# Manual del Programador

## Proyecto: Traductor Bidireccional Braille-español

## Versión 2

## Equipo: KDR CONSULTECH

## Fecha: 30 de junio de 2024

**MANUAL DEL PROGRAMADOR**

# Tabla de contenido



## Descripción del proyecto

Este programa de traducción de Braille es una aplicación bidireccional que permite convertir texto en español a caracteres Braille y viceversa. Además, ofrece la capacidad de generar archivos PDF e imágenes en modo espejo con la respectiva traducción en Braille. El proyecto utiliza diccionarios para mapear los caracteres alfabéticos y numéricos junto a sus correspondientes símbolos en Braille, garantizando una traducción precisa y eficiente.

Una de las funcionalidades destacadas de la aplicación es su capacidad para reconocer y procesar comandos de voz utilizando la biblioteca Vosk. Esta funcionalidad permite que los usuarios puedan dictar texto en español, que luego se transcribe y convierte a Braille. De igual manera, la aplicación puede interpretar texto en Braille y convertirlo a español mediante comandos de voz, mejorando significativamente la accesibilidad y la usabilidad para personas con discapacidades visuales.

## Estructura del código

El código se organiza en varios archivos y módulos que trabajan en conjunto para la traducción de texto a Braille y viceversa, así como para la generación de archivos PDF e imágenes. A continuación, se describen los componentes principales:

- T2B_code.py: maneja la lógica de traducción de texto a Braille y se define un diccionario llamado ‘mapeo_braille’ para mapear caracteres individuales y números a su representación en braille.
- B2T_form_design.py: este módulo define la ventana para la traducción de Braille a texto por voz.
- T2B_form_design.py: contiene métodos para crear la interfaz de usuario y gestionar la traducción de texto a Braille y la generación de archivos e imágenes.

- Main_form_design.py: define la Ventana principal de la aplicación, incluyendo la barra superior, barra lateral y el panel de contenido principal, además, de permitir la navegación entre los diferentes formularios de la aplicación.
- convertTo.py: este módulo proporciona la funcionalidad para convertir texto en Braille en diferentes formatos: PDF utilizando Tkinter para la interfaz gráfica y ReportLab para la generación de PDF.
- audio_recorder.py: Maneja la grabación de audio, utilizando la biblioteca threading para gestionar la grabación en segundo plano.
- vosk_recognition.py: Utiliza la biblioteca Vosk para el reconocimiento de voz, permitiendo la transcripción de audio grabado a texto en español.
- BrailleApp_KDR.py: es la entrada principal del programa, inicializa la aplicación y ejecuta el ciclo principal de eventos de la interfaz gráfica.

## Requisitos del entorno

- Python en la versión 3.11 o superior
- Tkinter, biblioteca necesaria para la creación de interfaces graficas de usuario.
- **CustomTkinter:** Biblioteca que mejora la apariencia y funcionalidad de Tkinter, proporcionando widgets adicionales y temas personalizables.
- Reportlab: biblioteca necesaria para crear documentos PDFs.
- Pillow(PIL): biblioteca para abrir, guardar y manipular imágenes en variedad de formato, PNG,JPGE, JPG, entre otros.
- **Vosk:** Biblioteca de reconocimiento de voz que se utiliza para transcribir audio a texto.
    - **Modelo de Vosk:** Se requiere un modelo específico para el reconocimiento de voz en español.
- **Threading:** Módulo de Python para manejar la grabación de audio en segundo plano, proporcionando una experiencia de usuario sin interrupciones.

## Instalaciones necesarias

- Clonar el repositorio del proyecto desde gitHub
- Instalar las dependencias necesarias utilizando pip:
    - pip install reportlab
    - pip install pillow
    - pip install customtkinter
    - pip install vosk

- Descargar el modelo de reconocimiento de voz de Vosk para español desde el link directo

  https://alphacephei.com/vosk/models/vosk-model-small-es-0.42.zip

- Ejecuta la aplicación:
  BrailleApp_KDR.py

## Detalle de métodos

*obtener_texto(self,texto):*
Divide la cadena de texto en palabras y almacena las posiciones de los asteriscos en la variable global `pos`.

