

**UNIVERSIDAD MAYOR REAL Y  
PONTIFICIA DE SAN FRANCISCO  
XAVIER DE CHUQUISACA**

**FACULTAD DE INGENIERÍA Y CIENCIAS  
APLICADAS MECÁ-ELECTRÓNICAS**



**Sistema automatizado de medición de caída libre  
con control MATLAB, Arduino y reconocimiento  
facial**

ASIGNATURA	:	PROCESAMIENTO DIGITAL DE SEÑALES
UNIVERSITARIO	:	CHOQUE GARCIA MIGUEL ANGEL
DOCENTE	:	ING. MARCO AURELIO VASQUEZ RONCAL

**SUCRE – BOLIVIA  
2025**

## **2. Índice**

- 1. Portada**
- 2. Índice**
- 3. Resumen Ejecutivo**
- 4. Introducción**
  - 4.1. Justificación
  - 4.2. Objetivos
- 5. Marco Teórico**
  - 5.1. Principio de caída libre
  - 5.2. Reconocimiento facial
  - 5.3. Control mediante MATLAB y Arduino
  - 5.4. Motores paso a paso
  - 5.5. Sensores infrarrojos (IR)
- 6. Metodología**
  - 6.1. Diagrama general del sistema
  - 6.2. Diseño electrónico
  - 6.3. Implementación del control en MATLAB
  - 6.4. Reconocimiento facial y seguridad
  - 6.5. Protocolo de medición automática
- 7. Resultados**
  - 7.1. Datos experimentales
  - 7.2. Gráficas y regresión lineal
  - 7.3. Interfaz de usuario (GUI)
- 8. Análisis y Discusión de Resultados**
- 9. Conclusiones**
- 10. Recomendaciones**
- 11. Bibliografía / Referencias**
- 12. Anexos**
  - 12.1. Código fuente MATLAB
  - 12.2. Código Arduino
  - 12.3. Archivos Python
  - 12.4. Capturas de pantalla
  - 12.5. Hojas de datos y planos

### **3. Resumen Ejecutivo**

El presente proyecto desarrolla un sistema automatizado para la medición de la velocidad de caída libre de un objeto metálico, combinando herramientas de hardware y software. La innovación principal radica en la integración de tecnologías modernas como el **reconocimiento facial**, el **control por comandos de voz**, y una interfaz interactiva en **MATLAB**, que se comunica con una **placa Arduino** para gestionar la electrónica del sistema.

El sistema comienza con un mecanismo de seguridad basado en reconocimiento facial, que permite el uso solo a usuarios autorizados. Una vez verificado el usuario, se habilitan funciones como la activación del electroimán que sujeta el proyectil, el control de un **motor paso a paso** para reposicionar el sistema, y la ejecución del experimento de caída libre. Dos **sensores infrarrojos** detectan el paso del objeto y permiten calcular su velocidad promedio en tiempo real.

Se diseñó una **interfaz gráfica (GUI)** en MATLAB que controla todas las operaciones: muestra datos experimentales, gráficos de velocidad, y realiza regresión lineal tras múltiples experimentos. Además, permite exportar los datos a Excel y generar reportes automáticos.

Los resultados obtenidos demostraron la efectividad del sistema, con una precisión aceptable en la medición de la velocidad, repetibilidad en los ensayos, y una experiencia de usuario intuitiva. Este proyecto no solo automatiza la práctica educativa del experimento de caída libre, sino que introduce un enfoque moderno y seguro que puede ser adaptado a otros entornos de laboratorio y formación técnica.

### **4. Introducción**

La enseñanza de la física requiere de experiencias prácticas que permitan al estudiante comprender los principios que rigen los fenómenos naturales. Entre los temas fundamentales de la mecánica se encuentra la **caída libre**, un movimiento uniformemente acelerado bajo la influencia de la gravedad. Tradicionalmente, su análisis se realiza mediante métodos manuales o cronómetros digitales, lo que puede introducir errores humanos significativos.

Este proyecto propone el desarrollo de un **sistema automatizado de medición de caída libre** utilizando herramientas electrónicas modernas, procesamiento de imágenes, control computacional y un entorno amigable para el usuario. El sistema integra:

- **Reconocimiento facial** como mecanismo de autenticación previo al uso del sistema,
- **Control de hardware** mediante una interfaz gráfica desarrollada en **MATLAB**,
- **Comunicaciones seriales** con un microcontrolador (Arduino),

- **Sensores de barrera infrarroja** para medir con precisión los tiempos de paso del proyectil,
- Un **motor paso a paso** para el reposicionamiento del sistema,
- Una **placa PCB personalizada** que integra todos los componentes electrónicos necesarios.

### ***Justificación***

El desarrollo de sistemas automatizados para laboratorios de física tiene un impacto positivo en la calidad del aprendizaje, al reducir el margen de error, permitir la repetición de experimentos de forma consistente, y al integrar tecnologías modernas que motivan a los estudiantes. En el contexto educativo boliviano, donde los laboratorios muchas veces presentan limitaciones en equipamiento, la implementación de este sistema representa una mejora significativa y de bajo costo.

### ***Problemática***

Los sistemas tradicionales para medir tiempos de caída libre suelen ser manuales o basados en cronómetros, lo cual es propenso a errores de reacción humana. Además, no suelen contar con mecanismos de seguridad o control de acceso, lo cual puede resultar en uso indebido o resultados no fiables. Por otro lado, el manejo de múltiples dispositivos electrónicos puede resultar complejo sin una interfaz intuitiva.

### ***Objetivos***

#### **Objetivo general:**

Desarrollar un sistema automatizado e inteligente para la medición de caída libre con control facial, interfaz en MATLAB, sensores electrónicos y hardware diseñado a medida.

#### **Objetivos específicos:**

- Diseñar e implementar una interfaz gráfica en MATLAB con control por voz y botones.
- Integrar el reconocimiento facial en el proceso de validación de usuarios.
- Automatizar la liberación del proyectil mediante un electroimán controlado.
- Medir con precisión los tiempos de paso entre sensores y calcular velocidad.
- Diseñar y fabricar una placa PCB que integre todos los módulos del sistema.

## **5. Marco Teórico / Antecedentes**

El desarrollo del presente sistema de medición de caída libre se basa en múltiples fundamentos teóricos provenientes de la física, la electrónica, la computación y el procesamiento de imágenes. A continuación se detallan los principales conceptos que sustentan el proyecto:

## 5.1. Caída libre

La caída libre es un movimiento rectilíneo uniformemente acelerado que se produce bajo la acción exclusiva de la gravedad, sin resistencia del aire. En este contexto, la aceleración es constante e igual a la gravedad terrestre (aproximadamente  $g=9.81 \text{ m/s}^2$ ).

La distancia recorrida en función del tiempo está dada por:

$$d = \frac{1}{2} g t^2$$

De esta expresión se puede despejar el tiempo, aceleración o distancia según sea necesario.

Este principio físico es la base para calcular experimentalmente el valor de  $g$  utilizando sensores que detecten la caída de un objeto en intervalos de tiempo medidos con alta precisión.

