Autómata con Implementación de Gestos y Conexión IoT

Ingeniería de Software

DEV: Camacho Castelán Jose Manuel

DEV: Pichal Pío Jose Armando

SM: López Castillo Haziel

14 de diciembre de 2025

Índice

Visión del Proyecto

El Objetivo

El objetivo de este proyecto era la creación de un autómata con la capacidad de recibir gestos mediante un control remoto, el cual al recibir el comando pueda mandar una señal a dispositivos inteligentes y de esa forma ser un nodo funcional e interactivo para implementaciones de rutina adyacentes a las tecnologías del Internet de las Cosas (IoT).

La Visión

La visión de este proyecto es crear una interfaz interactiva donde se pueda aprender el funcionamiento de conexiones inteligentes entre dispositivos electrónicos, agregando dinamismo a las rutinas usando gestos en vez de la automatización total.

Los Usuarios

El usuario pensado para este producto sería cualquier persona que cuente con dispositivos inteligentes que puedan ser controlados por protocolos a través del Internet de las Cosas, con enfoque especial en usuarios con el interés en agregar dinamismo a las rutinas usando gestos en vez de total automatización. Los roles activos son el Usuario Operador (quien usa la Varita) y el Usuario Administrador (quien accede al Panel de Control).

Requerimientos del Sistema

Requerimientos Funcionales

El sistema debe reconocer gestos específicos tales como Lumos, Nox, Ascendio, Descendo, Avada Kedavra y Petrificus Totalus, clasificando cada comando para su ejecución correspondiente. El autómata debe proporcionar retroalimentación visual mediante pantalla OLED y auditiva mediante Buzzer de forma sincronizada tras la ejecución de un comando. Debe contar con un Panel de Control web accesible en red local que muestre logs y el estado del autómata en tiempo real.

Requerimientos No Funcionales

El sistema debe ser altamente disponible en la red local. La ejecución de la inteligencia artificial debe usar threading para evitar el bloqueo del servidor web. Debe ofrecer una interfaz visual e interactiva que sea accesible desde cualquier dispositivo conectado a la red local.

Cronograma Final de Actividades

El proyecto se completó a través de 6 Sprints, con 6 horas de trabajo por Sprint (3 horas por día durante 2 días).

Sprint 1 - Base Técnica: Se logró establecer el flujo inicial de datos del sensor MPU-6050 hacia la Raspberry Pi, creando la base para la recolección de información de movimientos.

Sprint 2 - Arquitectura de IA: Se pivotó la arquitectura hacia Flask y Python ejecutándose en la Raspberry Pi 4. Se diseñó la estructura de clases de inteligencia artificial para el procesamiento de gestos.

Sprint 3 - Hardware de Retroalimentación: Se completó la conexión física de la Raspberry Pi 4 con la pantalla OLED y el Buzzer, preparando los componentes de feedback del sistema.

Sprint 4 - Lógica IoT e IA: Los modelos de gestos para control IoT como Lumos y Ascendio fueron entrenados exitosamente. Se implementó la arquitectura de threading para procesamiento paralelo.

Sprint 5 - Panel Web y UI: Se implementó el Panel de Control web completo con sistema de login, registro de usuarios y visualización en tiempo real del estado del autómata.

Sprint 6 - Integración Final: Se completó el feedback sincronizado entre OLED y

Buzzer. Los comandos de estado final como Avada Kedavra y Petrificus Totalus fueron implementados y probados.

Alcance Final del Producto

El producto final es un autómata con un sistema de detección de gestos mediante aprendizaje automático, un backend en Flask y un Panel de Control web completamente funcional. La limitación final del proyecto es que la conexión y el control físico de los servomotores del autómata para su movimiento dinámico se pospusieron como trabajo futuro debido a restricciones de tiempo y priorización de funcionalidades core.

Costo Total del Desarrollo

El costo se desglosa en la suma del costo de hardware y el costo de tiempo del equipo de desarrollo, utilizando el salario promedio de un desarrollador de software Junior en Veracruz, México.

Costo de Hardware

Componente	**Costo (MXN)**
Raspberry Pi 4 Model B (4GB RAM)	$950.00
ESP32 Dev Board	$170.00
Sensor MPU-6050 (Giroscopio/Acelerómetro)	$85.00
Pantalla OLED SSD1306 (128x64)	$140.00
Buzzer Pasivo	$35.00
Servomotores (x2)	$170.00
Cables, resistencias y material de soldadura	$100.00
Subtotal Hardware	**$1,650.00**

Costo de Tiempo (Horas-Hombre)

Según datos del mercado laboral de Veracruz, el salario promedio de un desarrollador de software Junior es de aproximadamente $12,000 MXN mensuales, lo que equivale a $75 MXN por hora considerando jornadas de 160 horas al mes.

Concepto	**Horas**	**Tarifa/Hora**	**Costo Total**
Horas por desarrollador	36 hrs	$75.00	$2,700.00
Total del equipo (3 personas)	108 hrs	$75.00	$8,100.00
Costo Total de Desarrollo (HH)			**$8,100.00**

Costo Total del Proyecto

Concepto	**Monto (MXN)**
Costo de Hardware	$1,650.00
Costo de Horas-Hombre	$8,100.00
COSTO TOTAL DEL PROYECTO	**$9,750.00**

Arquitectura del Sistema

Diagrama General

El sistema sigue una arquitectura centralizada en la Raspberry Pi 4. El backend en Flask actúa como el coordinador central que recibe datos de la Varita mediante comunicación serial y distribuye comandos al Feedback Manager (que controla OLED y Buzzer) y al MQTTPublisher para la comunicación con dispositivos IoT.