Args: texto (str): El texto a ser dividido en palabras.

```
def obtener_texto(self, texto):
        i = 0
        j = 0
    #     # Dividir la cadena en palabras
        palabras = []

        while i < len(texto):

            if texto[i] == '*':
                palabras.append(i)
                j = -1
            j += 1
            i += 1

        print(palabras)

        global pos
        pos = palabras
```

*texto_a_braille(self, raw_texto):*
Convierte un texto en español a su equivalente en Braille utilizando un mapeo de caracteres.
str: El texto convertido a Braille.

```
def texto_a_braile(self, raw_texto):
        texto = raw_texto

        texto = texto.replace('\t', ' ' * 4)
        braille = ''

        i = 0
        while i < len(texto):
            char = texto[i]
            if char.isupper():
                braille += '⠠' + mapeo_braille.get(char.lower(), char)
            elif char.isdigit():
                braille += '⠼'
                while i < len(texto) and texto[i].isdigit():
                    braille += mapeo_braille.get(texto[i], texto[i])
                    i += 1
                continue
            else:
                braille += mapeo_braille.get(char, char)

            i += 1

        global final_braille
        final_braille = braille
        return braille
```

*Clase ConvertTo*
Esta clase maneja la conversión del texto en Braille a un archivo PDF y ofrece opciones para guardarlo en modo normal o espejo.

*__init__(self)*
Constructor de la clase ConvertTo. Inicializa los atributos y configura el entorno necesario para las conversiones

```
def __init__(self):
        print("convert")
        self.raw_braille = ''
        self.pos = []
        self.ruta_fuente = UtilPath().get_font_path()
        self.ruta_modelo = UtilPath().get_model_path()
```

```
        self.set_font()
        self.recognizer = VoskRecognizer(self.ruta_modelo)
```

*generar_pdf_espejo(self):*
Genera un archivo PDF con el texto en braille en modo espejo.

Args:
Texto convertido a braille.
Nombre del archivo PDF a generar.

```
def generar_pdf_espejo(self):

        archivo = self.get_save_name('0')
        self.raw_braille = self.get_raw_brallie()

        # Crear un objeto canvas para el documento PDF
        if archivo:
            nombre = archivo

            # Definir el tamaño de la página
            ancho, alto = letter

            # Inicializar el lienzo (canvas)
            c = canvas.Canvas(nombre, pagesize=letter)

            # Configurar la fuente personalizada
            c.setFont("Braille", 20)

            # Configurar la impresión en modo espejo horizontalmente
            c.transform(-1, 0, 0, 1, letter[0], 0)
            text_object = c.beginText(40, letter[1] - 40)

            lines = self.raw_braille.split('\n')
            # Dividir el texto en líneas para que quepa en la página
            print(lines)
            for linea in lines:
                # Calculo numero de paginas necesarias.
                wrapped_lines = self.wrap_text(linea, ancho - 70, c)
                for wrapped_line in wrapped_lines:
                    if text_object.getY() < 40:
                        c.drawText(text_object)
                        c.showPage()
                        c.setFont("Braille", 20)
                        c.transform(-1, 0, 0, 1, letter[0], 0)
                          text_object = c.beginText(40, letter[1] - 40)
                      # Escribir el texto en el lienzo
                      text_object.textLine(wrapped_line)
             c.drawText(text_object)
             c.showPage()

             # Guardar el PDF
             c.save()

             if c:
                 self.succesful_save()
```

*Wrap_text(self,text , max_width,c):*
Divide el texto en líneas para ajustarse al ancho de la página.

- text (str): El texto a dividir.
- max_width (int): El ancho máximo de la línea.
- c (canvas): El lienzo de ReportLab

```
# Dividir el texto en líneas para que quepa en la página
        wrapped_lines = []
        words = text.split(' ')
        while words:
            linea = ""
            while words and c.stringWidth(linea + words[0], "Braille", 20) <=
max_width:
                linea += words.pop(0) + " "
            wrapped_lines.append(linea)
        return wrapped_lines
```

