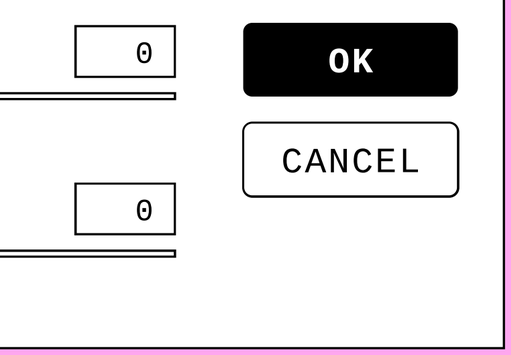
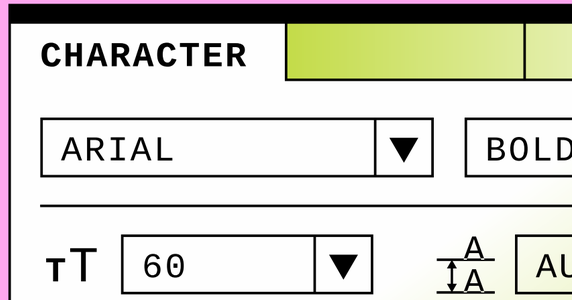


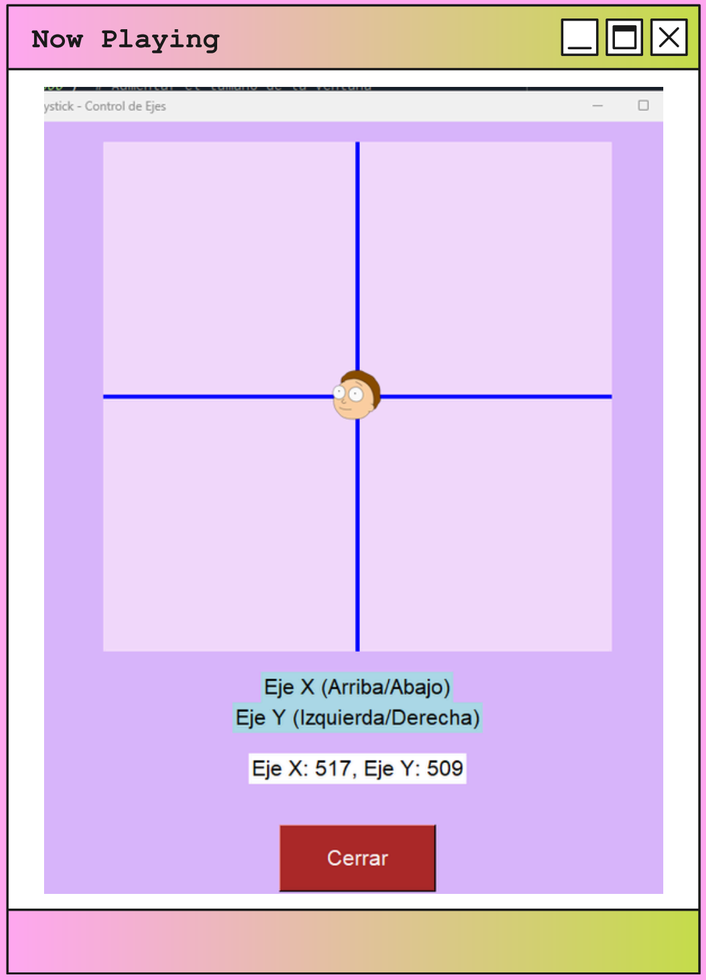
UACJ

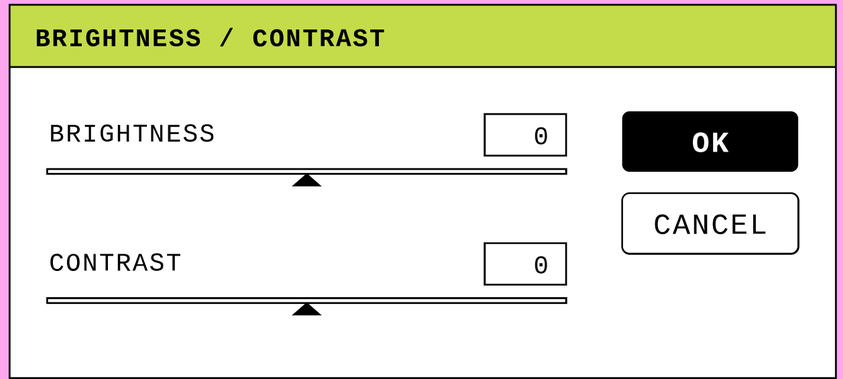
Interfaz Gráfica con Arduino y Joystick en Python

