

IMPLEMENTACION SISTEMA MANTIS

Plan diseñado para ser ejecutado como investigador independiente con acceso (por gestionar) a laboratorios colaboradores (nanomedicina, neurociencia, acústica) y recursos computacionales moderados.

OBJETIVO GENERAL

Implementar y validar experimentalmente la primera generación de MANTIS (v1.0) capaz de:

Decodificar actividad espontánea o evocada en cultivos neuronales mediante ultrasonido focalizado y nanotransductores moleculares, con correlación >0.8 frente a registro eléctrico de referencia.

1. MECANISMOS FÍSICOS

A. Phased Array Ultrasónico (Hardware)

- Plataforma: Kit de desarrollo Verasonics Vantage o alternativa open-source como OpenUS (basado en FPGA).
- Configuración :
 - 1024 elementos, frecuencia central: 5 MHz (resolución $\sim 300 \mu\text{m}$ en tejido).
 - Potencia ajustable: $0-1 \text{ W/cm}^2$ ISPTA (rango seguro: $TI < 0.7$).
- Modos operativos :
 - Tx : Pulsos modulados (10 ms ON / 90 ms OFF).
 - Rx : Recepción multicanal con muestreo $\geq 20 \text{ MS/s}$.

B. Corrección de Aberración Craneal (Simulación → Validación)

- Fase 1 (in silico) :
 - Usar k-Wave + modelo craneal Duke University Head Model .
 - Implementar algoritmo de time-reversal adaptativo .

- Métrica: reducir FWHM del foco de >1 mm a <300 μm .
- Fase 2 (ex vivo) :
 - Validar en cráneos de roedor/oveja con hidrogeles neuronales.
 - Medir distorsión de fase con transductores de referencia.

C. Detección de Armónicos

- Procesamiento en Rx :
 - FFT deslizante (ventana 50 ms, solapamiento 80%).
 - Extracción de potencia en bandas:
 - Fundamental: 4.5–5.5 MHz
 - 2f: 9–11 MHz
 - 3f: 13.5–16.5 MHz
- Métrica clave : SNR armónico 10 dB en presencia de MNTs activos.

2. MECANISMOS BIOLÓGICOS

A. Diseño y Síntesis de MNTs v1.0

Componente	Especificación	Proveedor/Método
Núcleo	Perfluorohexano (C_6F_{14}), 50 nm	NanoAssemblr (Precision Nanosystems)
Membrana	DOPC + colesterol (85:15) + 2 mol% DSPE-PEG2000	Autoensamblaje por microfluidica
Sensores	- Ci-VSP (plásmido codificante) - TRPV1 (proteína recombinante)	Transfección post-formación o reconstitución proteolipídica
Targeting	TAT (GRKKRRQRRR) conjugado a PEG	Síntesis peptídica sólida
Payload	Quasar 670-caged + HRP	Incubación pasiva

Control de calidad : DLS (tamaño), TEM (morfología), HPLC (pureza).

B. Protocolo de Activación y Seguridad

- Estimulación térmica segura :
 - Umbral: $40.5^{\circ}\text{C} \pm 0.3^{\circ}\text{C}$ (medido con termopar microscópico).
 - Límite de seguridad: si $T \geq 42^{\circ}\text{C} \rightarrow$ detener Tx automáticamente.
- Doble puerta lógica :
 - Liberación solo si: $(V_m < -40 \text{ mV}) \text{ Y } (T < 40.5^{\circ}\text{C})$.
 - Validado mediante patch-clamp + termografía infrarroja.

C. Biodegradabilidad y Toxicidad

- Ensayos in vitro :
 - Cultivos de microglía BV2 \rightarrow fagocitosis de MNTs (24 h).
 - Viabilidad celular (MTT assay): $>90\%$ tras exposición.
- Perfil de aclaramiento :
 - Incubación en suero humano \rightarrow HPLC-MS para seguimiento de fragmentos fluorados.
 - Tiempo medio de vida: objetivo <48 h.

