = 12;

// --- Variables motor y sensores ---
int motorSpeed = 150;
int stopDistance = 15;
const int UMBRAL_LLAMA = 900;
int valorLlama;
float currentDistance = 100.0;


enum TrainState {
  RUNNING,
  STOPPED_TIMER,
  STOPPED_OBSTACLE,
  STOPPED_FIRE
};
TrainState currentState = RUNNING;
TrainState previousStateBeforeObstacle = RUNNING;
TrainState previousStateBeforeFire = RUNNING;
unsigned long lastStateChangeTime = 0;
```

```
unsigned long timePaused = 0;
unsigned long timeFirePaused = 0;
const unsigned long runDuration = 5000;
const unsigned long stopDuration = 10000;

// --- Flags de estado y Melodías ---
int melodiaIndex = 0;
int harryIndex = 0;

// --- Simpsons ---
int melody[] = {
  NOTE_C4, NOTE_E4, NOTE_FS4, REST, NOTE_A4,
  NOTE_G4, NOTE_E4, NOTE_C4, NOTE_A3,
  NOTE_FS3, NOTE_FS3, NOTE_FS3, NOTE_G3, REST,
  NOTE_FS3, NOTE_FS3, NOTE_FS3, NOTE_G3, NOTE_AS3,
  NOTE_B3, REST
};
int durations[] = {
  2, 4, 4, 32, 8,
  2, 4, 4, 8,
  8, 8, 8, 4, 2,
  8, 8, 8, 4, 2,
  2, 2
};

// --- Harry Potter ---
int harryMelody[] = {
  REST, NOTE_D4, NOTE_G4, NOTE_AS4, NOTE_A4, NOTE_G4, NOTE_D5, NOTE_C5,
  NOTE_A4, NOTE_G4, NOTE_AS4, NOTE_A4, NOTE_F4, NOTE_GS4, NOTE_D4,
  NOTE_D4, NOTE_G4, NOTE_AS4, NOTE_A4, NOTE_G4, NOTE_D5, NOTE_F5,
NOTE_E5,
  NOTE_DS5, NOTE_B4, NOTE_DS5, NOTE_D5, NOTE_CS5, NOTE_CS4, NOTE_B4,
  NOTE_G4, NOTE_AS4, NOTE_D5, NOTE_AS4, NOTE_D5, NOTE_CS5, NOTE_A4,
NOTE_AS4,
  NOTE_D5, NOTE_CS5, NOTE_CS4, NOTE_D4
};
int harryDurations[] = {
  2, 4, 4, 8, 4, 4, 4, 8,
  2, 4, 8, 4, 4, 4, 4,
  4, 4, 8, 4, 2, 4, 4, 8,
  4, 8, 8, 4, 4, 4, 2,
  4, 4, 8, 4, 8, 4, 4, 8,
  4, 4, 4, 2
```

```
};

int harryTempo = 120;
int harryWholenote = (60000 * 4) / harryTempo;


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

  // Configuración pines Tren y Sensores
  pinMode(pinAIN1, OUTPUT);
  pinMode(pinAIN2, OUTPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzerPin, OUTPUT);

  // Estado inicial: Tren circulando
  currentState = RUNNING;
  previousStateBeforeObstacle = RUNNING;
  previousStateBeforeFire = RUNNING;
  digitalWrite(pinAIN2, LOW);
  analogWrite(pinAIN1, motorSpeed);
  lastStateChangeTime = millis();
  harryIndex = 0;
  noTone(buzzerPin);

  Serial.println("Sistema Iniciado: Ciclo temporizado con
pausa/reanudación por obstáculo y fuego.");
  Serial.println("-> Estado inicial: RUNNING (5s)");
}

void loop() {
  unsigned long currentTime = millis();

  valorLlama = analogRead(pinSensorLlamaAO);
  Serial.print("Valor Llama Actual: ");
  Serial.println(valorLlama);

  if (valorLlama < UMBRAL_LLAMA) {
    // ---Detección de fuego ---
    if (currentState != STOPPED_FIRE) {
      Serial.println("¡FUEGO DETECTADO! Pausando ciclo.");
      previousStateBeforeFire = currentState;
```

