RACHAEL INFERENCE - Guía de Usuario

28/08/2025

Preparación del entorno

1. Estructura de directorios

```
rachael-vision-inference/
├── Source/            # Scripts del sistema
├── Models/            # Tu modelo ONNX y configuración
├── Images/            # Imágenes para modo emulación (opcional)
├── Output/            # Imágenes NOK generadas (auto-creado)
├── docker-compose.yml     # Configuración Docker
└── Dockerfile         # Imagen Docker
```

2. Comprobación de Scripts

Comprueba que en la carpeta **Source/** se encuentran los siguientes archivos Python:

- RACHAEL.py
- basler_module.py
- inference_classification_pytorch_onnx.py
- graphs_module_v1.py
- visor.py

Configuración del Modelo

Copia tu modelo entrenado y el archivo de configuración json que lo acompaña (si lo has entrenado en el docker de entrenamiento *rachael-vision-classifier*) en la carpeta **Models/**

El sistema busca automáticamente las clases del modelo en *conversion_config.json*

Si no tienes archivo *conversion_cofig.json* crea un archivo de texto "*labels.txt*" con el siguiente formato cambiando las palabras por las etiquetas que debería categorizar tu modelo:

```
arandela
ok
pobres
valvula
varios
```

La carpeta Models/ debería quedar:

```
Models/
├── model.onnx          # Tu modelo ONNX (OBLIGATORIO)
├── conversion_config.json  # Configuración de clases (RECOMENDADO)
└── labels.txt          # Etiquetas alternativas (OPCIONAL)
```

Configuración de imágenes para emulación

Para probar el sistema sin cámaras físicas, coloca tus imágenes en la carpeta **Images/**

Usa formatos .jpg, .jpeg o .png

```
Images/
├── ok_muestra_001.jpg
├── ok_muestra_002.jpg
├── valvula_defecto_001.jpg
├── arandela_mal_002.jpg
└── varios_problema_003.jpg
```

Construcción y ejecución del Docker

1. Verificar Docker y NVIDIA Runtime

```bash
# Verificar Docker
docker --version

# Verificar runtime NVIDIA (para GPU)
docker run --rm --gpus all nvidia/cuda:11.0-base-ubuntu20.04 nvidia-smi
```

2. Configurar variables de entorno

Para modo emulación:

```bash
export PYLON_CAMEMU=1
export DISPLAY=:0
```

Para modo cámara física:

```bash
export PYLON_CAMEMU=0
export DISPLAY=:0
```

Si previamente has guardado la configuración de cámara basler en un archivo .pfs, modifica la ruta en el script **basler_module** en la linea 34:

```
DEFAULT_CONFIG_FILE = '/workspace/source/xx.pfs'
```

3. Construir la Imagen Docker y ejecutarla

```bash
cd rachael-vision-inference/
DOCKER_DEFAULT_RUNTIME=nvidia docker compose up --build
```

Este proceso:

- Descarga la imagen base NVIDIA L4T ML
- Instala dependencias Python
- Configura pypylon para cámaras Basler
- Prepara el entorno TensorRT

Personalización para tus etiquetas

Si tu modelo tiene labels diferentes, edita **basler_module.py**

```python
# Línea ~45 aproximadamente
def load_labels_from_config(model_path):
    # ...código existente...

    # Cambiar esta línea con TUS clases:
    return ['tu_clase_1', 'tu_clase_2', 'ok', 'defecto', 'otra_clase']
```

Y edita **graphs_module_v1.py** para cambiar los colores de tus labels en los gráficos