*convert_2_image(self)*
Convierte el texto en Braille a una imagen.

```
def convert_2_image(self):
        BACKGROUND_COLOR = "white"
        text_color = "black"

        # Filediallog para preguntar ruta para guardar la imagen
        archivo = self.get_save_name('1')
        self.raw_braille = self.get_raw_brallie()

        if archivo:
            nombre = archivo.split('.png')
            print(nombre)

            ancho, alto = letter
            margen = 45
            max_lines_per_image = 30

            # Crear nueva imagen con el tama;o especificado y el color de
fondo
            imagen = Image.new("RGB", (int(ancho), int(alto)),
BACKGROUND_COLOR)

            # Crear un objeto ImageDraw para dibujar en la imagen
            draw = ImageDraw.Draw(imagen)

            # Definir la fuente y el tamaño del texto
            if self.ruta_fuente:
                # fuente = ImageFont.truetype(self.ruta_fuente, 20)
                try:
                    fuente = ImageFont.truetype(self.ruta_fuente, 20)
                except IOError:
                    fuente = ImageFont.load_default()

                # Dividir el texto en líneas para que quepa en la imagen
                lineas = self.raw_braille.split('\n')

                # Definir la posición inicial para escribir el texto
```

```
                y_inicial = margen

                img_count = 1
                line_count = 0

                print(lineas)
                # Escribir el texto en la imagen
                for linea in lineas:
                    # draw.text((x_inicial, y_inicial), linea, fill =
text_color, font = fuente)
                    # y_inicial += 20
                    wrapped_lines = self.split_text_to_fit_line(linea,
int(ancho) - 2 * margen, draw, fuente)
                    for wrapped_line in wrapped_lines:
                        if line_count >= max_lines_per_image:  # Si se excede
el número máximo de líneas por imagen
                            # Guardar la imagen actual y crear una nueva
                            imagen.save(f"{nombre[0]}_parte_{img_count}.png")
                            img_count += 1
                            line_count = 0
                            y_inicial = margen
                            imagen = Image.new('RGB', (int(ancho),
int(alto)), color=(255, 255, 255))
                            draw = ImageDraw.Draw(imagen)

                        draw.text((margen, y_inicial), wrapped_line,
font=fuente, fill=(0, 0, 0))
                        y_inicial += 23
                        line_count += 1


                # Guardar la última imagen si hay alguna línea dibujada
                if line_count > 0:
                    imagen.save(f"{nombre[0]}_parte_{img_count}.png")

                if imagen:
                    self.succesful_save()
```

*split_text_to_fit_line(self, text, max_width, draw, font):*
Divide el texto en varias líneas si es más ancho que el ancho máximo especificado.

- text (str): El texto a dividir.
- max_width (int): El ancho máximo de la línea.
- draw (ImageDraw): El objeto de dibujo de PIL.
- font (ImageFont): La fuente utilizada para el texto.

```
def split_text_to_fit_line(self, text, max_width, draw, font):
    # Divide el texto en varias líneas si es más ancho que max_width.
        lines = []
        words = text.split(' ')
        print('palabras')
        print(words)
        current_line = ""
        is_first_word = True

        for word in words:
```

```
            if is_first_word and word == '':
                current_line = ' '
                is_first_word = False
                continue
            elif word == '':
                current_line += '\n'
            else:
                test_line = current_line + word if current_line else word
                width = draw.textlength(test_line, font=font)
                if width <= max_width:
                    current_line = test_line + ' '
                else:
                    lines.append(current_line)
                    current_line = word + ' '
            is_first_word = False
        if current_line:
            lines.append(current_line)
        return lines
```

*voice_to_braille(self):*
Convierte el audio transcrito a texto en Braille.

```
def voice_to_braille(self):
        self.transcribed_text = self.recognizer.transcribe_audio()
        if self.transcribed_text:
            self.raw_braille =
T2BCode().texto_a_braile(self.transcribed_text)
            messagebox.showinfo("Transcription", f"Transcribed Text:
{self.transcribed_text}\nBraille: {self.raw_braille}")
        else:
            messagebox.showwarning("Transcription", "No se pudo transcribir
el audio.")
```

*get_transcribed_text(self):*
Obtiene el texto transcrito desde el audio.

```
 def get_transcribed_text(self):
        return self.transcribed_text
```

*set_font(self)*
Registra una fuente TrueType con la librería ReportLab para usarla en los documentos PDF.

```
 def set_font(self):
         # Registrar un TrueType font con la libreria ReportLab de
pdfmetrics
         pdfmetrics.registerFont(TTFont('Braille',
self.ruta_fuente))
```

*get_save_name(self)*
Abre un diálogo para que el usuario elija la ubicación y el nombre del archivo donde se guardará el PDF.

Retorno: Cadena con la ruta y el nombre del archivo seleccionado por el usuario.

```
    def get_save_name(self):
        return filedialog.asksaveasfilename(
            defaultextension = ".*", title = "Save File", filetypes =
(("PDF Files", "*.pdf"), ("Text Files", "*.txt"))
        )
```

*succesful_save(self)*
Muestra un mensaje de éxito cuando el archivo PDF se guarda correctamente.

```
    def succesful_save(self):
        messagebox.showinfo("Success", "Archivo guardado con exito")
```

## about_form_Design.py

*__init__(self) -> None*
Inicializa la ventana llamando a los métodos config_window y create_widgets para configurar la ventana y crear los widgets necesarios.

```
def __init__(self) -> None:
        super().__init__()
        self.config_window()
        self.create_widgets()
```

*config_window(self)*
Configura la ventana principal, estableciendo el título y centrando la ventana en la pantalla.
Se establece el título de la ventana a "About", define el tamaño de la ventana (w, h), y posteriormente, llama a la función centrar_ventana del módulo util_vent para centrar la ventana.

```
def config_window(self):
        # configuracion incial de la ventana
        self.title("About")
        w, h = 400, 100
        self.geometry(f'{w}x{h}')
        self.resizable(False, False)
        self.grab_set()
        util_vent.centrar_ventana(self, w, h)
```

*create_widgets(self)*
Crea y configura los widgets dentro de la ventana.