Diseño de Interfaces II



- Adquisición de datos analógicos con Arduino (pyFirmata).
- Procesamiento de señales en Python.
- Comunicación serial entre Arduino y PC (Firmata).
- Renderizado gráfico en Tkinter con PIL.
- Conversión y escalamiento de datos para una interfaz visual.



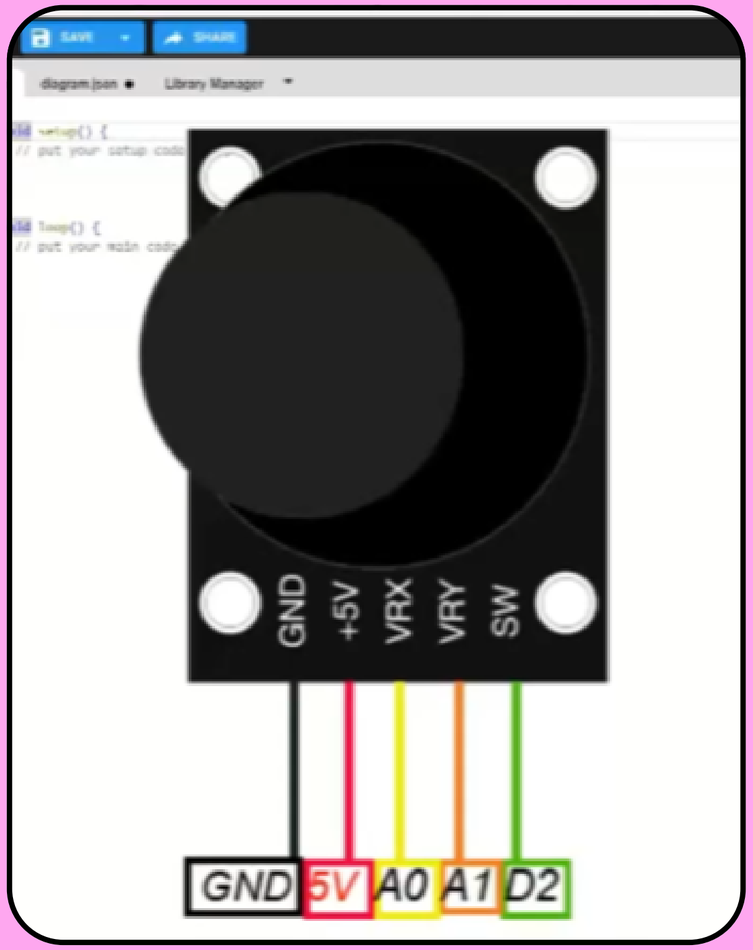


Conexiones Fisicas

1. Adquisición de Datos desde el Joystick
Un joystick analógico genera dos señales eléctricas, una para cada eje de movimiento (X y Y), las cuales se traducen en valores analógicos que son digitalizados por el Conversor Analógico-Digital (ADC) del Arduino.

📌 Conversión ADC
El ADC de Arduino tiene una resolución de 10 bits, lo que significa que convierte los valores de 0V a 5V en valores discretos en el rango de 0,10230, 10230, 1023:

$$V_{salida} = \frac{V_{entrada}}{5V} \times 1023$$



- VCC → 5V Arduino
- GND → GND Arduino
- Eje X → A0 (Entrada analógica)
- Eje Y → A1 (Entrada analógica)
- Botón SW → D2 (Entrada digital)

Inicialización de la Placa en Python

✓ Arduino('COM13') abre un puerto serie con Arduino.