```
    timeFirePaused = currentTime;
    currentState = STOPPED_FIRE;
    analogWrite(pinAIN1, 0);
    noTone(buzzerPin);
  }
  // Sonido de alarma
  tone(buzzerPin, 1000); delay(150);
  noTone(buzzerPin); delay(100);
  return;
  // --- Final de detección de fuego ---

} else if (currentState == STOPPED_FIRE) {
  // --- No fuego ---
  Serial.println("Fuego extinguido. Reanudando ciclo.");
  noTone(buzzerPin);

  unsigned long firePausedDuration = currentTime - timeFirePaused;
  lastStateChangeTime += firePausedDuration;

  currentState = previousStateBeforeFire;

  Serial.print("--> Reanudando estado: ");
  if (currentState == RUNNING) {
      Serial.println("RUNNING");
      digitalWrite(pinAIN2, LOW);
      analogWrite(pinAIN1, motorSpeed);
  } else if (currentState == STOPPED_TIMER) {
      Serial.println("STOPPED_TIMER");
  } else if (currentState == STOPPED_OBSTACLE) {
      Serial.println("STOPPED_OBSTACLE");
  }
}

// Sensor de Distancia (Solo si no hay fuego)
long duration;
digitalWrite(trigPin, LOW); delayMicroseconds(2);
digitalWrite(trigPin, HIGH); delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH, 30000);
if (duration > 0) {
    currentDistance = duration * 0.034 / 2;
} else {
    currentDistance = 100.0;
```

```
  }

  if (currentState != STOPPED_FIRE) {
      if (currentDistance >= 2 && currentDistance < stopDistance &&
currentState != STOPPED_OBSTACLE) {
          Serial.print(" L Obstáculo detectado a ");
Serial.print(currentDistance); Serial.println(" cm. Pausando ciclo.");
          previousStateBeforeObstacle = currentState;
          timePaused = currentTime;
          currentState = STOPPED_OBSTACLE;
          analogWrite(pinAIN1, 0);
          noTone(buzzerPin);
      }
  }

  // Apartado de Estados por los que pasa el tren
  switch (currentState) {
    case RUNNING:
      reproducirHarryPotter();
      if (currentTime - lastStateChangeTime >= runDuration) {
        Serial.println("-> Tiempo de marcha cumplido. Transición a
STOPPED_TIMER (10s)");
        currentState = STOPPED_TIMER;
        lastStateChangeTime = currentTime;
        analogWrite(pinAIN1, 0);
        noTone(buzzerPin);
        melodiaIndex = 0;
      }
      break;

    case STOPPED_TIMER:
      reproducirSimpsons();
      if (currentTime - lastStateChangeTime >= stopDuration) {
        Serial.println("-> Tiempo de parada cumplido. Transición a
RUNNING (5s)");
        currentState = RUNNING;
        lastStateChangeTime = currentTime;
        digitalWrite(pinAIN2, LOW);
        analogWrite(pinAIN1, motorSpeed);
        noTone(buzzerPin);
        harryIndex = 0;
      }
      break;
```

```
    case STOPPED_OBSTACLE:
      if (currentDistance >= stopDistance || currentDistance < 2) {
        Serial.println("-> Obstáculo retirado. Reanudando ciclo.");
        unsigned long pausedDuration = currentTime - timePaused;
        lastStateChangeTime += pausedDuration;
        currentState = previousStateBeforeObstacle;
        Serial.print("--> Reanudando estado: ");
        if (currentState == RUNNING) {
            Serial.println("RUNNING");
            digitalWrite(pinAIN2, LOW);
            analogWrite(pinAIN1, motorSpeed);
        } else {
            Serial.println("STOPPED_TIMER");