3. MECANISMOS COMPUTACIONALES

A. Pipeline de Decodificación en Tiempo Real

Mermaid

graph LR

A[Raw RF Data] --> B[Beamforming MVDR]

B --> C[Sliding FFT]

C --> D[Harmonic Power Extraction]

D -- E[Feature Vector: [P_2f, P_3f, Entropy, Coherence]]

E -- F[Temporal Transformer]

F -- G[Neural State Estimate]

- Modelo : Temporal Transformer with Harmonic Attention (TTHA).
- Entrada : ventana de 50 ms, 4 armónicos × 1024 canales → vector de 4096 features.
- Salida : tasa de disparo estimada (spikes/s) o clase de intención.
- Latencia objetivo : <20 ms (implementado en TensorRT en NVIDIA Jetson Orin).

B. Entrenamiento del Modelo

- Datos sintéticos :
 - Simulaciones COMSOL + NEURON → 10^5 muestras de $((p(t), V_m(t)))$.
- Datos reales (fase inicial) :
 - Cultivos neuronales de hipocampo de rata (DIV 14–21).
 - Registro simultáneo:
 - Referencia : MEA (Multi-Electrode Array, 60 canales).
 - Señal MANTIS : ultrasonido + MNTs.
- Loss function :

$$L = \text{MSE}(\hat{r}, r_{\text{MEA}}) + \lambda \cdot \text{DTW}(\hat{r}, r_{\text{MEA}})$$

$$\mathcal{L} = \text{MSE}(\hat{r}, r_{\text{MEA}}) + \lambda \cdot \text{DTW}(\hat{r}, r_{\text{MEA}})$$

C. Calibración Personalizada

- Protocolo de 5 minutos :
 1. Estímulo visual/auditivo evocado.

2. Registro simultáneo MEA + MANTIS.
 3. Fine-tuning del decodificador con 100 trials.
- Resultado : modelo personalizado sin necesidad de cirugía.

FASE DE PRUEBAS INICIALES (Roadmap 0–18 meses)

Etapas	Objetivo	Métrica de Éxito	Recursos Necesarios
T0–3 meses	Síntesis y caracterización de MNTs v1.0	Tamaño 80–120 nm, PDI <0.2	Laboratorio de nanomedicina
T3–6 meses	Validación in vitro (cultivos + MEA)	Correlación $\backslash (r > 0.7 \backslash)$ entre MANTIS y MEA	Neurociencia básica, MEA
T6–12 meses	Integración hardware-software (ultrasonido + IA)	Latencia <25 ms, SNR >8 dB	FPGA/PC, Verasonics/OpenUS
T12–18 meses	Prueba en roedor anestesiado (corteza somatosensorial)	Decodificación de estímulos táctiles con accuracy >80%	Cirugía animal, ética aprobada

VALIDACIÓN CRÍTICA: ¿Qué demostraría que MANTIS funciona?

1. Causalidad : La señal armónica desaparece si:
 - Se bloquean los canales de voltaje (TTX).
 - Se eliminan los MNTs (lavado).
2. Especificidad : No hay señal en regiones sin MNTs.
3. Reproducibilidad : Mismo resultado en ≥ 3 preparaciones biológicas independientes.
4. Seguridad : Histología post-estudio muestra ausencia de necrosis o inflamación.

Recursos mínimos como investigador independiente

- Colaboraciones clave :
 - Laboratorio de nanomedicina (síntesis MNTs).
 - Laboratorio de neurociencia (cultivos, MEA, roedores).
 - Grupo de ultrasonido terapéutico (hardware).
- Costo estimado (v1.0) :
 - Materiales: ~\$15,000 USD
 - Acceso a equipos: colaboración o alquiler (~\$5,000)
- Alternativas low-cost :
 - Usar arrays ultrasónicos comerciales de menor canalización (64 elementos).
 - Simulaciones en lugar de ex vivo para aberración craneal.

De esta manera, podríamos iniciar pruebas en 3–6 meses y tendríamos un prototipo funcional en 12–18 meses . El enfoque modular permite validar cada pilar por separado antes de la integración final.