✓ util.Iterator(board).start() previene bloqueos en la comunicación serial.

python Copiar Editar

```python
from pyfirmata import Arduino, util
board = Arduino('COM13')  # Configurar Arduino en el puerto correcto
it = util.Iterator(board)
it.start()
```

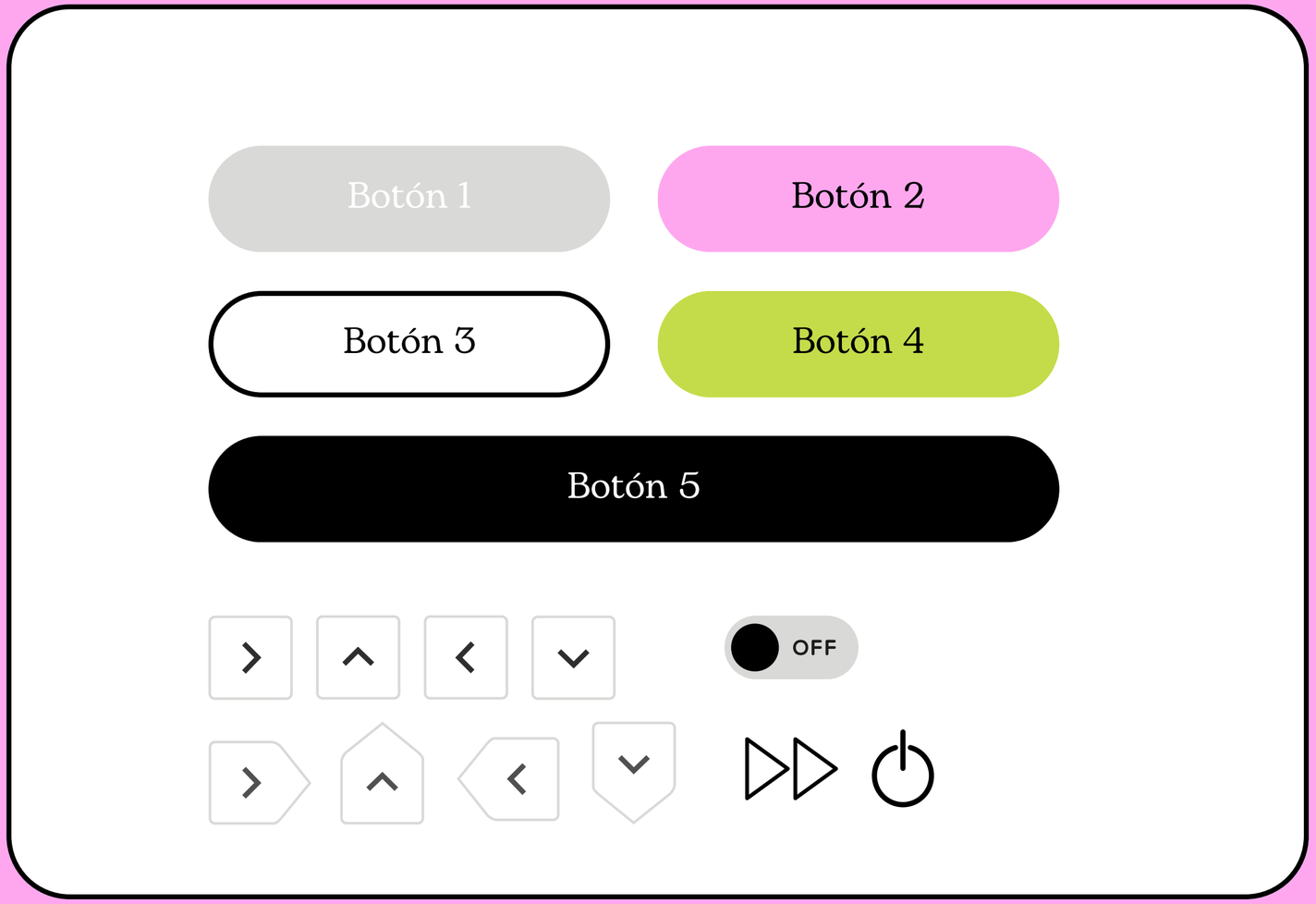
python Copiar Editar

```python
x_pin = board.get_pin('a:0:i')  # Eje X
y_pin = board.get_pin('a:1:i')  # Eje Y
x_pin.enable_reporting()
y_pin.enable_reporting()
```

✓ Se habilita la lectura de los pines analógicos del joystick.

✓ Los valores se actualizan en tiempo real con pyFirmata.