        }
         noTone(buzzerPin);
      }
      // Si el obstáculo sigue ahí, permanece en este estado.
      break;

    case STOPPED_FIRE:
      break;
  }
  delay(10);
}

// --- FUNCIONES DE MELODÍA ---
// Las funciones reproducirSimpsons y reproducirHarryPotter
void reproducirSimpsons() {
  if (currentState == STOPPED_TIMER) {
      int size = sizeof(durations) / sizeof(int);
      if (melodiaIndex < size) {
        int noteDuration = 1500 / durations[melodiaIndex];
        if (melody[melodiaIndex] != REST) {
          tone(buzzerPin, melody[melodiaIndex], noteDuration * 0.9);
        }
        delay(noteDuration);
        noTone(buzzerPin);
        melodiaIndex++;
      }
  } else {
```

```
      if (melodiaIndex > 0 && melodiaIndex < (sizeof(durations) /
sizeof(int))) {
         noTone(buzzerPin);
      }
  }
}

void reproducirHarryPotter() {
  if (currentState == RUNNING) {
      int totalNotes = sizeof(harryMelody) / sizeof(int);
       if (totalNotes != sizeof(harryDurations) / sizeof(int)) {
         Serial.println("ERROR: Desajuste Melodía/Duración HP");
         currentState = STOPPED_OBSTACLE;
         analogWrite(pinAIN1, 0);
         noTone(buzzerPin);
         return;
       }
      if (harryIndex < totalNotes) {
        int divider = harryDurations[harryIndex];
        if (divider == 0) divider = 1;
        int noteDuration = (divider > 0) ? (harryWholenote / divider) :
(harryWholenote / abs(divider)) * 1.5;
        if (harryMelody[harryIndex] != REST) {
          tone(buzzerPin, harryMelody[harryIndex], noteDuration *
0.90);
        }
        delay(noteDuration);
        noTone(buzzerPin);
        harryIndex++;
      } else {
        harryIndex = 0;
      }
  } else {
       if (harryIndex > 0 && harryIndex < (sizeof(harryMelody) /
sizeof(int))) {
         noTone(buzzerPin);
      }
  }
}
```

CASOS DE USO

Funcionamiento del tren:

- **Sistema de Reproducción de Sonidos:** Emite sonidos realistas del tren para mejorar la experiencia. Se emiten tres tipos de melodías: Harry Potter para cuando está en circulación, Los Simpson para cuando está parado en las estaciones y la alarma de fuego para cuando se detecta la anomalía.
- **Sistema de Detección de Incendios:** Simula la detección de un incendio en la maqueta. Un sensor de llama detecta una anomalía y envía una señal al microcontrolador, que puede detener el tren y activar una alarma con sonido.
- **Sistema de Detección de Obstáculos:** Evita que el tren choque con objetos en la vía. Si algo está muy cerca, el microcontrolador detiene el tren.
- **Control de Movimiento del Tren:** Se controla la velocidad del motor del tren.
- **Control Temporizado entre estaciones:** Asegura que el tren se detenga en cada estación durante un tiempo específico antes de continuar su recorrido.
- **Iluminación automática:** Enciende automáticamente las luces de las farolas cuando se detecta un bajo nivel de luz.

PROBLEMAS Y SOLUCIONES.

1. **Driver incorrecto**

   Uno de nuestros principales problemas fue el motor del tren, pues necesitábamos programarlo para reducir su velocidad. Para ello compramos un driver, pero tras una tarde de pruebas y sin ningún resultado, nos dimos cuenta de que el elemento que habíamos comprado necesitaba más voltaje del proporcionado por la batería de nuestro tren. El driver requería 12V y el tren solo contaba con 3V. Tras ello tuvimos que comprar de nuevo el componente pero esta vez con el voltaje correcto.

2. **Falta de pines**

   Como queríamos poner iluminación empleando las farolas, había que conectarlas todas a la protoboard, pero al ser tantas nos vimos sin pines disponibles para el resto de cosas, por lo que compramos un transistor MOSFET para únicamente usar un pin para encender las farolas.

3. **Conexión de farolas con el arduino.**