Primero, crea un Label para mostrar la versión del software, configura las propiedades del Label (color de texto, fuente, padding, ancho) y empaqueta el Label en la ventana. Después, crea otro Label para mostrar el autor del software y configura las propiedades del Label (color de texto, fuente, padding, ancho) para volver a empaqueta el Label en la ventana.

```
def create_widgets(self):
        # configuracion de los widgets
        self.labelVersion = ctk.CTkLabel(
            self, text = "Version : 2.0"
```

```
        )
        self.labelVersion.configure(
            font = FONT_AWSOME_20
        )
        self.labelVersion.pack(pady = 10)

        self.labelAutor = ctk.CTkLabel(
            self, text = "Autor : KDR CONSULTECH"
        )
        self.labelAutor.configure(
            font = FONT_AWSOME_20

        )
        self.labelAutor.pack(pady = 10)
```

## T2B_form_design.py

*__init__(self, main_panel)*
Inicializa la interfaz gráfica llamando a los métodos create_frames, create_top_widgets, create_center_widgets y create_bottom_widgets para configurar la ventana y crear los widgets necesarios.

```
def __init__(self, main_panel):
        self.traslator = T2BCode()
        self.converter = ConvertTo()
        # self.is_recording = False
        self.recorder = AudioRecorder()
        self.create_frames(main_panel)
        self.create_top_widgets()
        self.create_center_widgets()
        self.create_bottom_widgets()
```

*create_frames(self, main_panel)*
Crea y configura los marcos (frames) principales de la ventana.

Parámetros:
main_panel: El panel principal en el que se colocarán los marcos.

```
def create_frames(self, main_panel):
        self.top_frame = ctk.CTkFrame(main_panel)
        self.top_frame.pack(side='top', fill='both', expand=True)
        self.top_frame.pack(side='top', fill='both', expand=True)

        self.center_frame = ctk.CTkFrame(main_panel)
        self.center_frame.pack(side='top', fill='both', expand=True)
        self.center_frame.pack(side='top', fill='both', expand=True)

        self.bottom_frame = ctk.CTkFrame(main_panel)
        self.bottom_frame.pack(side='top', fill='both', expand=False)
        self.bottom_frame.pack(side='top', fill='both', expand=False)
```

*create_top_widgets(self)*
Crea y configura los widgets en el marco superior, incluyendo el cuadro de texto y los botones de traducción y limpieza.

```
      ancho = 20
        alto = 1

        self.label_input = ctk.CTkLabel(self.top_frame, text="Texto a
Convertir", font=FONT_AWSOME_20)
        self.label_input.pack(pady=5, side='top', fill='both',
expand=False)

        # Crear Entry widget
        self.textBox_input = ctk.CTkTextbox(
            self.top_frame, font=FONT_ARIAL_15, fg_color=FG_TEXTBOX, wrap =
'word'
        )
        self.textBox_input.pack(padx=100, pady=5, side='top', fill='both',
expand=True)
        self.textBox_input.bind("<<Modified>>", self.trad_2_braille)
        self.textBox_input.edit_modified(False)
```

*create_center_widgets(self):*
Crea y configura los widgets en el frame central.

```
 def create_center_widgets(self):
        self.label_output = ctk.CTkLabel(
            self.center_frame, text="Texto en Braille", font=FONT_AWSOME_20
        )
        self.label_output.pack(pady=5, side='top', fill='both', expand=False)

        # Crear textbox de salida
        self.textBox_output = ctk.CTkTextbox(
            self.center_frame, font=FONT_ARIAL_15, fg_color=FG_TEXTBOX, wrap
= 'word', state='disabled'
        )
        self.textBox_output.pack(padx=100, pady=5, side='top', fill='both',
expand=True)

        # Deshabilitar eventos de teclado y mouse en el textbox de salida
        self.desabilitar_eventos_textbox()
```

*disable_event(self, event):*
Maneja los eventos deshabilitados, impidiendo cualquier acción.
event (tk.Event): El evento deshabilitado.
str: Retorna "break" para evitar la propagación del evento.

```
def disable_event(self, event):
        return "break"
```

