

Sensor de nivel – Notas de Aplicación

Descripción general

Esta nota de aplicación describe como desarrollar un sensor de nivel de alta precisión utilizando el modelo uRAD Level Sensing. Este sensor de nivel puede servir para medir el llenado en tanques de materiales sólidos o líquidos, nivel de ríos o pantanos para aplicaciones de desbordamientos o cualquier aplicación donde sea necesario medir con precisión la distancia en la dirección frontal.

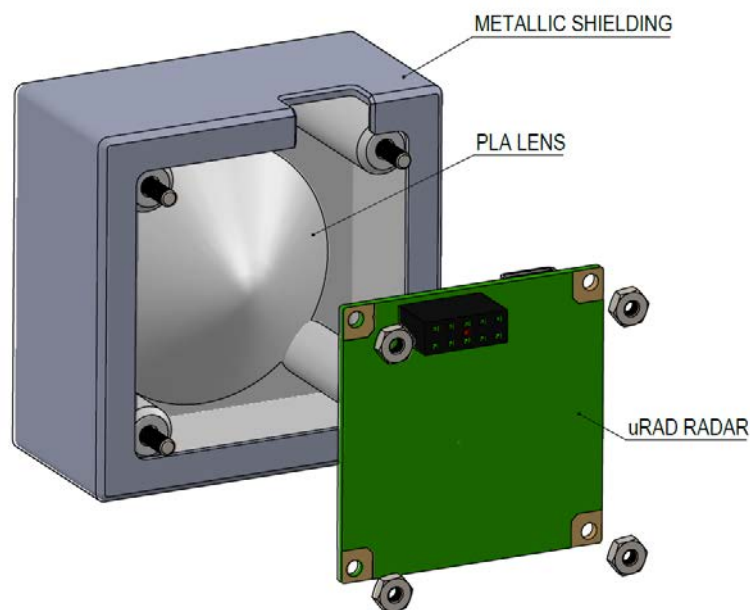
Dispositivos

Esta aplicación se puede desarrollar con nuestra solución uRAD Level Sensing basada en uRAD Industrial y uRAD Automotive. Se han elegido estos modelos de radar, principalmente por:

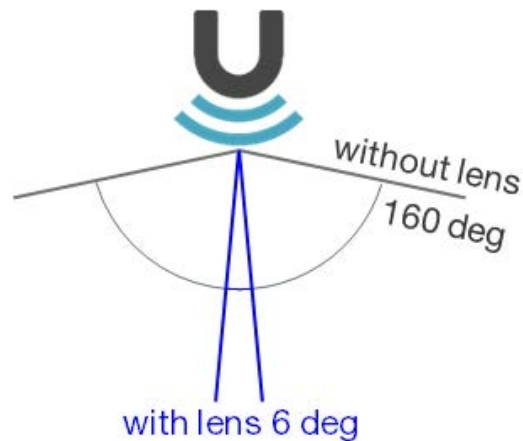
- Funcionan en la banda de 60-64 o 77-81 GHz, bandas de emisión para aplicaciones de sensado de nivel.
- Permite alcanzar precisiones de ± 1 mm.
- Resulta en un equipo muy compacto.
- Se puede conseguir un ángulo de visión muy estrecho, inferior a 6 grados añadiéndole una lente.

Hardware

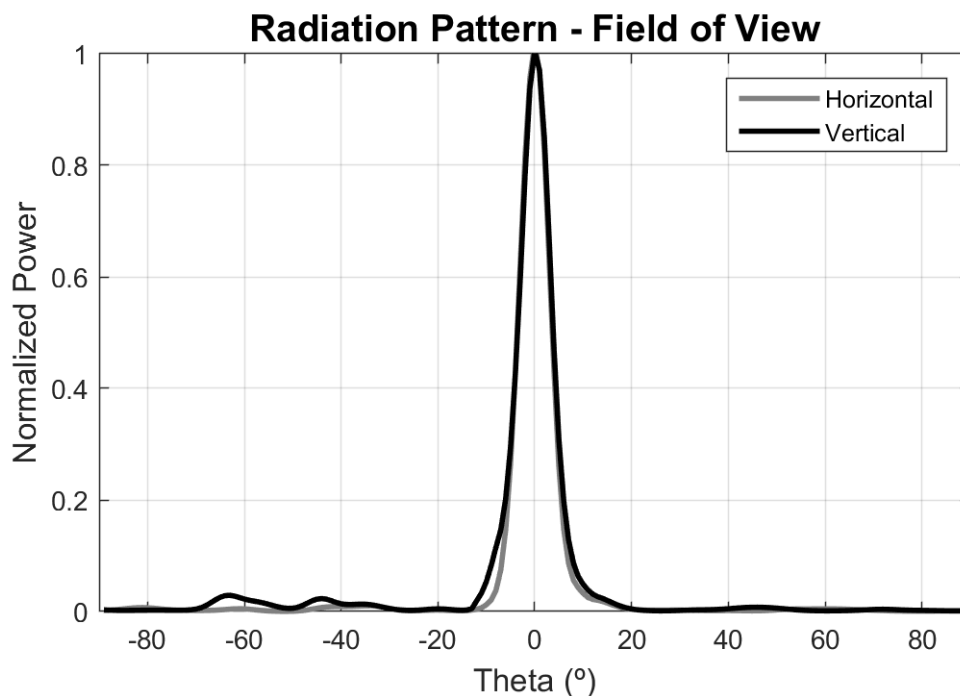
El montaje básico es muy simple y consiste en un radar uRAD y una caja de PLA con lente integrada y apantallamiento metálico en sus laterales.