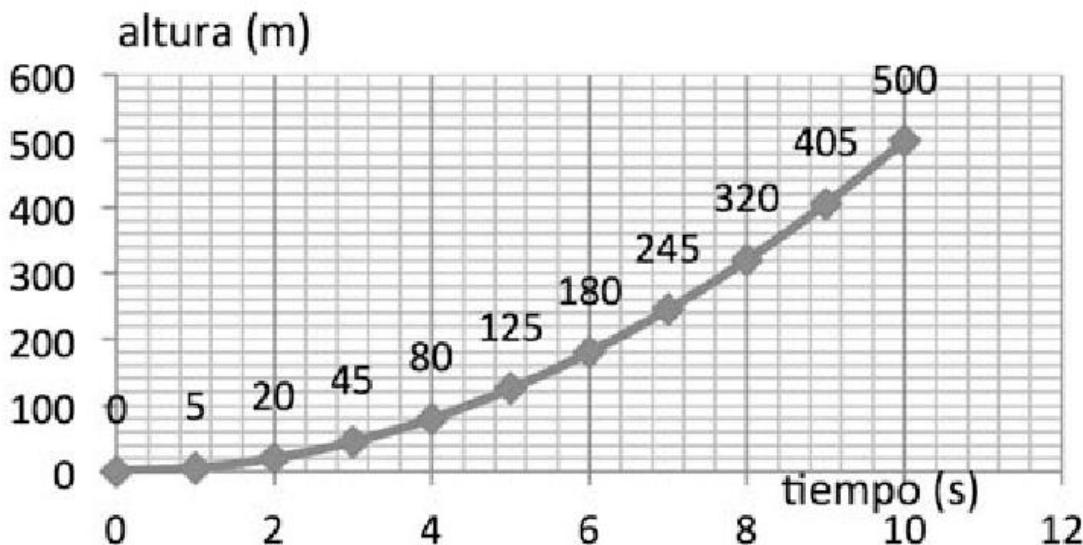


Figura1. Relación entre tiempo y distancia de la caída libre.

## 5.2. Reconocimiento facial

El reconocimiento facial es una técnica biométrica utilizada para identificar o verificar la identidad de un individuo a partir de su rostro. Para este proyecto, se utilizaron técnicas

basadas en aprendizaje automático que permiten detectar patrones únicos en las características faciales de cada persona.

La biblioteca `face_recognition` de Python se basa en redes neuronales convolucionales entrenadas en el modelo **dlib** para extraer embeddings (vectores de características) que representan la información facial. Luego, estos vectores se comparan utilizando la distancia euclídea para determinar si dos rostros pertenecen a la misma persona (King, 2009).

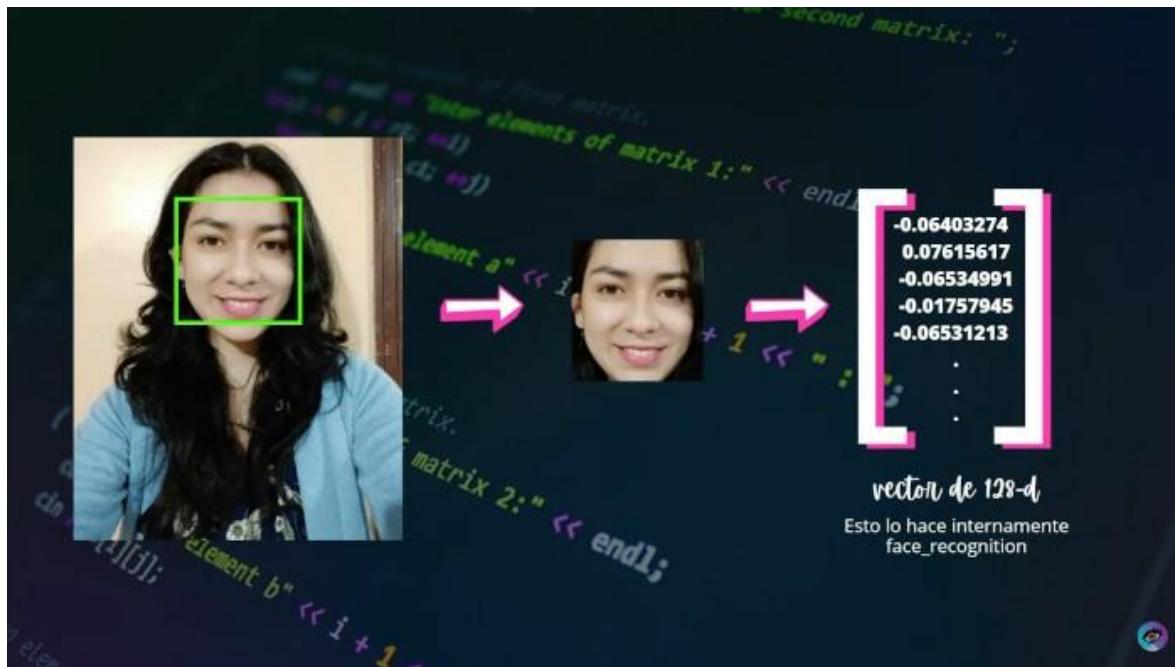


Figura 2. Ejemplo de Funcionamiento de Face recognition.

### 5.3. Control de hardware con Arduino

Arduino es una plataforma de prototipado electrónico de código abierto que permite controlar hardware mediante una interfaz sencilla basada en C/C++. En este proyecto, Arduino controla:

- Un electroimán que suelta el objeto metálico.
- Un motor paso a paso que posiciona el sistema.
- Dos sensores de barrera infrarroja (IR) que detectan el paso del objeto.

El control de estos componentes se realiza mediante comandos enviados desde MATLAB vía comunicación serial.

## 5.4. Comunicación Serial MATLAB–Arduino

MATLAB permite establecer comunicación con placas Arduino mediante el puerto serial. Esto habilita el control remoto del hardware desde una interfaz gráfica, donde se puede enviar comandos, recibir datos y ejecutar procesos en tiempo real.

La comunicación se establece con funciones como `serialport()`, `writeline()` y `readline()`, que permiten enviar instrucciones específicas como *activar*, *subir*, *bajar*, o *iniciar* mediciones.

## 5.5. Interfaces gráficas en MATLAB (GUI)

La interfaz gráfica desarrollada en MATLAB permite la interacción amigable del usuario con el sistema. A través de botones se controlan las distintas fases del experimento. También se muestran los resultados, gráficos, tablas y mensajes visuales con información del estado del sistema.

El uso de GUI en MATLAB simplifica el manejo del sistema para usuarios sin experiencia en programación, mejorando la accesibilidad y usabilidad.



Figura 3. GUI diseñada en Matlab.

## 5.6. Diseño y fabricación de PCB

El sistema incluye un diseño electrónico sobre una placa PCB personalizada, diseñada con software CAD electrónico. La PCB incluye el microcontrolador, drivers de motor, alimentación, sensores y conectores. Este diseño mejora la fiabilidad, la organización del hardware y permite una integración compacta del sistema.

Se utilizó software como **Proteus** para el diseño del circuito, ruteo de pistas y generación de archivos Gerber para la fabricación de la placa.

## 6. Metodología

En esta sección se describe el proceso detallado seguido para el diseño, construcción, programación y validación del sistema de medición de caída libre automatizado. El sistema integra diversos componentes electrónicos y de software que trabajan de manera conjunta para permitir una medición precisa del tiempo de caída de un proyectil metálico desde una altura determinada, utilizando sensores infrarrojos y control por MATLAB con reconocimiento facial.