*create_bottom_widgets(self)*
Crea y configura los widgets en el marco inferior, incluyendo los botones para generar archivos PDF.

```
# Crear botones
        self.button_clear_box = ctk.CTkButton(self.bottom_frame)
```

```
        self.button_img = ctk.CTkButton(self.bottom_frame)
        self.button_espejo = ctk.CTkButton(self.bottom_frame)
        self.button_start_recording = ctk.CTkButton(self.bottom_frame,
command=self.start_recording)
        self.button_stop_recording = ctk.CTkButton(self.bottom_frame,
command=self.stop_recording, state='disabled')
        self.button_copy_braille = ctk.CTkButton(self.bottom_frame,
text="Copiar Braille", command=self.copy_braille)

        buttons_info = [
            ("Limpiar", self.button_clear_box, "\uf00d", self.clear_textbox),
            ("IMG", self.button_img, "\uf1c5", self.to_img_normal),
            ("PDF", self.button_espejo, "\uf1c1", self.to_pdf_espejo),
            ("Por Voz",self.button_start_recording,"\uf130",
self.start_recording),
            ("Detener Grabación",self.button_stop_recording,"\uf04d",
self.stop_recording),
            ("Copiar", self.button_copy_braille, "\uf0c5", self.copy_braille)
        ]

        for text, button, icon, cm in buttons_info:
            ancho = 20
            alto = 1
            self.bottom_buttons_config(button, text, icon, FONT_ROBOTO_15,
ancho, alto, cm)

        self.button_start_recording.pack(padx=25, pady=5, side='left',
fill='y', expand=True)
        self.button_stop_recording.pack(padx=25, pady=5, side='left',
fill='y', expand=True)
```

*trad_2_braille(self,event):*
Traduce el texto en el textbox de entrada a Braille y lo muestra en el textbox de salida.
event (tk.Event): El evento que activa la traducción.

```
def trad_2_braille(self, event):
        new_text = self.get_text()
        final_text = self.traslator.texto_a_braile(new_text)
        self.textBox_output.configure(state='normal')
        self.textBox_output.delete("1.0", 'end-1c')
        self.textBox_output.insert("1.0", final_text)
        self.textBox_output.configure(state='disabled')
        self.textBox_input.edit_modified(False)
```

*clear_Textbox(self):*
Limpia el contenido de los textboxes de entrada y salida.

```
 def clear_textbox(self):
        self.textBox_input.delete("1.0", 'end')
        self.textBox_output.delete("1.0", 'end')
        self.traslator.set_final_braille()
```

*start_recording(self):*
Inicia la grabación de audio para convertir voz a texto.

```
def start_recording(self):
        # self.is_recording = True
        self.button_start_recording.configure(state='disabled')
        self.button_stop_recording.configure(state='normal')
        self.recorder.stop_event.clear()
        self.recorder.record_audio()
```

*stop_recording(self):*
Detiene la grabación de audio y procesa el audio grabado.

```
        self.button_start_recording.configure(state='normal')
        self.button_stop_recording.configure(state='disabled')
        self.recorder.stop_recording()
        self.converter.voice_to_braille()
        self.process_recorded_audio()
```

*process_recorded_audio(self):*
Procesa el audio grabado, convierte a texto y actualiza el textbox de entrada.

```
def process_recorded_audio(self):
        transcribed_text = self.converter.get_transcribed_text()
        if transcribed_text:
            self.textBox_input.delete("1.0", 'end')
            self.textBox_input.insert("1.0", transcribed_text)
            # self.trad_2_braille(None)
        else:
            messagebox.showwarning("Transcription", "No se pudo transcribir
el audio.")
```

*copy_braille(self):*
Copia el contenido en Braille del textbox de salida al portapapeles.

```
def copy_braille(self):
        self.textBox_output.configure(state='normal')
        self.textBox_output.clipboard_clear()
        self.textBox_output.clipboard_append(self.textBox_output.get("1.0",
'end-1c'))
        self.textBox_output.configure(state='disabled')
        messagebox.showinfo("Copiar Braille", "El texto en Braille ha sido
copiado.")
```

*get_text(self)*
Obtiene el texto del cuadro de texto.

```
 def get_text(self):
        return self.textbox.get(0.0, 'end')

    def get_text_braille(self, text):
        return self.trad_2_braille()
```

## Main_from_design.py

*__init__(self)*
Inicializa la ventana principal llamando a los métodos config_window, paneles, barra_sup_ctrl y menu_lat_ctrl para configurar la ventana y crear los componentes necesarios.

```
def __init__(self):
        super().__init__()
        self.config_window()
        self.paneles()
        self.barra_sup_ctrl()
        self.menu_lat_ctrl()
        self.open_T2B()
```