   Como hemos mencionado a lo largo del documento, nuestra pequeña ciudad cuenta con iluminación nocturna que en principio estaba programada en arduino. Siguiendo el problema anterior, al ir conectadas nuestras farolas a la placa central que

controlaba el tren, ya no solo teníamos que hacer que girara la protoboard central con el arduino, sino también que lo hiciera la protoboard de las farolas. Siendo para nosotras imposible evitar el enredo de los cables, decidimos crear un circuito aparte, no programado, para las farolas alimentado por una batería externa. Por tanto las farolas cuentan con un funcionamiento independiente al del tren, pero de igual manera gracias al LDR pudiendo encenderse nuestra ciudad al detectar oscuridad.

4. **Circulación del tren.**

Otro de los problemas más importantes ha sido el circulamiento del tren con la protoboard y el arduino, pues siendo nuestro circuito circular no podíamos fijar las placas a ningún punto sin que el tren se enredara o se desenchufara.

En 1º lugar decidimos construir una "torre de control" en medio de la maqueta para situar las placas y que el tren pudiera dar la vuelta con los cables tirando hacia el centro. No obstante esta idea no dio resultado porque tras alargar los cables, estos se enredaban de manera constante, provocando que algunos de ellos se soltaran o que el tren se descarrilara.

El 2º intento fue pegar las placas a la parte superior del tren, llevando por tanto este toda la electrónica encima. Si bien es cierto que con ello los cables no se nos enredaron, supuso demasiado peso para nuestra locomotora, produciendo por tanto su descarrilamiento de manera constante de las vías.

Finalmente tras probar y pensar diversas medidas, llegamos a la conclusión de que la mejor opción era llevar nosotras en la mano ambas placas e ir dando la vuelta a la vez que el tren. Al intentar esto el tren dio vueltas completas al circuito cumpliendo con todas las funciones programadas.

5. **Velocidad del motor y cables que lo componen.**

Con el motor hemos tenido 2 problemas. Por un lado, a la hora de unirlo al driver para poder controlarlo con nuestro código, nos dimos cuenta de que los cables con los que venía eran muy cortos y muy finos. Decidimos unirlos con otros cables empleando clemas, pero al ser tan finos esto no dio resultado, por lo que finalmente terminamos soldando cables para ganar más longitud. Por otro lado, una vez puesto en marcha el motor, el tren iba a una velocidad muy rápida, lo que producía que se saliera del circuito y además le impedía realizar las paradas correspondientes. Para solucionar esto modificamos un poco el código, reduciendo la velocidad inicial del tren a 150, que es lo mínimo que necesita el motor para poder circular.

6. **Arranque tras la detección de un objeto**

Una de las funcionalidad que decidimos que el tren realizará, era frenar en caso de detectar un objeto y en principio parecía que funcionaba correctamente, pues al detectar algo el tren se paraba y se emitía la melodía correspondiente a dicho caso; sin embargo, tras eliminar el objeto, el tren no continuaba con su recorrido. La solución encontrada fue modificar un poco el código para que tras dicha parada de emergencia, el tren pudiera reanudar la circulación.

CONCLUSIÓN

Este proyecto nos ha permitido integrar los conocimientos de electrónica, programación y diseño a través de un entorno práctico y creativo. La maqueta no solo cumple con las funciones simuladas, sino que además es la representación de una pequeña ciudad inteligente, con la interacción entre la iluminación, el sonido y el movimiento. A lo largo del proceso hemos pasado por diversas dificultades técnicas, sobre todo en la parte hardware a la que no estamos tan acostumbradas. Esto nos ha obligado a investigar, adaptarnos y ser pacientes, siendo por tanto un proyecto enriquecedor que ha reforzado nuestras capacidades de organización, trabajo en equipo y resolución de problemas.