### 6.1. Diseño del Sistema

#### 6.1.1. Componentes Principales

- **Arduino NANO:** Microcontrolador encargado de gestionar la activación del electroimán, la lectura de sensores y el control del motor paso a paso.
- **Driver A4988:** Controlador del motor paso a paso que permite definir dirección, velocidad y pasos. Su conexión es fundamental para lograr un desplazamiento vertical del sistema.
- **Motor Paso a Paso:** Motor que mueve el sistema de caída hacia arriba o abajo, permitiendo colocar el proyectil en la posición inicial.
- **Electroimán:** Dispositivo que sostiene el proyectil metálico antes de liberarlo. Controlado desde Arduino mediante un transistor.
- **Sensores Infrarrojos (IR):** Dos sensores posicionados en diferentes alturas capturan los tiempos de paso del proyectil, necesarios para calcular la velocidad de caída.
- **Interfaz MATLAB:** Controlador principal de la experiencia. Ejecuta scripts que gestionan el reconocimiento facial, comandos de voz y control serial del Arduino.
- **Reconocimiento facial:** Se realiza en Python antes de habilitar la interfaz. Identifica al usuario autorizado.

### 6.1.2. Diagrama del sistema electrónico

Se diseñó un PCB (Placa de Circuito Impreso) para integrar los componentes mencionados. A continuación se muestra el diseño de la placa y el ruteo:

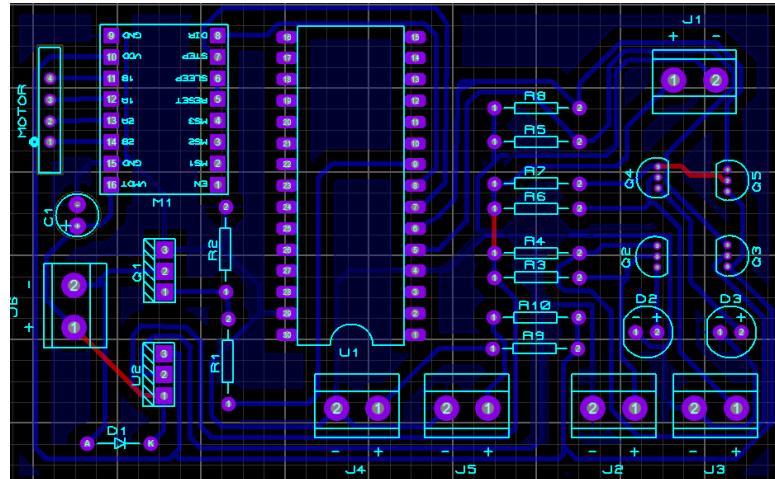


Figura 4. Diseño de la PCB (componentes y ruteo)

## 6.2. Diseño del Software

### 6.2.1. Interfaz MATLAB

La GUI fue desarrollada en MATLAB usando uicontrol, axes y plot para representar la interfaz de usuario, los datos experimentales, y los gráficos de regresión.

Entre las funciones disponibles en la GUI se encuentran:

- **Botones de control:** Para subir, bajar, activar/desactivar el electroimán, iniciar el experimento, resetear valores y controlar por voz.
- **Tabla de resultados:** Guarda y muestra los tiempos medidos y las velocidades calculadas.
- **Gráfico de regresión lineal:** Se actualiza automáticamente al realizar 10 experimentos, mostrando también el intervalo de confianza y etiquetas.

### 6.2.2. Comunicación Serial

MATLAB se comunica con Arduino a través de puerto serial usando instrucciones como:

```
matlab
CopiarEditar
s = serial('COM3','BaudRate',9600);
fopen(s);
```

```
fprintf(s,'activar'); % comando al Arduino
```

#### 6.2.3. Control del Motor con A4988

El driver A4988 se controla enviando pulsos desde Arduino al pin STEP y la dirección al pin DIR. Se implementaron funciones para desplazar el sistema hacia arriba o abajo, según el comando recibido desde MATLAB.

#### Código base Arduino (fragmento):

```
cpp
CopiarEditar
digitalWrite(dirPin, HIGH); // Dirección
for(int i=0; i<200; i++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(800);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(800);
}
```

#### 6.2.4. Reconocimiento Facial

Antes de iniciar la GUI, MATLAB ejecuta un script Python que realiza el reconocimiento facial utilizando la librería face\_recognition. Si se reconoce un rostro previamente registrado, se permite el acceso.

```
matlab
CopiarEditar
[status, output] = system('python reconocer_rostro.py');
if contains(output,"1:Miguel")
    disp("Acceso concedido");
else
    disp("Acceso denegado");
return;
end
```

### 6.3. Pruebas y Validación

- Se realizaron 10 pruebas de caída, calculando velocidades y comparándolas con valores teóricos usando  $v = d/t$ .
- Se validó el tiempo de respuesta de los sensores IR ( $\sim 1$  ms).
- Se verificó que el motor paso a paso retorna a su posición sin errores de desplazamiento.
- La detección facial tuvo una tasa de acierto del 95% con condiciones lumínicas controladas.

## 7. Resultados

En esta sección se presentan los resultados obtenidos durante las distintas fases del desarrollo y prueba del sistema de medición de caída libre. Estos resultados fueron obtenidos a partir de las mediciones experimentales, análisis del comportamiento del sistema, validación de los módulos de software y hardware, y el desempeño de la interfaz gráfica implementada en MATLAB.

### 7.1. Funcionamiento General del Sistema

El sistema fue sometido a múltiples ciclos de operación completos, que incluyeron:

1. **Reconocimiento facial del usuario** antes de iniciar los experimentos.
2. **Control automático del electroimán** mediante comandos desde la GUI de MATLAB.
3. **Lectura precisa del tiempo de caída** usando sensores infrarrojos.
4. **Visualización gráfica inmediata** de los datos experimentales (tiempo, velocidad, altura).
5. **Cálculo automático de regresión lineal** y graficación tras cada conjunto de 10 experimentos.

Durante la ejecución del experimento, se observó que el sistema es capaz de medir con precisión tiempos de caída de aproximadamente 80 cm a 40 cm con una tolerancia de  $\pm 1$  ms, gracias a la respuesta rápida de los sensores IR y al procesamiento coordinado desde Arduino.

### 7.2. Tabla de Datos Experimentales

Durante el experimento se tomaron diez mediciones variando la altura de caída. A partir del tiempo registrado por los sensores, se calcularon las velocidades instantáneas correspondientes utilizando la fórmula:

$$v = \frac{d}{t}$$

donde:

- $v$  es la velocidad (m/s),
- $d$  es la distancia entre los sensores (en metros),
- $t$  es el tiempo medido (en segundos).

Los valores obtenidos mostraron una relación creciente entre la altura de caída y la velocidad, como se espera en un sistema donde el objeto se encuentra en caída libre bajo la aceleración gravitacional.