*config_window(self)*
Configura la ventana principal con un título y dimensiones específicas.

```
def config_window(self):
        #Configuracion inicial de la ventana
        w, h = 1024, 600
        self.title('Traductor de Braille')
        self.geometry(f'{w}x{h}')
        util_vent.centrar_ventana(self, w, h)

        # La ventana no puede hacerse más pequeña de lo inicial
        self.minsize(w, h)

        # Redimensionamiento solo para agrandar
        self.resizable(True, True)

    # La ventana no puede hacerse más pequeña de lo inicial
        self.minsize(w, h)

    # Redimensionamiento solo para agrandar
        self.resizable(True, True)
```

*paneles(self)*
Crea y configura los frames principales de la ventana, incluyendo la barra superior, el menú lateral y el panel de contenido.

```
def paneles(self):
        self.barra_sup = ctk.CTkFrame(self, height=60)
        self.barra_sup.pack(side = 'top', fill = 'both', expand = False)

        self.menu_lat = ctk.CTkFrame(self, width=150)
        self.menu_lat.pack(side = 'left', fill = 'both', expand = False)

        self.fondo = ctk.CTkFrame(self)
        self.fondo.pack(side = 'right', fill = 'both', expand = True)
```

*barra_sup_ctrl(self)*
Configura la barra superior con un título, un botón para mostrar/ocultar el menú lateral y una etiqueta de información.

```
def barra_sup_ctrl(self):
        # Configuracion de la barra superior
        # Etiqueta de titulo
        self.labelTituloLeft = ctk.CTkLabel(
            self.barra_sup, text = "MENÚ"
        )
        self.labelTituloLeft.configure(
            font = FONT_AWSOME_20, pady = 10, padx = 25, width = 175
            )
        self.labelTituloLeft.pack(side = 'left')

        # Boton del menu principal
        self.menu_button = ctk.CTkButton(
            self.barra_sup, width = 30, text = "\uf0c9", font =
FONT_ROBOTO_15,
            command = self.toggle_panel
        )
        self.menu_button.pack(side = 'left', pady = 10)

        # Etiqueta de informacion
        self.labelTituloRight = ctk.CTkLabel(
            self.barra_sup, text = "Version 1.0 - KDR"
        )
        self.labelTituloRight.configure(
            font = FONT_AWSOME_10, padx = 10, width = 25
            )
        self.labelTituloRight.pack(side = 'right')
```

*menu_lat_ctrl(self)*
Configura el menú lateral con botones para acceder a diferentes formularios de la aplicación.

```
def menu_lat_ctrl(self):
        # Configuracion de la barra lateral
        ancho_menu = 30
        alto_menu = 2

        # Botones del menu lateral
        self.T2BButton = ctk.CTkButton(self.menu_lat)
        self.B2TButton = ctk.CTkButton(self.menu_lat)
        self.aboutButton = ctk.CTkButton(self.menu_lat)

        buttons_info = [
            ("T2B", self.T2BButton, "\uf2a1", self.open_T2B),
            ("B2T", self.B2TButton, "\uf031", self.open_build),
            ("About", self.aboutButton, "\uf05a", self.open_about)
        ]

        for text, button, icon, cm in buttons_info:
            self.menu_button_conf(button, text, icon, FONT_ROBOTO_15,
ancho_menu, alto_menu, cm)
```

*menu_button_conf(self, button, text, icon, font_awesome, ancho, alto, cm)*
Configura los botones del menú lateral con texto, icono, fuente, tamaño y comando especificados.

```
def menu_button_conf(self, button, text, icon, font, ancho, alto, cm):
        button.configure(
            text = f"  {icon}\t{text}\t", anchor = "center", font = font,
width = ancho, height = alto,
            command = cm
        )
        button.pack(side = 'top', padx = 15, pady = 5)
```

*toggle_panel(self)*
Muestra u oculta el menú lateral.

```
def toggle_panel(self):
        # Cambiar el estado del panel lateral
        if self.menu_lat.winfo_ismapped():
            self.menu_lat.pack_forget()
            self.labelTituloLeft.pack_forget()
            self.menu_button.pack(padx = 10,  pady = 10)
        else:
            # self.menu_lat.pack(side = tk.LEFT, fill = tk.Y)
            self.menu_button.pack_forget()
            self.menu_lat.pack(side = 'left', fill = 'both', expand = False)
            self.labelTituloLeft.pack(side = 'left')
            self.menu_button.pack(side = 'left', pady = 10)
```

*open_about(self)*
Abre el formulario de información "About".

*open_build(self)*
Limpia el panel de contenido y carga el formulario "Building".

*open_T2B(self)*
Limpia el panel de contenido y carga el formulario "T2B".