Tecnologías Utilizadas

Hardware Principal:

- Raspberry Pi 4 Model B (4GB RAM)
- ESP32 Dev Board
- Sensor MPU-6050
- Pantalla OLED SSD1306
- Buzzer Pasivo

Software y Frameworks:

- Python 3.9
- Flask (servidor web)
- Scikit-learn (modelos de aprendizaje automático)
- Threading (procesamiento paralelo)
- SQLite (base de datos)

Diseño de la Base de Datos

El modelo de datos se basa en SQLite por su simplicidad y bajo overhead para este tipo de aplicación embebida.

Tabla Usuarios:

- id (Clave Primaria, Autoincremental)
- nombre_completo (Texto)
- usuario (Texto, Único)
- password (Texto, Hash)

Esta tabla almacena las credenciales de acceso al Panel de Control web, permitiendo múltiples usuarios con acceso seguro mediante autenticación.

Estructura de Módulos

Backend:

- app.py - Servidor Flask principal y enrutamiento
- Entrenamiento.py - Módulo de entrenamiento y clasificación de gestos
- Models/ - Directorio con modelos entrenados serializados

Frontend:

- templates/ - Vistas HTML del Panel Web
- static/ - Archivos CSS, JavaScript e imágenes

Hardware:

- Firmware ESP32 - Código para lectura del sensor MPU-6050
- Controllers/ - Clases de control de OLED y Buzzer

Interfaz de Usuario

Wireframes

Los wireframes del sistema incluyen dos vistas principales:

Vista de Login y Registro: Pantalla inicial donde el usuario ingresa sus credenciales o se registra por primera vez. Diseño minimalista con campos de texto para usuario y contraseña.

Dashboard Principal: Interfaz que muestra el estado del autómata en tiempo real, incluyendo el stream de datos del sensor (visualización gráfica del movimiento), historial de gestos detectados y logs del sistema.

Flujo de Pantallas

El flujo de navegación del usuario es el siguiente:

1. El usuario accede a la dirección IP de la Raspberry Pi desde cualquier dispositivo en la red local

2. Se presenta la pantalla de Login donde ingresa sus credenciales

3. Una vez autenticado, accede al Dashboard principal

4. Desde el Dashboard, el usuario puede iniciar la detección de gestos y monitorear el sistema en tiempo real

5. El sistema muestra feedback visual del estado de conexión y los comandos ejecutados

Estilo Visual y Decisiones de Diseño

El estilo visual es funcional y ligero, basado en la temática de magia y hechizos inspirada en el universo de Harry Potter. El diseño prioriza el rendimiento y la accesibilidad desde cualquier dispositivo en la red local, utilizando una paleta de colores oscuros para reducir la fatiga visual y resaltar los elementos importantes mediante contrastes.

Las decisiones de diseño se enfocaron en crear una interfaz intuitiva que no requiera capacitación previa, con iconografía clara y mensajes de estado descriptivos que guían al usuario en cada paso.

Bitácora de Cambios

Versiones del Sistema

Versión 1.0 (Fin de Sprint 2): Arquitectura de Python y Flask establecida como base del sistema. Comunicación serial entre ESP32 y Raspberry Pi funcional.

Versión 2.0 (Fin de Sprint 4): Lógica de inteligencia artificial y threading implementados y operativos. Modelos de gestos para control IoT entrenados y probados.

Versión 3.0 (Fin de Sprint 6): Producto final completado. Feedback sincronizado entre OLED y Buzzer funcional. Panel web totalmente operativo con sistema de autenticación.

Funciones Agregadas, Modificadas o Descartadas

Funciones Agregadas:

Sprint 4: Implementación de threading para la ejecución de la inteligencia artificial en segundo plano, evitando el bloqueo del servidor web durante el procesamiento de gestos.

Sprint 5: Implementación del sistema de login y registro de usuarios utilizando SQLite como base de datos local.

Sprint 6: Implementación de la clase de integración para sincronizar los mensajes visuales en OLED con los sonidos del Buzzer, creando una experiencia de feedback coherente.

Funciones Modificadas:

Sprint 2: Se modificó completamente la arquitectura de comunicación. Inicialmente se planeaba que la ESP32 funcionara como servidor web, pero se cambió a una arquitectura donde la Raspberry Pi con Flask actúa como servidor central y la ESP32 solo envía datos mediante comunicación serial.

Funciones Descartadas o Pospuestas:

La conexión y control físico de los servomotores del autómata se pospuso como trabajo futuro. Esta decisión se tomó para priorizar la funcionalidad core del reconocimiento de gestos y la integración IoT.

Registro de Decisiones Técnicas

Decisión en Sprint 2 - Migración a Arquitectura Centralizada:

Se decidió migrar toda la lógica de negocio a la Raspberry Pi ejecutando Flask en lugar de distribuir la inteligencia entre la ESP32 y la RPi. Esta decisión se tomó porque la

ESP32 presentaba limitaciones de memoria y procesamiento para ejecutar modelos de aprendizaje automático de manera eficiente. La centralización en la Raspberry Pi facilitó el desarrollo, las actualizaciones de software y el debugging.

Decisión en Sprint 3 - Uso de Raspberry Pi 4:

Se optó por una Raspberry Pi 4 en lugar de modelos anteriores para garantizar la potencia de procesamiento necesaria para ejecutar las operaciones de inteligencia artificial y el servidor web de manera simultánea sin degradación del rendimiento.

Decisión en Sprint 4 - Implementación de Threading y Queues:

Se implementó una arquitectura basada en threading y colas de mensajes para asegurar que la API de Flask no se bloqueara mientras la clasificación de gestos (una operación de alta latencia) se ejecutaba en paralelo. Esta decisión fue crítica para mantener la responsividad del sistema y permitir múltiples operaciones concurrentes.