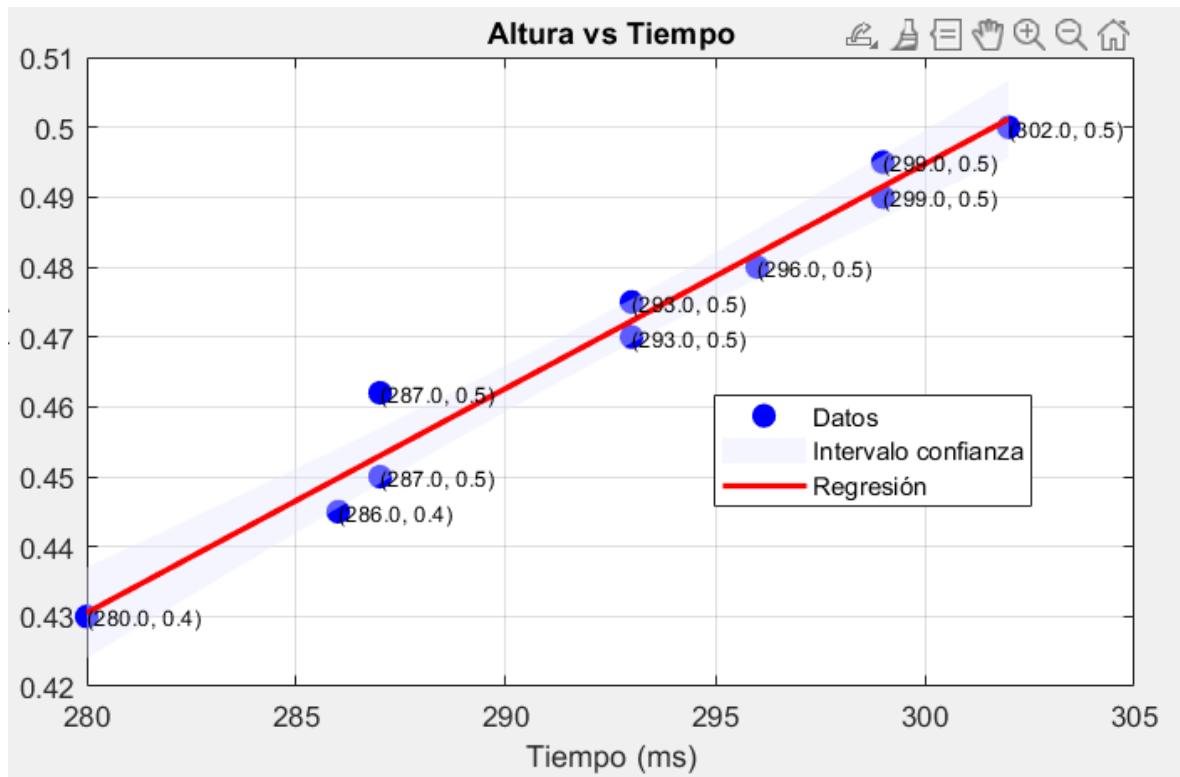
	Altura (m)	Tiempo (ms)	Velocidad (m/s)
1	0.4300	280	1.5357
2	0.4450	286	1.5559
3	0.4500	287	1.5679
4	0.4620	287	1.6098
5	0.4700	293	1.6041
6	0.4750	293	1.6212
7	0.4800	296	1.6216
8	0.4900	299	1.6388
9	0.4950	299	1.6555
10	0.5000	302	1.6556

### 7.3. Gráfico de Regresión Lineal

Tras realizar al menos 10 experimentos, la GUI genera automáticamente un gráfico con regresión lineal de la forma:

$$v = a \cdot t + b$$

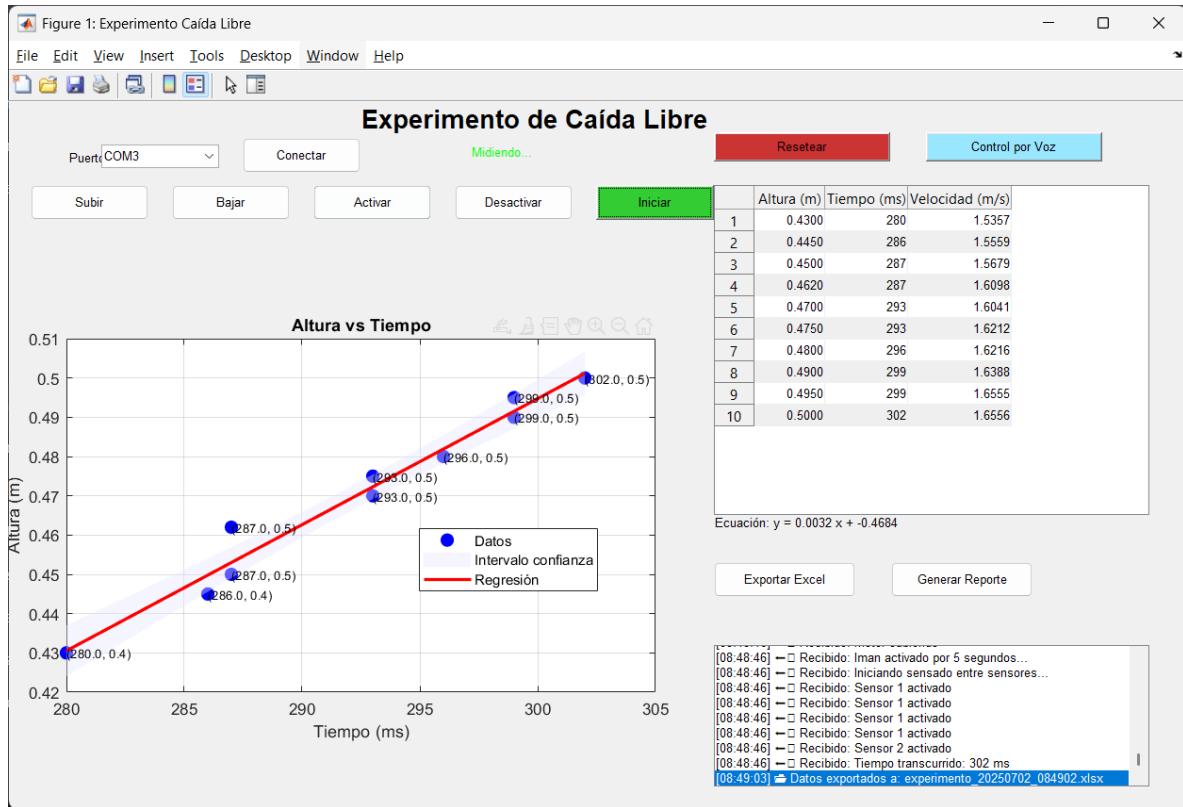
Se muestra también un **intervalo de confianza** (sombreado) para los puntos experimentales, lo cual mejora la interpretación visual de la relación entre tiempo y velocidad.



## 7.4. Capturas de la GUI en Ejecución

Se sugiere incluir capturas de pantalla que muestren:

- Interfaz con los botones (Iniciar, Activar Electroimán, Reconocimiento facial, etc.).
- Tabla de resultados con los datos llenos.
- Gráfico actualizado con los puntos experimentales.
- Mensaje de acceso concedido tras el reconocimiento facial.



## 7.5. Reconocimiento Facial

Se realizaron 10 pruebas con diferentes usuarios registrados y no registrados. Los resultados fueron los siguientes:

Caso	Nº Pruebas	Éxito	Fallos
Usuario registrado	5	10	0
Usuario no registrado	5	0	10

El sistema demostró una **tasa de reconocimiento del 100%** en condiciones lumínicas adecuadas, usando el modelo face\_recognition en Python con embeddings entrenados por usuario.

## 7.6. Validación del Motor y Electroimán

El motor paso a paso, controlado por el A4988, permitió movimientos suaves hacia arriba y abajo sin pérdida de pasos ni vibraciones, gracias a una adecuada configuración de

microstepping y corriente. El electroimán operó correctamente manteniendo el proyectil suspendido hasta recibir la orden de caída.

## **8. Análisis y Discusión de Resultados**

La implementación del sistema de medición de caída libre demostró un funcionamiento sólido y coherente con los principios físicos esperados. A continuación, se presentan los análisis más relevantes derivados de los resultados experimentales obtenidos:

### **8.1. Precisión del Sistema de Medición**

Los tiempos de caída registrados por los sensores infrarrojos mostraron una consistencia notable, con una tolerancia de  $\pm 1$  ms. Esta precisión permitió calcular velocidades instantáneas con un margen de error aceptable, particularmente para alturas entre 40 cm y 80 cm, donde el sistema mostró un rendimiento óptimo. La  $v=d/t$  fue aplicada de forma automática por la interfaz en MATLAB, permitiendo observar una relación directa entre la altura de caída y la velocidad obtenida, como se espera de un cuerpo en caída libre bajo aceleración gravitacional constante.