DOCUMENTOS ESENCIALES:

1. PROTOCOLO DETALLADO DE SÍNTESIS DE MNTs v1.0

(Molecular Neural Transducers — versión inicial para pruebas in vitro)

Objetivo : Obtener nanovesículas de 80–120 nm, con membrana funcionalizada con Ci-VSP y TRPV1, núcleo de perfluorohexano, y superficie PEG-TAT, capaces de liberar Quasar 670 en respuesta a voltaje + calor.

Materiales

Componente	Proveedor	Concentración final	Cantidad
Lípidos	Avanti Polar Lipids	—	—
– DOPC	Avanti 850375P	85 mol%	8.5 mg
– Cholesterol	Avanti 700000P	13 mol%	1.3 mg
– DSPE-PEG2000	Avanti 880240P	2 mol%	0.2 mg
Perfluorohexano (C ₆ F ₁₄)	Sigma-Aldrich 370912	—	50 µL
Quasar 670-caged	Lumiprobe 61080	100 µM en núcleo	10 µg
HRP (horseradish peroxidase)	Sigma P6782	1 mg/mL en núcleo	10 µg
Ci-VSP plásmido	Addgene 21280	50 µg/mL	5 µg
TRPV1 proteína recombinante	Abcam ab195703	0.5 mg/mL	5 µg
Péptido TAT (GRKKRRQRRR)	GenScript	1 mM stock	100 µL
Buffer PBS	pH 7.4, estéril	—	10 mL
Microfluidic chip	Precision Nanosystems NanoAssemblr	—	1 unidad

Protocolo paso a paso (duración total: ~4 horas)

Paso 1: Preparación de fases

1. Fase orgánica (lípidos + PFC) :

- Disolver DOPC, colesterol y DSPE-PEG200 en cloroformo (10 mg/mL total).

- Secar bajo nitrógeno → película lipídica.
- Resuspender en etanol absoluto (1 mL).
- Añadir perfluorohexano (50 µL) → mezclar por vórtex 1 min.

2. Fase acuosa (payload + sensores):

- En PBS (pH 7.4), mezclar:
 - Quasar 670-caged (10 µg)
 - HRP (10 µg)
 - Ci-VSP plásmido (5 µg)
 - TRPV1 (5 µg)
- Ajustar volumen a 1 mL .

Paso 2: Formación de nanovesículas (microfluidics)

1. Cargar fase orgánica en jeringa A, fase acuosa en jeringa B.
2. Usar NanoAssemblr con relación de flujo 3:1 (acuoso:orgánico) .
3. Flujo total: 12 mL/min durante 2 minutos .
4. Recoger el producto en tubo de 15 mL con PBS frío.

Paso 3: Funcionalización con TAT

1. Añadir péptido TAT (100 µL de 1 mM) a la suspensión.
2. Incubar 30 min a 4°C con agitación suave (300 rpm).
3. Purificar por ultracentrifugación :
 - 100,000 × g, 1 h, 4°C.
 - Resuspender en 1 mL PBS + 5% sacarosa (crioprotector).

Paso 4: Caracterización

Parámetro	Método	Especificación objetivo
Tamaño	DLS (Malvern Zetasizer)	80–120 nm, PDI < 0.2
Potencial zeta	DLS	-10 a -20 mV
Morfología	TEM negativo	Vesículas esféricas, núcleo visible
Proteínas incorporadas	Western blot	Banda positiva para Ci-VSP y TRPV1
Liberación	Termoplaca + patch-clamp	>80% liberación solo con $V_m > -40\text{mV}$ $V_m > -40\text{mV}$ + $T > 40.5^\circ\text{C}$ $T > 40.5^\circ\text{C}$

Paso 5: Almacenamiento

- Aliquotar en 100 μL .
- Congelar en nitrógeno líquido.
- Almacenar a -80°C .
- Estabilidad: 3 meses sin pérdida de función.

Nota crítica : La reconstitución de Ci-VSP y TRPV1 en la membrana se verifica mediante fluorescencia de unión a voltaje (DiBAC4) y respuesta a capsaicina (TRPV1 agonista).