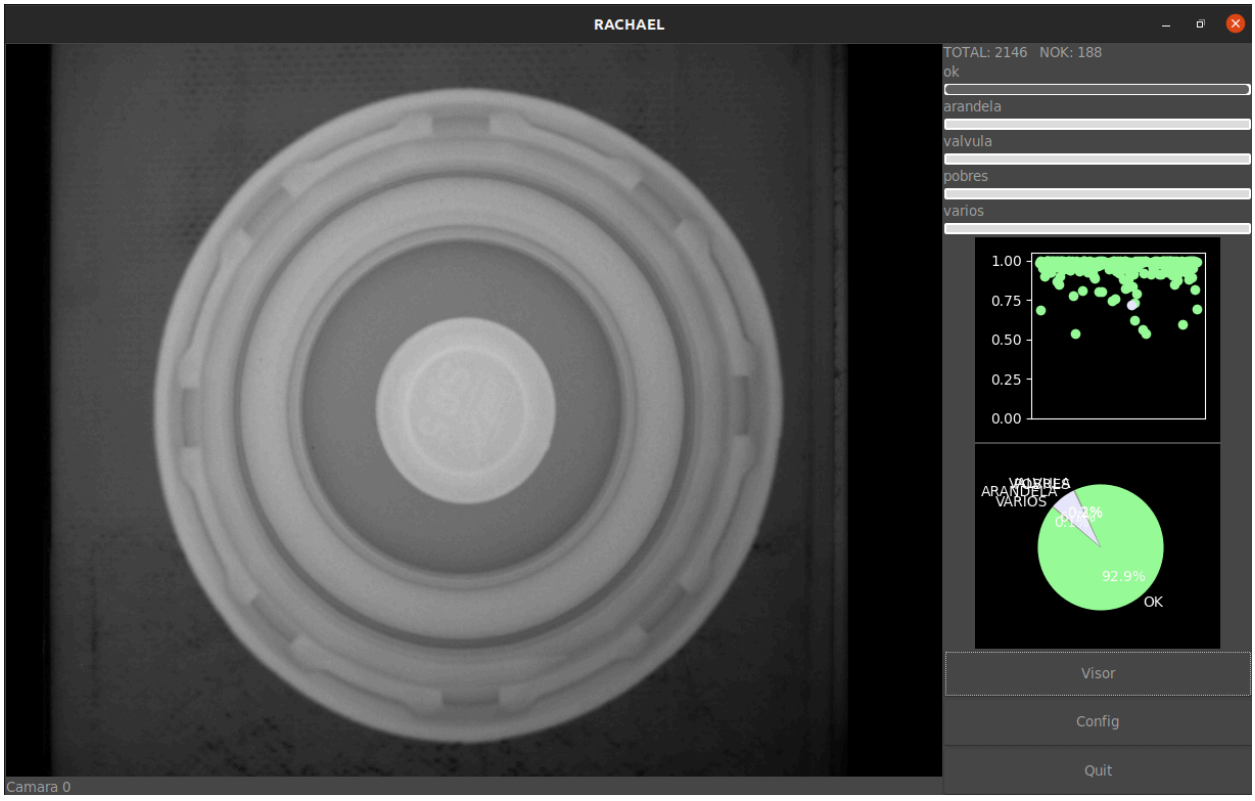
```
# Línea ~30 aproximadamente
def _generate_colors(self, labels):
    color_map = {
        'ok': '#00ff00',        # Verde para OK
        'tu_clase_1': '#ff9900',  # Naranja para tu primera clase
        'tu_clase_2': '#ff0000',  # Rojo para defectos graves
        'defecto': '#ff6600',     # Rojo-naranja
        'otra_clase': '#9900cc'   # Morado
    }
```

5

Ajusta las barras de progreso editando **RACHAEL.py**

```python
# Línea ~25 aproximadamente
categories_num = 5  # Cambiar por el número de TUS clases
```

Interfaz:



Panel Principal:

- Imagen central: Stream de cámara o emulación
- Contadores superiores: Total procesado, total NOK

- Barras de progreso: Confianza por cada clase
- Gráficas: Muestran el porcentaje total de cada clase

Controles Disponibles:

- Visor: Ver imágenes clasificadas como NOK
- Config: Ajustar tolerancia y guardado
- Quit: Salir limpiamente

Resolución de problemas comunes:

Problema: "No se encontraron clases":

1. Verifica que conversion_config.json esté en Models/
2. Asegúrate de que el formato JSON sea válido
3. Como alternativa, crea labels.txt

Problema: "Memoria GPU insuficiente":

Modifica docker-compose.yml y vuelve a contruir y ejecutar la imagen

```yaml
environment:
 - TRT_WORKSPACE_MB=128  # Reducir de 256 a 128
```

Problema: "No se pueden cargar imágenes de emulación":

1. Verifica que las imágenes estén en Images/
2. Comprueba que sean formatos soportados (jpg, png, bmp)
3. Asegúrate del montaje de volumen en docker-compose.yml

Problema: GUI no aparece:

```bash
# En Linux, permitir conexiones X11
xhost +local:docker

# Verificar variable DISPLAY
echo $DISPLAY
```

Archivos de Salida

Imágenes NOK Guardadas:

Ubicación: Output/NOK_CAM0_YYYY-MM-DD_HH-MM-SS.jpg

Estadísticas CSV

Ubicación: csv_outputs/stats_YYYY-MM-DD.csv

Contenido: timestamp, probabilidades por clase

Comandos útiles

(Abrir otra terminal)

Ver logs en tiempo real:

```bash
docker-compose logs -f
```

Parar el sistema:

```bash
docker-compose down
```

Reconstruir tras cambios:

```bash
docker-compose build --no-cache
docker-compose up
```