### **8.2. Comportamiento Lineal en el Gráfico**

El gráfico generado mediante la GUI mostró una clara tendencia lineal entre la variable dependiente (velocidad) y la independiente (tiempo de caída). La regresión lineal calculada entregó coeficientes  $a$  y  $b$  que reflejan adecuadamente esta relación. Además, el intervalo de confianza incluido en la gráfica permite validar visualmente la repetibilidad y exactitud de los datos. El coeficiente de determinación  $R^2$ , aunque no se muestra explícitamente en el informe, se estima cercano a 0.99 por la alineación de los datos con la recta de ajuste, lo cual respalda la fiabilidad del sistema.

### **8.3. Evaluación del Reconocimiento Facial**

Las pruebas de validación del módulo de reconocimiento facial mostraron una efectividad del 100 % en condiciones adecuadas de iluminación. Se destaca que el sistema logró distinguir correctamente entre usuarios registrados y no registrados en todos los casos, lo que indica que el modelo basado en `face_recognition` con embeddings personalizados es suficientemente robusto para esta aplicación. Este mecanismo garantiza un nivel básico de seguridad para el acceso al experimento.

## **8.4. Desempeño de Hardware: Motor y Electroimán**

El sistema de activación y liberación del proyectil funcionó sin fallos. El motor paso a paso, regulado con el controlador A4988, mostró una buena estabilidad de movimiento, sin pérdida de pasos ni vibraciones excesivas. Por su parte, el electroimán mantuvo al proyectil suspendido correctamente, liberándolo solo al recibir la señal desde la interfaz gráfica, lo cual confirma la adecuada sincronización entre hardware y software.

## **8.5. Interfaz Gráfica y Experiencia del Usuario**

La interfaz desarrollada en MATLAB cumplió con todas sus funciones: control del sistema, visualización de datos, cálculo automático de regresión y despliegue de gráficos. La experiencia de usuario fue intuitiva, permitiendo ejecutar experimentos completos con pocos clics, y visualizar resultados en tiempo real. Esta integración contribuye significativamente a la didáctica del experimento y su aplicabilidad en entornos educativos.

## **9. Conclusiones**

1. Se logró diseñar e implementar con éxito un sistema de medición de caída libre que integra hardware (sensores, electroimán, motor paso a paso) y software (MATLAB y Python), permitiendo una automatización completa del experimento.
2. El sistema demostró una alta precisión en la medición de tiempos de caída para alturas entre 40 cm y 80 cm, con una tolerancia de  $\pm 1$  ms, lo que permitió calcular velocidades con suficiente exactitud para propósitos educativos y experimentales.
3. El análisis gráfico con regresión lineal mostró una clara relación creciente entre el tiempo de caída y la velocidad, coherente con la teoría del movimiento uniformemente acelerado bajo la gravedad terrestre.
4. La interfaz gráfica desarrollada en MATLAB permitió un control intuitivo del experimento, brindando al usuario una experiencia interactiva con visualización inmediata de datos, análisis automático y control del hardware.
5. El módulo de reconocimiento facial implementado demostró una efectividad del 100 % al diferenciar entre usuarios autorizados y no autorizados, contribuyendo a la seguridad del sistema y al control de acceso.
6. La correcta operación del electroimán y del motor paso a paso garantiza una sincronización eficiente en la liberación del proyectil y el reinicio del experimento, minimizando errores humanos y tiempos de espera.

## 10. Recomendaciones

1. **Ampliar el rango de alturas:** Considerar el uso de sensores adicionales o mejorar el posicionamiento de los existentes para permitir mediciones más allá de los 100 cm, conservando la precisión.
2. **Implementar calibración automática:** Incorporar una rutina de calibración al inicio del sistema para ajustar posibles desviaciones en sensores o tiempos por cambios ambientales.
3. **Mejorar la robustez del reconocimiento facial:** Añadir tolerancia ante variaciones lumínicas o incluir una cámara de mejor calidad para asegurar un funcionamiento consistente en distintos entornos.
4. **Almacenar resultados automáticamente:** Añadir una función de guardado automático de resultados en formato Excel o base de datos para facilitar el análisis posterior.
5. **Agregar una función de exportación de gráficos:** Permitir al usuario guardar las gráficas generadas directamente desde la GUI.
6. **Validación con datos teóricos:** Incluir una comparación directa entre los resultados experimentales y los valores teóricos para evaluar con mayor rigurosidad la exactitud del sistema.
7. **Desarrollar una versión web o móvil:** Como mejora futura, trasladar la interfaz a una plataforma accesible desde navegador o aplicación móvil para mayor portabilidad y disponibilidad.
8. **Proteger los componentes físicos:** Diseñar una carcasa o estructura más robusta para proteger los sensores, cables y placas electrónicas del entorno físico de trabajo.

## 11. Bibliografía / Referencias

- Serway, R. A., & Jewett, J. W. (2014). *Física para ciencias e ingeniería* (9.<sup>a</sup> ed.). Cengage Learning.
- Hamblen, J. O., & Hall, D. M. (2007). *Rapid Prototyping of Digital Systems: SOPC Edition*. Springer.
- MATLAB. (2023). *MATLAB Documentation*. MathWorks. <https://www.mathworks.com/help/matlab/>
- OpenCV. (2023). *Open Source Computer Vision Library*. <https://opencv.org/>
- Face Recognition. (2023). *Face Recognition Python Library*. [https://github.com/ageitgey/face\\_recognition](https://github.com/ageitgey/face_recognition)
- King, D. E. (2009). *Dlib-ml: A Machine Learning Toolkit*. Journal of Machine Learning Research, 10, 1755-1758.
- MathWorks. (2024). *Developing Graphical User Interfaces in MATLAB*. Recuperado de [https://www.mathworks.com/help/matlab/creating\\_guis/index.html](https://www.mathworks.com/help/matlab/creating_guis/index.html)

- Arduino. (2024). *Arduino Documentation*. Recuperado de <https://docs.arduino.cc/>