2. SCRIPT PYTHON PARA PIPELINE DE DECODIFICACIÓN EN TIEMPO REAL

Este script implementa el pipeline completo desde datos RF hasta estimación de tasa de disparo , usando PyTorch y procesamiento de señal en tiempo real.

Requisitos : Python 3.9+, PyTorch 2.0+, NumPy, SciPy, librosa (opcional)

```
```python
```

```
 mantis_decoder.py
```

```
import numpy as np
```

```
import torch
```

```
import torch.nn as nn
```

```
from scipy.signal import butter, filtfilt, hilbert
```

```
from scipy.fft import rfft, rfftfreq
```

```
import time
```

```

```

## 1. PREPROCESAMIENTO DE SEÑAL

```

```

```
def bandpass_filter(data, low, high, fs, order=4):
```

```
 nyq = 0.5 * fs
```

```
 b, a = butter(order, [low/nyq, high/nyq], btype='band')
```

```
 return filtfilt(b, a, data, axis=-1)
```

```
def extract_harmonics(rf_signal, fs=20e6, f0=5e6, window_size=1024, hop=512):
```

```
 """
```

```
 rf_signal: (n_channels, n_samples)
```

```
 Devuelve potencia en fundamental, 2f, 3f
```

```
 """
```

```
 harmonics = []
```

```
 for i in range(0, rf_signal.shape[1] - window_size, hop):
```

```
window = rf_signal[:, i:i+window_size]
```

```
 FFT
```

```
fft_vals = np.abs(rfft(window, axis=-1))
```

```
freqs = rfftfreq(window_size, 1/fs)
```

```
 Potencia en bandas
```

```
p1 = np.mean(fft_vals[:, (freqs >= 4.5e6) & (freqs <= 5.5e6)], axis=-1)
```

```
p2 = np.mean(fft_vals[:, (freqs >= 9e6) & (freqs <= 11e6)], axis=-1)
```

```
p3 = np.mean(fft_vals[:, (freqs >= 13.5e6) & (freqs <= 16.5e6)], axis=-1)
```

```
 harmonics.append(np.stack([p1, p2, p3], axis=-1)) (n_channels, 3)
```

```
return np.array(harmonics) (n_frames, n_channels, 3)
```

```

```

## 2. MODELO DE DECODIFICACIÓN

```

```

```
class TemporalTransformerHarmonic(nn.Module):
```

```
 def __init__(self, n_channels=1024, n_harmonics=3, d_model=128, nhead=8,
num_layers=4):
```

```
 super().__init__()
```

```
 self.input_proj = nn.Linear(n_channels * n_harmonics, d_model)
```

```
 encoder_layer = nn.TransformerEncoderLayer(d_model, nhead,
dim_feedforward=512, batch_first=True)
```

```
 self.transformer = nn.TransformerEncoder(encoder_layer, num_layers)
```

```
 self.regressor = nn.Linear(d_model, 1) tasa de disparo
```

```

def forward(self, x):
 x: (batch, seq_len, n_channels, n_harmonics)

 B, L, C, H = x.shape

 x = x.view(B, L, -1) (B, L, C*H)

 x = self.input_proj(x) (B, L, d_model)

 x = self.transformer(x) (B, L, d_model)

 x = self.regressor(x[:, -1]) solo último timestep

 return x.squeeze(-1)

```

### -----

### 3. PIPELINE EN TIEMPO REAL

### -----

```

class MANTISDecoder:

 def __init__(self, model_path="mantis_ttha.pth", fs=20e6):
 self.fs = fs

 self.model = TemporalTransformerHarmonic()

 self.model.load_state_dict(torch.load(model_path, map_location='cpu'))

 self.model.eval()

 self.buffer = [] almacena últimos N ventanas

 def process_rf_frame(self, rf_data):
 """
 rf_data: (1024, 1024) - datos RF crudos de un frame

 Retorna: tasa de disparo estimada (spikes/s)

```

```
"""
```

#### 1. Extraer armónicos

```
harmonics = extract_harmonics(rf_data[None, :], fs=self.fs) (1, 1024, 3)
```

#### 2. Acumular buffer (ventanas secuenciales)