Creación de la Interfaz Gráfica en Tkinter

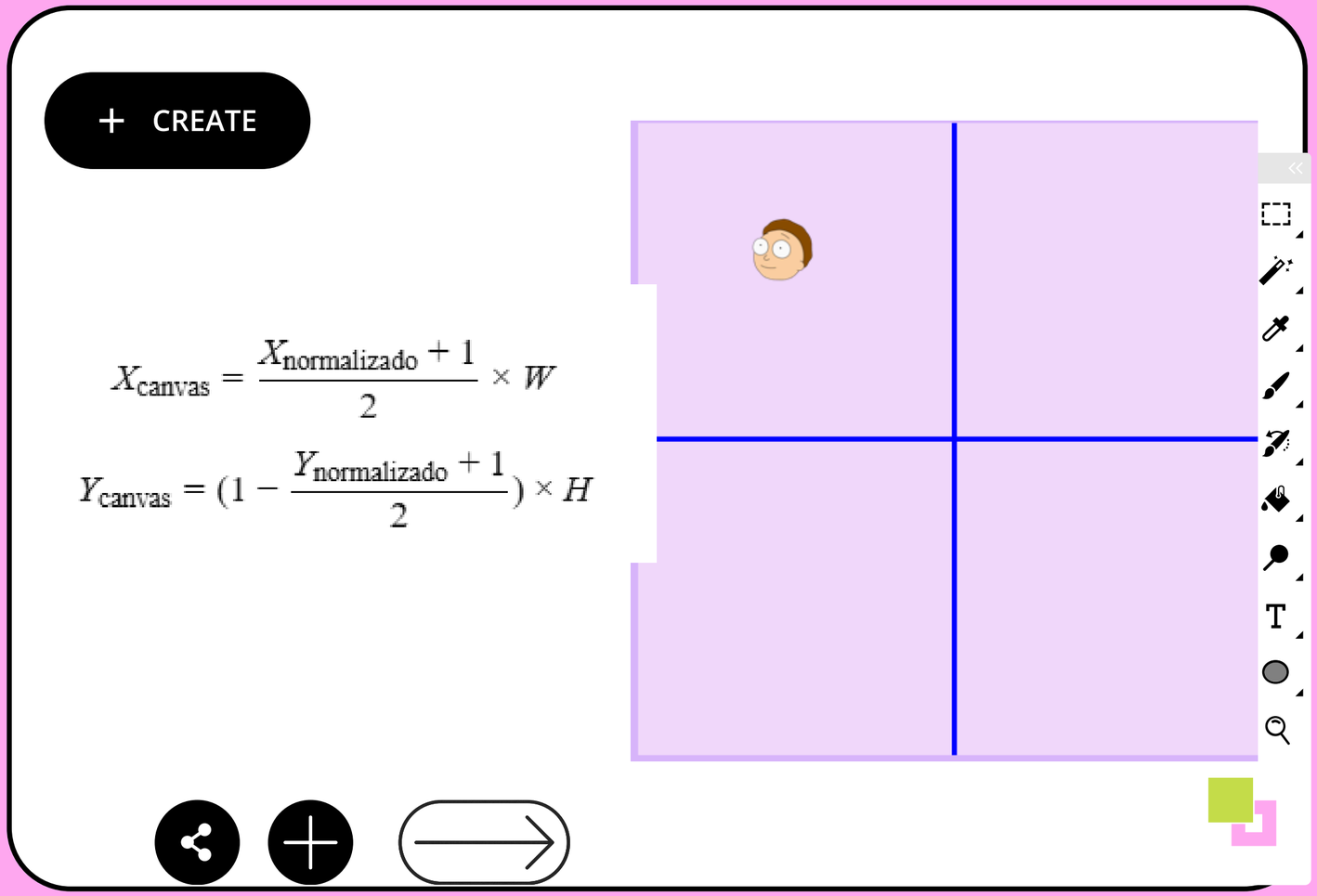


✓ Se define una ventana de 700x800 píxeles.