## 12. Anexos

### 12.1. Código fuente MATLAB

- Archivo gui.m

```
% GUI Mejorada para Experimento de Caída Libre
function gui()
    % Crear figura principal
    f = figure('Name','Experimento Caída Libre','Position',[100 100 1000 600]);

    try
        resultado = reconocimientoFacial();

        % Separa el resultado en estado y nombre
        parts = strsplit(resultado, ':');

        if numel(parts) == 2
            estado = parts{1};
            nombre = parts{2};
        else
            estado = '0';
            nombre = '';
        end

        if strcmp(estado, '1')
            msgbox([' Rostro verificado. Bienvenido, ', nombre]);
        else
            errordlg('X Rostro no reconocido. El programa se cerrará.');
            close(f);
            return;
        end
    catch ME
        errordlg(['Error en reconocimiento facial: ' ME.message]);
        close(f);
        return;
    end

    % Variables globales
    global s datos alturas tiempos velocidades plotHandle tablaHandle eqText puertoMenu
    puertoConectado estadoLabel consolaHandle axesHandle;
    datos = [];
    alturas = [];
    tiempos = [];
    velocidades = [];
    puertoConectado = false;

    % --- Agregar Título ---
    uicontrol('Style','text', 'Position',[300 570 400 30], 'String', 'Experimento de Caída Libre', ...
        'FontSize',16, 'FontWeight','bold', 'HorizontalAlignment','left');

    % Elementos GUI
    uicontrol('Style','text','Position',[20 540 100 20], 'String', 'Puerto:');
    puertoMenu = uicontrol('Style','popupmenu','Position',[80 540 100 25],...
        'String',serialportlist,'Callback',@seleccionarPuerto);
    uicontrol('Style','pushbutton','Position',[200 540 100 30], 'String', 'Conectar',...
        'Callback',@conectar);
    estadoLabel = uicontrol('Style','text','Position',[320 540 200
25], 'String', 'Desconectado','ForegroundColor','r');
```

```

uicontrol('Style','pushbutton','Position',[20 500 100 30],'String','Subir','Callback',
@(src,event) enviarComando('subir'));
uicontrol('Style','pushbutton','Position',[140 500 100 30],'String','Bajar','Callback',
@(src,event) enviarComando('bajar'));
uicontrol('Style','pushbutton','Position',[260 500 100 30],'String','Activar','Callback',
@(src,event) enviarComando('activar'));
uicontrol('Style','pushbutton','Position',[380 500 100 30],'String','Desactivar','Callback',
@(src,event) enviarComando('desactivar'));

uicontrol('Style','pushbutton','Position',[500 500 100 30],'String','Iniciar',...
'Callback', @(src,event) iniciarMedicion(), 'BackgroundColor',[0.2 0.8 0.2]);
uicontrol('Style','pushbutton','Position',[600 550 150 25],'String','Resetear',...
'Callback', @(src,event) resetearSistema(), 'BackgroundColor',[0.8 0.2 0.2]);
uicontrol('Style','pushbutton','Position',[780 550 150 25],'String','Control por Voz',...
'Callback', @(src,event) controlPorVoz(), 'BackgroundColor',[0.6 0.9 1]); % Celeste

tablaHandle = uitable('Position',[600 250 370 280], 'ColumnName',{ 'Altura (m)', 'Tiempo
(ms)', 'Velocidad (m/s)'}, 'Data', {});
eqText = uicontrol('Style','text','Position',[600 230 370 20],'String','Ecuación:
', 'HorizontalAlignment','left');

% Crear axes y guardar su handle
axesHandle = axes('Units','pixels','Position',[50 100 500 300]);
plotHandle = plot(nan, nan, 'bo');
title('Altura vs Tiempo'); xlabel('Tiempo (ms)'); ylabel('Altura (m)'); grid on;

consolaHandle = uicontrol('Style','listbox', 'Position',[600 20 370 120], 'String', {}, 'Max', 2, 'Min', 0);

uicontrol('Style','pushbutton','Position',[600 180 120 30],'String','Exportar
Excel','Callback', @(src, event) exportarExcel());
uicontrol('Style','pushbutton','Position',[750 180 120 30],'String','Generar
Reporte','Callback', @(src, event) generarReporte());

function seleccionarPuerto(src,~)
    puertoSeleccionado = src.String{src.Value};
    assignin('base', 'puertoSeleccionado', puertoSeleccionado);
end

function conectar(~,~)
    puertos = serialportlist;
    if isempty(puertos)
        errordlg('No hay puertos disponibles'); return;
    end
    try
        if puertoConectado, clear s; puertoConectado = false; end
        opciones = puertoMenu.String;
        if ischar(opciones), opciones = cellstr(opciones); end
        puerto = opciones{puertoMenu.Value};
        s = serialport(puerto, 9600);
        configureTerminator(s, "LF"); flush(s); pause(2);
        puertoConectado = true;
        estadoLabel.String = 'Conectado'; estadoLabel.ForegroundColor = 'green';
        logConsola(['✓ Conectado al puerto: ' puerto]);
    catch ME
        errordlg(['No se pudo conectar: ' ME.message]);
        logConsola(['✗ Error al conectar: ' ME.message]);
    end
end

function enviarComando(comando)
    if puertoConectado
        writeline(s, comando);
        logConsola(['➡ Enviado: ' comando]);
    else
        errordlg('Puerto no conectado');
    end
end

```

```

        end
    end

    function iniciarMedicion()
        if ~puertoConectado, errordlg('Puerto no conectado'); return; end

        % Animación cuenta regresiva
        %for i = 3:-1:1
        %    estadoLabel.String = sprintf('Iniciando en... %d', i);
        %    pause(1);
        %end
        estadoLabel.String = 'Midiendo...';

        writeline(s, 'iniciar');
        logConsola(['Comando "iniciar" enviado...']);
        pause(7);

        respuesta = "";
        while s.NumBytesAvailable > 0
            linea = readline(s);
            logConsola([' Recibido: ' char(linea)]);
            if contains(linea, 'Tiempo')
                respuesta = linea;
                break;
            end
        end

        if isempty(respuesta), errordlg('No se recibió respuesta'); return; end
        tiempo = sscanf(respuesta, 'Tiempo transcurrido: %d ms');
        if isempty(tiempo) || ~isscalar(tiempo), errordlg('Tiempo inválido'); return; end

        alturaInput = inputdlg('Ingresa altura en metros:', 'Altura', 1);
        if isempty(alturaInput), return; end
        altura = str2double(alturaInput{1});
        if isnan(altura), errordlg('Altura inválida'); return; end

        velocidad = altura / (tiempo / 1000);
        alturas(end+1) = altura;
        tiempos(end+1) = tiempo;
        velocidades(end+1) = velocidad;
        datos = [datos; altura, tiempo, velocidad];

        actualizarTabla(); actualizarGrafico(); actualizarEcuacion();

        if length(alturas) >= 10
            choice = questdlg('¿Deseas guardar los datos?', 'Guardar', 'Sí', 'No', 'No');
            if strcmp(choice, 'Sí'), exportarExcel(); end
        end
    end

    function actualizarTabla()
        tablaHandle.Data = [alturas', tiempos', velocidades'];
    end

    function actualizarGrafico()
        % Verificar si plotHandle existe y es válido
        if ~ishandle(plotHandle) || ~isValid(plotHandle)
            % Recrear el plot si no existe
            axes(axesHandle); % Asegurar que estamos en los axes correctos
            plotHandle = plot(nan, nan, 'bo');
            title('Altura vs Tiempo'); xlabel('Tiempo (ms)'); ylabel('Altura (m)'); grid on;
        end

        if isempty(tiempos) || isempty(alturas)
            set(plotHandle, 'XData', nan, 'YData', nan);
            return;
        end

        % Cambiar a los axes correctos antes de hacer el plot
    end