*clear_panel(self, panel)*
Elimina todos los widgets de un panel especificado.

```
    def open_about(self):
        AboutFormDesign()

    def open_build(self):
        self.clear_panel(self.fondo)
        BuildingFormDesign(self.fondo)

    def open_T2B(self):
        self.clear_panel(self.fondo)
        T2BFormDesign(self.fondo)

    def clear_panel(self, panel):
        for widget in panel.winfo_children():
            widget.destroy()
```

**Dependencias**

*Util.util_ventana: Para centrar la ventana en la pantalla.*
*Util.config_color*s: Para los colores de la interfaz.
*Forms.about_form_design*: Para el formulario de información "About".
*Forms.B2T_form_design*: Para el formulario de traducción de braille a texto “B2T”.
*Forms.T2B_form_design*: Para el formulario de traducción de texto a braille "T2B".

Define los colores utilizados en la interfaz gráfica con Util.config_colors:

```
# Configuración de colores de los elementos de la GUI
COLOR_BARRA_SUP = "DARKBLUE"
COLOR_MENU_LAT = "LIGHTBLUE"
COLOR_FONDO = "LIGHTGRAY"
COLOR_CURSOR_MENU = "GRAY"
COLOR_BOTONES = "DARKBLUE"
```

## B2T_from_design.py

*__init__(self, main_panel)*
Inicializa el formulario de construcción creando los marcos (frames) superior e inferior, y configurando una etiqueta con el mensaje de "EN CONSTRUCCIÓN".

```
def __init__(self, main_panel):
        self.traslator = T2BCode()
        self.converter = ConvertTo()
        self.recorder = AudioRecorder()
        self.create_frames(main_panel)
        self.create_top_widgets()
        self.create_center_widgets()
          self.create_bottom_widgets()
```

*process_recorded_audio(self):*
Procesa el audio grabado, convierte a texto y actualiza el textbox de entrada.

```
def process_recorded_audio(self):
        transcribed_text = self.converter.get_transcribed_text()
        if transcribed_text:
            self.textBox_input.configure(state='normal')
            self.textBox_input.delete("1.0", 'end')
            self.textBox_input.insert("1.0", transcribed_text)
            self.textBox_input.configure(state='disabled')
            self.trad_2_braille(None)
        else:
            messagebox.showwarning("Transcription", "No se pudo transcribir
el audio.")
```

**Dependencias**

*tkinter:* Para la creación de la interfaz gráfica de usuario.
*Util.config_colors*: Para los colores de la interfaz, en este caso COLOR_FONDO para el fondo de la etiqueta.

## Vosk_recognition.py

*__init__(self, main_path):*
Constructor de la clase VoskRecognizer. Inicializa el modelo de Vosk para reconocimiento de voz.
model_path (str): Ruta del modelo Vosk

```
 def __init__(self, model_path):
        if not os.path.exists(model_path):
            raise FileNotFoundError("Model path does not exist.")
        self.model = Model(model_path) # Carga el modelo de Vosk desde la
ruta especificada en converTo
```

*get_raw_audio(Self):*
Obtiene el nombre del archivo de audio grabado

```
def get_raw_audio(self):
        return AudioRecorder().get_filename()
```

*transcribe_audio(self):*
Transcribe el audio grabado a texto usando el modelo Vosk.
str: Texto transcrito del archivo de audio

Obtiene el nombre del archivo de audio crudo, abre el archivo de audio en formato WAV. Verifica que el archivo de audio esté en formato mono PCM con una tasa de muestreo de 16000 Hz e inicializa el reconocedor Kaldi con el modelo Vosk y la tasa de muestreo del archivo de audio. Lee el archivo de audio en bloques y realiza el reconocimiento de voz, acumula el texto transcrito a medida que se procesa el archivo de audio y devuelve el texto transcrito.

```
def transcribe_audio(self):
        filename = self.get_raw_audio()
        wf = wave.open(filename, "rb")
        if wf.getnchannels() != 1 or wf.getsampwidth() != 2 or
wf.getframerate() != 16000:
            raise ValueError("Audio file must be WAV format mono PCM.")
        rec = KaldiRecognizer(self.model, wf.getframerate())
        self.transcription = ""
        while True:
            data = wf.readframes(4000)
            if len(data) == 0:
                break
            if rec.AcceptWaveform(data):
                result = rec.Result()
                self.transcription += json.loads(result)["text"] + " "
        result = rec.FinalResult()
        self.transcription += json.loads(result)["text"]
        return self.transcription.strip()
```

## Audio_recorder.py

*_init_(self):*
Constructor de la clase AudioRecorder. Inicializa los componentes necesarios para la grabación de audio.

```
def __init__(self):
        self.stop_event = threading.Event()
        self.temp_dir = tempfile.gettempdir()
        self.filename = os.path.join(self.temp_dir, "temp_audio.wav")
        self.frames = []
        self.p = pyaudio.PyAudio()
        self.stream = None
        atexit.register(self.cleanup)
```