✓ Se asigna un color de fondo.

```python
import tkinter as tk
root = tk.Tk()
root.title("Joystick - Control de Ejes")
root.geometry("700x800")
root.configure(bg="#dab6fc")
```

Renderizado de la Cruz Cartesiana



$$X_{canvas} = \frac{X_{normalizado} + 1}{2} \times W$$

$$Y_{canvas} = (1 - \frac{Y_{normalizado} + 1}{2}) \times H$$

📌 Se usa un Canvas de 500x500 píxeles con una línea horizontal y una vertical para representar los ejes.

```python
canvas = tk.Canvas(root, width=500, height=500, bg="#f3dafc")
canvas.pack(pady=20)
canvas.create_line(250, 0, 250, 500, fill="blue", width=4)  # Línea vertical
canvas.create_line(0, 250, 500, 250, fill="blue", width=4)  # Línea horizontal
```

En la interfaz gráfica, los valores del joystick deben transformarse a coordenadas de pantalla en un canvas de Tkinter con dimensiones de W×H .

📌 Mapeo Lineal

Dado que los valores normalizados varían entre [-1 , 1] necesitamos escalarlos al rango de píxeles del canvas [0,W] y [0,H].

W,H son el ancho y alto del área de visualización.

Xcanvas, Ycanvas son las coordenadas en la interfaz gráfica.

La transformación en Y se invierte, ya que en sistemas gráficos la coordenada Y aumenta hacia abajo.

Modelado de la Respuesta del Joystick

Donde K es la ganancia estática (relación entre entrada y salida).
τ es la constante de tiempo del sistema (depende del circuito de filtrado y la mecánica del joystick).
s es la variable de Laplace.

El joystick puede modelarse como un sistema de dos grados de libertad con una respuesta mecánica y eléctrica que sigue un comportamiento lineal en un rango restringido.

📌 Modelo Matemático del Sistema

El joystick es un sistema mecánico-electrónico que puede representarse con una función de transferencia de primer orden

OOPS!

$$H(s) = \frac{K}{\tau s + 1}$$

RESPONDING.

CRY

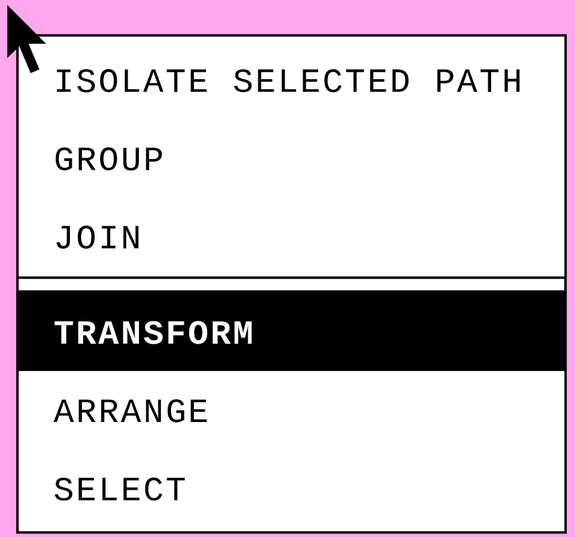
La respuesta del sistema en términos de desplazamiento sigue una ecuación diferencial de la forma:

$$\tau \frac{dX}{dt} + X = KU$$

donde U es la señal de entrada del usuario.
Si el joystick se mueve rápidamente, el sistema puede sobrecorregir debido a la inercia mecánica.

📌 7. Visualización de la Cabeza de Morty

```python
from PIL import Image, ImageTk
original_image = Image.open("C:/Users/diana/Pyton_microcontroladores/ImagenesInterfaces/morty
resized_image = original_image.resize((50, 50))  # Mantener el tamaño original
morty_image = ImageTk.PhotoImage(resized_image)
marker = canvas.create_image(250, 250, image=morty_image)  # Posición inicial
```



🔹 PIL (Pillow) es una biblioteca para el procesamiento de imágenes en Python.
🔹 Image es un módulo de Pillow que permite cargar y modificar imágenes.
🔹 ImageTk permite convertir imágenes de Pillow a un formato compatible con Tkinter, lo cual es necesario para mostrarlas en un Label o Canvas

- La imagen se almacena en la variable original_image como un objeto PIL.Image.

1 Se importan las bibliotecas necesarias (Image y ImageTk).

2 Se carga la imagen desde el almacenamiento con Image.open().

3 Se redimensiona la imagen con resize().

4 Se convierte la imagen al formato de Tkinter con ImageTk.PhotoImage().

5 Se muestra la imagen en un Canvas en la posición (250, 250).

Interfaz

El uso de Tkinter en conjunto con Pillow (PIL) y PyFirmata permite desarrollar interfaces interactivas para el control de hardware como Arduino, integrando elementos gráficos avanzados como imágenes en movimiento, interacción con joysticks y retroalimentación visual en tiempo real.