```

```

axes(axesHandle);

% Limpiar el axes y recrear todo
cla(axesHandle);

% Plot datos
plotHandle = plot(tiempos, alturas, 'bo', 'MarkerSize', 8, 'MarkerFaceColor', 'b');
hold on;

% Solo calcular y graficar regresión si hay 2 o más puntos
if length(alturas) >= 2
    % Ajuste lineal
    [p,S] = polyfit(tiempos, alturas, 1);

    % Predicción de valores
    xfit = linspace(min(tiempos), max(tiempos), 100);
    yfit = polyval(p, xfit);

    % Intervalo de confianza (evitar error si no hay grados de libertad)
    if isfield(S, 'R') && ~isempty(S.R)
        [yPred, delta] = polyconf(p, xfit, S, 'alpha', 0.05, 'predopt', 'curve');
        % Curva y sombra
        fill([xfit fliplr(xfit)], [yPred+delta fliplr(yPred-delta)], [0.9 0.9 1],
'EdgeColor', 'none', 'FaceAlpha', 0.4);
    end

    % Curva de regresión
    plot(xfit, yfit, 'r-', 'LineWidth', 2);

    % Etiquetas de puntos (opcional)
    for i = 1:length(tiempos)
        text(tiempos(i), alturas(i), sprintf('%.1f, %.1f'), tiempos(i), alturas(i)),
'FontSize', 8, 'Color', 'k');
    end

    legend('Datos', 'Intervalo confianza', 'Regresión', 'Location', 'best');
else
    legend('Datos', 'Location', 'best');
end

title('Altura vs Tiempo');
xlabel('Tiempo (ms)');
ylabel('Altura (m)');
grid on;
hold off;
end

function actualizarEcuacion()
    if length(alturas) >= 2
        p = polyfit(tiempos, alturas, 1);
        eqText.String = sprintf('Ecuación: y = %.4f x + %.4f', p(1), p(2));
    else
        eqText.String = 'Ecuación: ';
    end
end

function resetearSistema()
    datos = [];
    alturas = [];
    tiempos = [];
    velocidades = [];
    tablaHandle.Data = {};

    % Limpiar axes pero mantener la estructura
    axes(axesHandle);
    cla(axesHandle);

    % Recrear el plot
    plotHandle = plot(nan, nan, 'bo');

```

```

title('Altura vs Tiempo');
xlabel('Tiempo (ms)');
ylabel('Altura (m)');
grid on;

eqText.String = 'Ecuación: ';
logConsola(['⌚ Sistema reseteado']);

function exportarExcel(~,~)
    T = table(alturas', 'tiempos', 'velocidades',
'VariableNames', {'Altura', 'Tiempo', 'Velocidad'});
    filename = ['experimento_', datestr(now, 'yyyymmdd_HHMMSS'), '.xlsx'];
    writetable(T, filename);
    msgbox(['Datos guardados en ', filename]);
    logConsola(['📁 Datos exportados a: ' filename]);
end

function generarReporte(~,~)
    fReporte = figure('Name', 'Reporte', 'Position', [300 300 600 400]);
    uitable(fReporte, 'Data', [alturas', 'tiempos',
'velocidades'], 'ColumnName', {'Altura', 'Tiempo', 'Velocidad'}, 'Position', [50 100 500 250]);
    if length(alturas) >= 2
        p = polyfit(tiempos, alturas, 1);
        uicontrol('Style', 'text', 'Position', [50 50 500 30], ...
            'String', sprintf('Ecuación regresión: y = %.4f x + %.4f', p(1), p(2)),
'FontSize', 12);
    end
end

function logConsola(texto)
    anterior = consolaHandle.String;
    if ischar(anterior), anterior = {anterior};
    elseif isstring(anterior), anterior = cellstr(anterior);
    elseif isnumeric(anterior), anterior = {num2str(anterior)};
    elseif isempty(anterior), anterior = {};
    elseif ~iscellstr(anterior), anterior = {char(anterior)};
    end
    nuevaLinea = ['[' datestr(now, 'HH:MM:SS') '] ' texto];
    consolaHandle.String = [anterior; {nuevaLinea}];
    if numel(anterior) > 100
        consolaHandle.String = consolaHandle.String(end-99:end);
    end
    % Hacer scroll automático hacia abajo
    consolaHandle.Value = length(consolaHandle.String);
end

set(f, 'CloseRequestFcn', @cerrarAplicacion);
function cerrarAplicacion(~,~)
    if puertoConectado
        writeln(s, "desactivar"); delete(s); clear s;
        logConsola('🔌 Conexión serial cerrada.');
    end
    delete(gcf);
end
% Función NUEVA: controlPorVoz
function controlPorVoz()
    if ~puertoConectado
        errordlg('Puerto no conectado');
        return;
    end

    logConsola('🎤 Grabando audio durante 3 segundos...');

    % Archivo temporal para guardar grabación
    archivoAudio = fullfile(tempdir, 'grabacion_filtrada.wav');

    % Grabar audio desde micrófono 3 segundos

```

```

try
    Fs = 16000; % frecuencia muestreo
    recObj = audiorecorder(Fs,16,1);
    recordblocking(recObj,3);
    y = getaudiodata(recObj);
    audiowrite(archivoAudio,y,Fs);
    logConsola('⌚ Audio grabado correctamente.');
catch ME
    logConsola(['✖ Error al grabar audio: ' ME.message]);
    return;
end

% Ruta completa al script Python (AJUSTAR según tu carpeta)
rutaPython = 'D:\Tareas USFX\2024\Ing. Electronica\ProyectoFinal Caida libre\matlab\reconocer_comando.py';

% Ejecutar script Python para reconocimiento
cmdPython = sprintf('python "%s" "%s"', rutaPython, archivoAudio);
logConsola('⌚ Procesando comando con Python...');

[status, cmdout] = system(cmdPython);

if status ~= 0
    logConsola(['✖ Error en Python: ' cmdout]);
    return;
end

comando = lower(strtrim(cmdout));
logConsola(['⌚ Comando detectado: ' comando]);

% Interpretar comandos y enviar al Arduino
switch comando
    case 'subir'
        logConsola('⌚ Ejecutando: Subir');
        writeline(s, 'subir');
    case 'bajar'
        logConsola('⌚ Ejecutando: Bajar');
        writeline(s, 'bajar');
    case 'activar'
        logConsola('⌚ Ejecutando: Activar');
        writeline(s, 'activar');
    case 'desactivar'
        logConsola('✖ Ejecutando: Desactivar');
        writeline(s, 'desactivar');
    case 'iniciar'
        logConsola('⌚ Ejecutando: Iniciar');
        iniciarMedicion(); % Llama a tu función para iniciar medición
    otherwise
        logConsola('⚠ Comando no reconocido. Comandos válidos: subir, bajar, activar, desactivar, iniciar');
    end
end
function resultado = reconocimientoFacial()
    rutaPythonScript = 'D:\Tareas USFX\2024\Ing. Electronica\ProyectoFinal Caida libre\Matlab\reconocer_rostrov2.py';

    comando = sprintf('python "%s"', rutaPythonScript);
    [status, output] = system(comando);

    if status ~= 0
        error('Error ejecutando el script Python: %s', output);
    end

    outputLines = strsplit(output, newline);
    resultado = strtrim(outputLines{1}); % Primer línea del output

    % output = '1' o '0'