```
self.buffer.append(harmonics[0])
```

```
if len(self.buffer) 10: ventana temporal de 500 ms
```

```
self.buffer.pop(0)
```

```
if len(self.buffer) < 10:
```

```
return 0.0 esperar suficientes datos
```

#### 3. Inferencia

```
with torch.no_grad():
```

```
input_tensor = torch.tensor(np.array(self.buffer)[None, :]).float() (1, 10, 1024, 3)
```

```
firing_rate = self.model(input_tensor).item()
```

```
return max(0.0, firing_rate) no negativo
```

```

```

#### 4. EJEMPLO DE USO

```

```

```
if __name__ == "__main__":
```

```
decoder = MANTISDecoder()
```

Simular datos RF entrantes (ej. desde Verasonics API)

```
while True:
```

```
 rf_frame = np.random.randn(1024, 1024) * 1e-3 placeholder
```

```
 start = time.time()
```

```
 rate = decoder.process_rf_frame(rf_frame)
```

```
 latency = (time.time() - start) * 1000
```

```
 print(f"Estimated firing rate: {rate:.2f} spikes/s | Latency: {latency:.1f} ms")
```

```
 if latency > 20:
```

```
 print("⚠ Latencia excede 20 ms")
```

```
 time.sleep(0.01) simular streaming
```

```
...
```

## Instrucciones de uso

### 1. Entrenamiento del modelo :

- Genera datos sintéticos con COMSOL + NEURON.
- Entrena `TemporalTransformerHarmonic` y guarda pesos como `mantis\_ttha.pth`.

### 2. Integración con hardware :

- Reemplaza `np.random.randn` por lectura real desde tu sistema de ultrasonido (ej. SDK de Verasonics o OpenUS).

### 3. Optimización :

- Convierte el modelo a TensorRT para Jetson Orin:

```
```bash  
  
trtexec --onnx=mantis.onnx --saveEngine=mantis.trt  
  
```
```

### Métricas de desempeño esperadas

| Componente          | Objetivo                   |
|---------------------|----------------------------|
| Latencia total      | <20 ms                     |
| Uso de CPU          | <50% en Jetson Orin        |
| Correlación con MEA | $r \geq 0.8$ (en cultivos) |

## DOCUMENTOS COMPLEMENTARIOS

### 1. PROTOCOLO DE VALIDACIÓN IN VITRO CON MEA

\*(Multi-Electrode Array — para correlacionar MANTIS con actividad eléctrica real)\*

#### Objetivo

Validar que la señal armónica generada por MNTs correlaciona significativamente ( $r \geq 0.8$ ) con la actividad espontánea o evocada registrada por MEA en cultivos neuronales.

## **Materiales**

- Cultivos primarios de hipocampo de rata (DIV 14–21)
- Sistema MEA (ej. Multi Channel Systems, 60 electrodos, 30  $\mu\text{m}$  pitch)
- Cámara de perfusión compatible con ultrasonido
- MNTs v1.0 (recién sintetizados)
- Estimulador eléctrico (para respuestas evocadas)
- Sistema de ultrasonido focalizado (5 MHz, phased array)
- Software: MC\_Rack (MEA), Python (MANTIS decoder)

## **Protocolo paso a paso**

### **Día 0 : Preparación del cultivo**

1. Sembrar neuronas hipocampales en placa MEA recubierta con poli-D-lisina.
2. Mantener en incubadora (37°C, 5% CO<sub>2</sub>) hasta DIV 14.

### **Día 14 : Registro basal**

1. Conectar placa MEA al amplificador.
2. Registrar actividad espontánea durante 30 min (37°C, buffer ACSF).
3. Identificar redes sincronizadas (ráfagas, LFP).

### **Día 15 : Aplicación de MNTs**

1. Perfundir MNTs (10<sup>8</sup> partículas/mL) en ACSF.
2. Incubar 2 h (permitir unión a membrana).
3. Lavar suavemente para eliminar MNTs no unidos.



## Día 15 : Adquisición simultánea

### 1. Iniciar registro MEA + ultrasonido en paralelo:

- MEA : muestreo a 25 kHz, filtro 200–3000 Hz.
- Ultrasonido : modo pulso-eco, 5 MHz, 1024 elementos, ventana de 50 ms cada 100 ms.

### 2. Realizar tres condiciones:

- Basal : 10 min sin estímulo.
- Evocado : estimulación eléctrica (1 mA, 100  $\mu$ s, 1 Hz) en electrodos periféricos.
- Bloqueo : añadir TTX (1  $\mu$ M)  $\rightarrow$  verificar supresión de ambas señales.

## Análisis

1. Extraer tasa de disparo por electrodo (spike sorting con Kilosort o SpyKING CIRCUS).
2. Promediar tasa en región de interés (ROI) bajo foco ultrasónico.
3. Correlacionar con señal armónica  $\backslash (H_2(t) \backslash$  de MANTIS:

$\backslash$

$$r = \frac{\text{cov}(\text{MEA\_rate}, H_2)}{\sigma_{\text{MEA}} \sigma_{H_2}}$$

$\backslash$

4. Éxito :  $\backslash (r \geq 0.8 \backslash)$  en  $\geq 3$  cultivos independientes.

**Control crítico : repetir sin MNTs  $\rightarrow$  señal armónica debe ser ruido de fondo.**

## 2. SCRIPT DE SIMULACIÓN COMSOL-PYTHON ACOPLADO

Este script genera datos sintéticos realistas para entrenar el decodificador, acoplando COMSOL (acústica + transporte) con NEURON (neuronal) .

Requisito : COMSOL Multiphysics 6.0+ con LiveLink for Python