En este caso, analizamos el flujo completo de carga, manipulación y visualización de imágenes en un Canvas de Tkinter, lo que facilita la integración de representaciones gráficas dinámicas en proyectos de robótica e IoT. Además, mediante el uso de la librería PyFirmata, logramos una comunicación estable entre Python y Arduino, permitiendo un control preciso de los dispositivos conectados.

Puntos Clave:

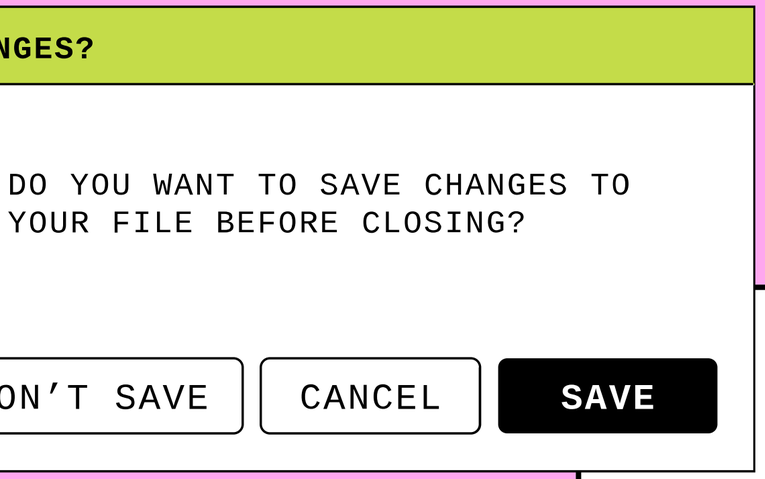
✓ Modularidad: Separar la carga y manipulación de imágenes facilita la reutilización del código.

✓ Optimización: Redimensionar imágenes con interpolación adecuada mejora la calidad sin sacrificar rendimiento.

✓ Interactividad: La actualización en tiempo real mediante root.after() o itemconfig() permite crear sistemas reactivos.

```python
import tkinter as tk
from pyfirmata import Arduino, util
from PIL import Image, ImageTk  # Para redimensionar la imagen

# Configuración de Arduino
board = Arduino('COM13')  # Cambia 'COM4' por el puerto correcto
it = util.Iterator(board)
it.start()

# Configurar los pines del joystick
x_pin = board.get_pin('a:0:i')  # Eje X conectado al pin A0
y_pin = board.get_pin('a:1:i')  # Eje Y conectado al pin A1
x_pin.enable_reporting()
y_pin.enable_reporting()

# Configuración de la ventana
root = tk.Tk()
root.title("Joystick - Control de Ejes")
root.geometry("700x800")  # Aumentar el tamaño de la ventana
root.configure(bg="#dab6fc")

# Canvas para la cruz y el marcador
canvas = tk.Canvas(root, width=500, height=500, bg="#f3dafc", highlightthickness=0)  # Aumentar el tamaño del Canvas
canvas.pack(pady=20)

# Dibujar la cruz más grande
canvas.create_line(250, 0, 250, 500, fill="blue", width=4)  # Línea vertical más grande
canvas.create_line(0, 250, 500, 250, fill="blue", width=4)  # Línea horizontal más grande

# Cargar la imagen de Morty y mantener su tamaño
original_image = Image.open("C:/Users/diana/Pyton_microcontroladores/ImagenesInterfaces/morty.png")
resized_image = original_image.resize((50, 50))  # Mantener el tamaño original
morty_image = ImageTk.PhotoImage(resized_image)
marker = canvas.create_image(250, 250, image=morty_image)  # Centrar en el canvas más grande

# Etiquetas de los ejes más grandes
label_x = tk.Label(root, text="Eje X (Arriba/Abajo)", bg="#aad8e6", font=("Arial", 16))  # Fuente más grande
label_x.pack()

label_y = tk.Label(root, text="Eje Y (Izquierda/Derecha)", bg="#aad8e6", font=("Arial", 16))  # Fuente más grande
label_y.pack()
```

Realiza un ejemplo con
una imagen propia!!



Gracias

Python