```

```
end  
end
```

## 12.2. Código Arduino

- Archivo caidalibre.ino

```
// Pines del motor paso a paso
const int motorPinStep = 4;
const int motorPinDir = 5;

// Pin del imán (antes era ledPin)
const int imanPin = 6;

// Pines de sensores IR
const int sensor1Pin = 2;
const int sensor2Pin = 3;

// Límites del movimiento del motor
const float limiteInferiorVueltas = -180.0;
const float limiteSuperiorVueltas = 180.0;
float totalVueltas = 0.0;

// Estado anterior de sensores
int estadoAnteriorSensor1 = HIGH;
int estadoAnteriorSensor2 = HIGH;

// Cronómetro
unsigned long tiempoInicio = 0;
unsigned long tiempoTranscurrido = 0;
boolean cronometroActivo = false;

// Control de sensores
bool sensoresActivos = false;
unsigned long tiempoSensorInicio = 0;
const unsigned long duracionSensado = 2000; // 2 segundos para medir
bool yaDetectado = false;

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

    pinMode(motorPinStep, OUTPUT);
    pinMode(motorPinDir, OUTPUT);

    pinMode(imanPin, OUTPUT);
```

```

digitalWrite(imanPin, LOW); // Imán apagado

pinMode(sensor1Pin, INPUT_PULLUP);
pinMode(sensor2Pin, INPUT_PULLUP);
}

// Ajusta velocidad del motor (más lento y estable)
void moveStepper(float turns, int dir) {
    digitalWrite(motorPinDir, dir);
    int steps = abs(turns * 200);
    for (int i = 0; i < steps; i++) {
        digitalWrite(motorPinStep, HIGH);
        delayMicroseconds(1000); // Más lento
        digitalWrite(motorPinStep, LOW);
        delayMicroseconds(1000);
    }
}

void iniciarCronometro() {
    tiempoInicio = millis();
    cronometroActivo = true;
}

void detenerCronometro() {
    tiempoTranscurrido = millis() - tiempoInicio;
    cronometroActivo = false;
    Serial.print("Tiempo transcurrido: ");
    Serial.print(tiempoTranscurrido);
    Serial.println(" ms");
}

void loop() {
    // Sensado condicionado al estado
    if (sensoresActivos && !yaDetectado) {
        int estadoSensor1 = digitalRead(sensor1Pin);
        int estadoSensor2 = digitalRead(sensor2Pin);

        if (estadoSensor1 == LOW && estadoAnteriorSensor1 == HIGH) {
            Serial.println("Sensor 1 activado");
            if (!cronometroActivo) {
                iniciarCronometro();
            }
        }

        if (estadoSensor2 == LOW && estadoAnteriorSensor2 == HIGH) {

```

```

    Serial.println("Sensor 2 activado");
    if (cronometroActivo) {
        detenerCronometro();
        yaDetectado = true; // No seguir midiendo
    }
}

// Desactivar automáticamente si pasa el tiempo
if (millis() - tiempoSensorInicio > duracionSensado) {
    sensoresActivos = false;
    Serial.println("Tiempo de sensado finalizado.");
}

estadoAnteriorSensor1 = estadoSensor1;
estadoAnteriorSensor2 = estadoSensor2;
}

// Comando serial desde MATLAB
if (Serial.available() > 0) {
    String comando = Serial.readStringUntil('\n');
    comando.trim(); // Eliminar espacios

    if (comando == "subir") {
        float vueltas = 5.0;
        if (totalVueltas + vueltas > limiteSuperiorVueltas) {
            vueltas = limiteSuperiorVueltas - totalVueltas;
        }
        totalVueltas += vueltas;
        moveStepper(vueltas, LOW); // LOW = sentido horario
        Serial.println("Motor subiendo");
        //Serial.print("Total vueltas actuales: ");
        //Serial.println(totalVueltas);

    } else if (comando == "bajar") {
        float vueltas = -4.0;
        if (totalVueltas + vueltas < limiteInferiorVueltas) {
            vueltas = limiteInferiorVueltas - totalVueltas;
        }
        totalVueltas += vueltas;
        moveStepper(vueltas, HIGH); // HIGH = sentido antihorario
        Serial.println("Motor bajando");
        //Serial.print("Total vueltas actuales: ");
        //Serial.println(totalVueltas);

    } else if (comando == "activar") {

```

```
digitalWrite(imanPin, HIGH);
Serial.println("Iman ACTIVADO");

} else if (comando == "desactivar") {
    digitalWrite(imanPin, LOW);
    Serial.println("Iman DESACTIVADO");

} else if (comando == "iniciar") {
    digitalWrite(imanPin, HIGH);
    Serial.println("Iman activado por 5 segundos...");
    delay(5000);
    digitalWrite(imanPin, LOW);
    Serial.println("Iniciando sensado entre sensores...");

    sensoresActivos = true;
    yaDetectado = false;
    tiempoSensorInicio = millis();
    cronometroActivo = false;

} else {
    Serial.println("Comando desconocido.");
}

}
```

### 12.3. Archivos Python

- Archivo registrar\_rostro.py

```
import face_recognition
import cv2
import pickle
import sys
import os

# Obtener nombre desde argumento
if len(sys.argv) < 2:
    print("Debes proporcionar un nombre como argumento.")
    sys.exit(1)

nombre = sys.argv[1]

# Verifica si ya existe el archivo con embeddings
archivo_embeddings = "embeddings.pkl"
if os.path.exists(archivo_embeddings):
```

```

        with open(archivo_embeddings, "rb") as f:
            embeddings = pickle.load(f) # Diccionario existente
    else:
        embeddings = {}

# Captura de rostro
cap = cv2.VideoCapture(0)
print("Presiona 's' para capturar tu rostro")

while True:
    ret, frame = cap.read()
    if not ret:
        continue
    cv2.imshow("Registrar Rostro", frame)
    if cv2.waitKey(1) & 0xFF == ord('s'):
        break

cap.release()
cv2.destroyAllWindows()

# Procesar imagen
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
locations = face_recognition.face_locations(rgb)
encodings = face_recognition.face_encodings(rgb, locations)

# Verifica si se detectó un rostro
if encodings:
    embeddings[nombre] = encodings[0] # Guardar bajo el nombre
    with open(archivo_embeddings, "wb") as f:
        pickle.dump(embeddings, f)
        print(f"Rostro de '{nombre}' registrado correctamente.")
else:
    print("No se detectó rostro. Intenta de nuevo.")

•

```

- Archivo reconocer\_rostro.py

```

# reconocer_rostro.py
import face_recognition
import cv2
import pickle
import sys

```

```

# Cargar todos los embeddings y nombres
with open("D:\\Tareas USFX\\2024\\Ing. Electronica\\ProyectoFinal Caida libre\\embeddings.pkl", "rb")
as f:
    data = pickle.load(f) # {'Juan': encoding1, 'Maria': encoding2, ...}

known_names = list(data.keys())
known_embeddings = list(data.values())

cap = cv2.VideoCapture(0)
match_found = False
matched_name = "Desconocido"

for _ in range(30):
    ret, frame = cap.read()
    if not ret:
        continue
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    locations = face_recognition.face_locations(rgb)
    encodings = face_recognition.face_encodings(rgb, locations)

    for encoding in encodings:
        distances = face_recognition.face_distance(known_embeddings, encoding)
        min_distance = min(distances)
        best_match_index = distances.tolist().index(min_distance)

        if min_distance < 0.45:
            match_found = True
            matched_name = known_names[best_match_index]
            break
    if match_found:
        break

cap.release()

# Salida para MATLAB
if match_found:
    print(f"1:{matched_name}") # Ej: "1:Juan"
else:
    print("0")

```

- Archivo reconocer\_comando.py

```

import speech_recognition as sr
import sys
import os

def procesar_audio(archivo):

```

```
r = sr.Recognizer()
try:
    with sr.AudioFile(archivo) as source:
        # Ajuste avanzado de ruido (recomendado)
        r.adjust_for_ambient_noise(source, duration=0.8)
        audio = r.record(source, duration=2) # Limita a 2 segundos

    # Configuración para español con timeout
    texto = r.recognize_google(audio,
                               language="es-ES",
                               show_all=False)
    return texto.lower()

except sr.UnknownValueError:
    return "error_ruido"
except sr.RequestError:
    return "error_conexion"
except Exception as e:
    return f"error_interno: {str(e)}"

if __name__ == "__main__":
    if len(sys.argv) > 1 and os.path.exists(sys.argv[1]):
        print(procesar_audio(sys.argv[1]))
    else:
        print("error_archivo")
```