Con la lente se consigue reducir el ángulo de visión de uRAD Industrial de 160 grados a tan solo 6 grados. De esta manera, se enfoca toda la potencia emitida en la dirección recta. Por ejemplo, para aplicaciones de llenado de tanques, esto es un aspecto crucial ya que el radar no debe ver las paredes del silo y que rebotes indeseados no le permitan medir con precisión.

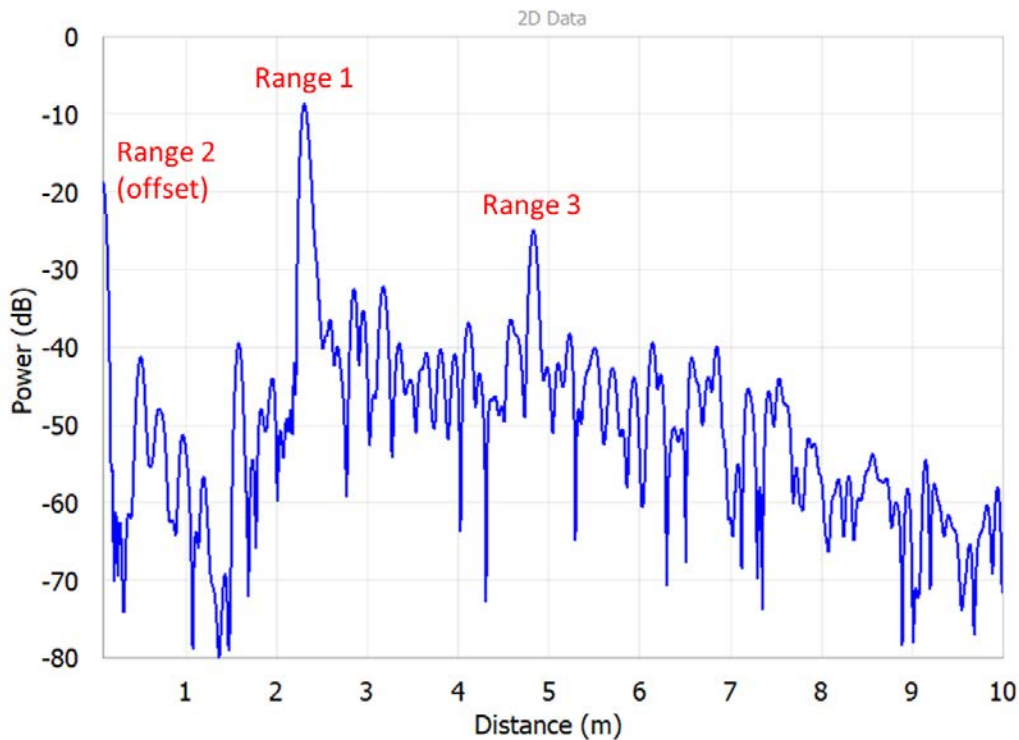


A continuación, se muestra el diagrama de radiación medido.



Aspectos teóricos

El radar calcula las distancias a los objetivos mediante la transformada rápida de Fourier (FFT) de la señal recibida. Observando el espectro de la FFT, se pueden identificar los picos con más nivel, como las reflexiones de los objetos o superficies. El software de sensado de nivel permite visualizar este espectro y devuelve el rango de los tres picos con mayor nivel. A continuación, se muestra un ejemplo:



En el ejemplo, el radar se fija sobre una mesa apuntando al techo que se encuentra a una distancia de 2.5 metros. En la gráfica se identifican 3 picos.