```
```python
```

```
    mantis_simulation.py
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from neuron import h, gui    NEURON simulator
```

```
import mph    COMSOL Python API
```

```
-----
```

1. SIMULACIÓN NEURONAL (NEURON)

```
-----
```

```
def run_neuron_simulation(duration=1000):    ms
```

```
    """Genera Vm(t) y spikes en una neurona cortical"""
```

```
    soma = h.Section(name='soma')
```

```
    soma.L = soma.diam = 20
```

```
    soma.insert('hh')
```

```
    ic = h.IClamp(soma(0.5))
```

```
    ic.delay = 100
```

```
    ic.dur = 800
```

```
    ic.amp = 0.1    nA
```

```
    t_vec = h.Vector().record(h._ref_t)
```

```

v_vec = h.Vector().record(soma(0.5)._ref_v)
spike_times = []

def detect_spike():
    if len(v_vec) > 1 and v_vec[-2] < -20 and v_vec[-1] >= -20:
        spike_times.append(t_vec[-1])

h.finitialize(-65)
while h.t < duration:
    h.fadvance()
    detect_spike()

return np.array(t_vec), np.array(v_vec), np.array(spike_times)

```

2. SIMULACIÓN ACÚSTICA-MOLECULAR (COMSOL)

```

def run_comsol_simulation(Vm_t, t_vec, comsol_model_path="mantis_model.mph"):
    """Ejecuta COMSOL con Vm(t) como entrada"""
    client = mph.start()
    model = client.load(comsol_model_path)

    Interpolar Vm a nodos COMSOL
    model.parameter('Vm_func', 'interp1(t_table, Vm_table, t)')
    model.set('t_table', t_vec.tolist())

```

```
model.set('Vm_table', Vm_t.tolist())
```

Resolver

```
model.solve('Study 1')
```

Extraer $p(r,t)$ en receptor

```
p_rx = model.evaluate('p', 'Receiver Point')
```

```
t_comsol = model.evaluate('t')
```

```
client.remove(model)
```

```
client.stop()
```

```
return t_comsol, p_rx
```