*Record_audio(self):*
Inicia la grabación de audio.

```
 def record_audio(self):

        chunk = 1024
        sample_format = pyaudio.paInt16
        channels = 1
        fs = 16000


        self.stream = self.p.open(format=sample_format,
                        channels=channels,
                        rate=fs,
                        frames_per_buffer=chunk,
                        input=True,
                        stream_callback = self.callback)

        self.frames = []
        print("Recording...")
```

*Stop_recording(self):*
Detiene la grabación de audio y guarda el archivo.

```
def stop_recording(self):
        # self.stop_event.set()
        self.stream.stop_stream()
        self.stream.close()

        wf = wave.open(self.filename, 'wb')
        wf.setnchannels(1)
        wf.setsampwidth(self.p.get_sample_size(pyaudio.paInt16))
        wf.setframerate(16000)
        wf.writeframes(b''.join(self.frames))
        wf.close()
        print("Recording complete")
```

*Callback(self, in_Data, frame_count, time_info, status):*
Callback para manejar el flujo de datos de audio.

Args:
in_data (bytes): Datos de entrada.
frame_count (int): Conteo de frames.
time_info (dict): Información de tiempo.
status (int): Estado del flujo de audio.

```
def callback(self, in_data, frame_count, time_info, status):
        if status:
            print(status)
        self.frames.append(in_data)
        return in_data, pyaudio.paContinue
```

*Get_filename(self):*
Obtiene el nombre del archivo de audio grabado.

```
def get_filename(self):
        return self.filename
```

*Cleanuo(self):*
Limpia el archivo de audio temporal al finalizar.

```
def cleanup(self):
        if os.path.exists(self.filename):
            os.remove(self.filename)
```

## Ejemplo de ejecución

1. Iniciar el programa

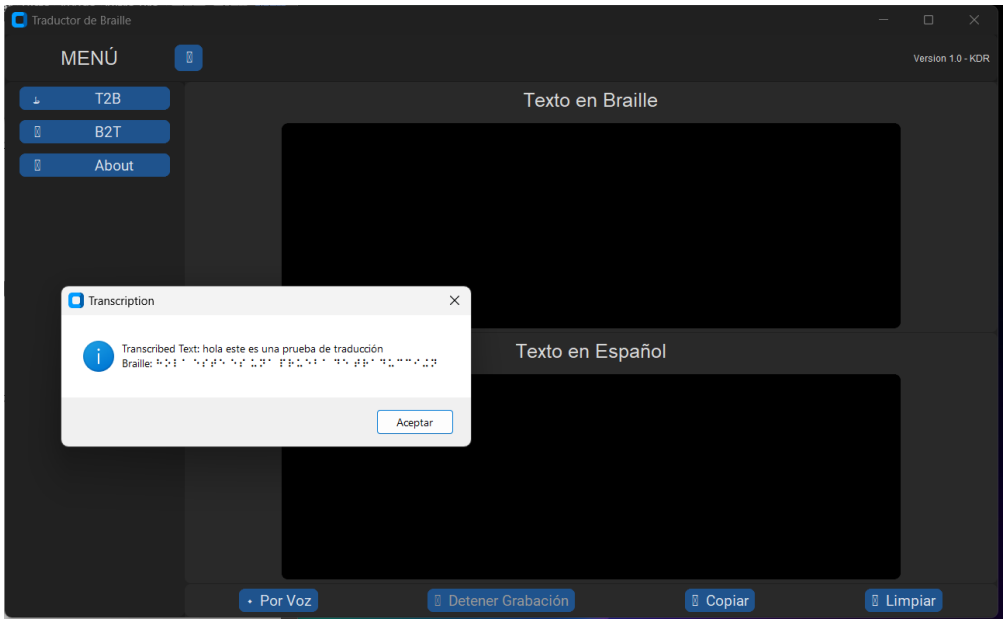
   BrailleApp_KDR.py

2. Se desplegará la siguiente ventana con la opción de traducción de texto español a braille.


3. Ingrese el texto a traducir en el campo correspondiente, la traducción a braille se generará de manera automática, la cual podrá observase en el recuadro inferior.


4. Una vez realizada la traducción, se podrá generar los documentos correspondientes en el formato deseado.

5. Pasando a la pestaña de traducción de braille a texto en español, donde se trabajará la traducción mediante audio.




6. Una vez finalizada la grabación, se evidenciará el texto en braille y su respectiva traducción en texto español.