3. GENERACIÓN DE DATOS DE ENTRENAMIENTO

```
def generate_training_sample():
```

1. Neuronal

```
t_neuron, Vm, spikes = run_neuron_simulation()
```

2. Acústico-molecular (simulado)

```
t_acoustic, p_rx = run_comsol_simulation(Vm, t_neuron)
```

3. Extraer armónicos (simulados)

```

from scipy.fft import rfft, rfftfreq

dt = t_acoustic[1] - t_acoustic[0]

fs = 1/dt

window = p_rx[:1024]  primer frame

fft_vals = np.abs(rfft(window))

freqs = rfftfreq(len(window), dt)

H2_power = np.mean(fft_vals[(freqs >= 9e6) & (freqs <= 11e6)])

```

4. Etiqueta: tasa de disparo en ventana

```
firing_rate = len(spikes[(spikes >= 0) & (spikes < 1000)]) / 1.0  spikes/s
```


```
return H2_power, firing_rate
```

Ejemplo: generar dataset

```

if __name__ == "__main__":
    X, y = [], []
    for i in range(1000):
        x, label = generate_training_sample()
        X.append(x)
        y.append(label)
    np.savez("mantis_synthetic_data.npz", X=np.array(X), y=np.array(y))
'''

```

 Nota : El modelo COMSOL debe incluir:

- Ecuación de Westervelt acoplada a $\beta(c_S)$

- Ecuación de reacción-difusión para $\nabla(c_S(V_m))$
- Geometría de tejido cerebral + transductor

3. GUÍA PARA INTEGRAR CON OPENUS

OpenUS es una plataforma open-source para ultrasonido terapéutico basada en FPGA. Aquí está la guía para conectarla al pipeline de MANTIS.

Requisitos

- Placa Red Pitaya STEMLab 125-14 o DE0-Nano-SoC
- Transductor ultrasónico 5 MHz, 128–1024 elementos
- OpenUS firmware v2.0+ (github.com/openus)

Pasos de integración

Paso 1: Configurar OpenUS

1. Clonar repositorio:

```
``bash
git clone https://github.com/openus/openus.git
cd openus/firmware
make program flashear FPGA
...

```

2. Modificar `config.yaml`:

```
``yaml

```

```
frequency: 5e6
prf: 10      pulsos por segundo
pulse_length: 10e-3  10 ms
channels: 1024
sample_rate: 20e6
...
```

Paso 2: Interfaz Python → OpenUS

Usa la API REST de OpenUS para controlar Tx/Rx:

```
```python
openus_interface.py

import requests
import numpy as np

class OpenUSController:
 def __init__(self, ip="192.168.1.100"):
 self.base_url = f"http://{ip}:8080"

 def transmit_pulse(self, focus_x=0, focus_y=0, focus_z=30e-3):
 """Enfocar y transmitir pulso"""
 payload = {
 "mode": "tx",
 "focus": [focus_x, focus_y, focus_z],
 "power": 0.5, 0–1
 "duration": 0.01 10 ms
 }
```

```
}
requests.post(f"{self.base_url}/pulse", json=payload)
```

```
def receive_rf(self, n_samples=1024):
 """Recibir datos RF crudos"""
 response = requests.get(f"{self.base_url}/rf?n={n_samples}")
 rf_data = np.array(response.json()) (1024, n_samples)
 return rf_data
```

Integración con MANTIS decoder

```
if __name__ == "__main__":
 us = OpenUSController()
 decoder = MANTISDecoder() del script anterior

 for _ in range(100):
 us.transmit_pulse(focus_z=25e-3) 25 mm profundidad
 time.sleep(0.05) esperar eco
 rf = us.receive_rf()
 rate = decoder.process_rf_frame(rf)
 print(f"Firing rate: {rate:.2f} Hz")
 ...
```

Paso 3: Calibración de fase (aberración)

1. Ejecutar sweep de calibración:

```
``python
us.calibrate_skull(ct_path="skull_model.npy") opcional
```



...

2. OpenUS aplica corrección de fase automáticamente en cada pulso.

---

Arquitectura final del sistema experimental

...

[OpenUS FPGA]  $\leftrightarrow$  [Python Control]  $\leftrightarrow$  [MANTIS Decoder]

↑

[Transductor 1024-element]

↓

[Cultivo neuronal + MNTs en placa MEA]

↓

[MEA System] → Validación de ground truth

...