- El más significativo, range 1, a 2.5 metros y que tiene una mayor amplitud es el techo, que proporciona la mayor reflexión.
- Hay un segundo pico, range 2, prácticamente en 0, que no corresponde a ningún valor real. A una distancia muy corta, generalmente inferior a 10 cm, siempre está presente un pico que no es real, llamado offset, que es inherente a la tecnología radar y que hay que descartar.
- Un tercer pico, range 3, a 5 metros corresponde con la doble reflexión producida por la mesa y el techo, que el radar es también capaz de detectar.

El software de sensado devuelve la distancia de los tres picos con más amplitud, pero **la configuración permite seleccionar el rango de distancias donde buscar los tres picos**. Por lo tanto, es posible eliminar fácilmente el offset de los resultados o descartar todos aquellos picos fuera del rango de interés.

Funcionamiento

Lo primero, se sube el firmware a la placa uRAD. Hemos desarrollado un firmware específico para esta aplicación que se distribuye de manera gratuita con la compra cuando se solicita. El proceso de subir un firmware nuevo a uRAD está explicado en el manual de usuario general.

El funcionamiento del sensor es bastante simple. El dispositivo maestro, que controla uRAD, manda los comandos de configuración al radar por USB o UART. Una vez uRAD recibe los comandos, empieza a enviar los valores de medida.

La mayoría de los comandos son fijos por defecto. Solo hay que configurar unos pocos. Hemos hecho este proceso transparente para el usuario. Por lo tanto, se debe configurar:

```
#### CONFIGURATION PARAMETERS ####
model = 'IWR'           # 'IWR' or 'AWR'
maximum_distance = 12  # maximum distance to measure. From 12 to 150 meters
range_min = 0          # lower limit of the range of interest
range_max = 50         # upper limit of the range of interest
sampling_rate = 2      # min = 2, max = 20 samples per second
offset = 0              # apply an offset to all range measured values
```

- **model:** 'IWR' si su modelo de radar es uRAD Industrial y funciona a 60 GHz, o 'AWR' si es uRAD Automotive funcionando a 77 GHz.
- **maximum_distance:** el radar puede configurarse para medir una distancia máxima de 12 a 150 metros. **Es mejor elegir el valor más bajo posible para tener la mejor resolución.**
- **range_min & range_max:** rango de distancia de interés. Limita los resultados a este rango de distancias.
- **sampling_rate:** medidas por segundo que el radar envía al dispositivo maestro. De 2 a 20 muestras por segundo.
- **offset:** sumar una compensación (en metros) a los valores medidos.

Ejemplos de programación

Junto con el hardware y firmware, también se proporcionan diversos scripts de Python y C++ para controlar el radar, o bien por su conector USB o por el conector de pines a través de UART.

- **level_sensing_UART.py**

Este programa es útil cuando el sensor se controla por el conector de pines usando una sola línea UART. Solo se envían los resultados de distancia (**el gráfico de FFT y los valores de amplitud no se reciben**).

Se deben fijar algunos parámetros de configuración extra:

```
#### OUTPUT RESULTS ####
saveResults = True           # Save results in .txt file
printResults = True          # Print results in the console
numberOfPeaksToSave = 3     # Number of peaks to save (max = 3)
fileSizeMinutes = 2         # Minutes to split the .txt file

#### UART PORT ####
port_name = '/dev/serial0'

#### RESET ####
reset_pin_number = 6        # GPIO pin number of the Master Device
reset = True
```

- **saveResults**: los resultados se guardan dentro de la carpeta *output_files* en el archivo **YYYY_MM_DD_HH_mm_results.txt**. Cada archivo de resultados tiene una cabecera con la fecha y hora.
- **printResults**: imprime los resultados en la consola de Python.
- **numberOfPeaksToSave**: elige el número de picos (valores de distancia más significativos) a guardar. Máximo 3.
- **fileSizeMinutes**: divide el archive de resultados por minutos, para limitar el tamaño de cada archive.

También, se debe configurar el nombre del Puerto UART y si se va a usar el pin de reinicio (y su número de pin).

- **level_sensing_USB.py**

Similar al anterior pero controlado por USB. No incluye el pin de reinicio. **Por lo tanto, cada vez que quieras lanzar el código otra vez, se debe resetear manualmente el radar con el botón físico de reinicio.**

```
#### OUTPUT RESULTS ####
saveResults = True           # Save results in .txt file
```

```

printResults = True           # Print results in the console
numberOfPeaksToSave = 3      # Number of peaks to save (max = 3)
fileSizeMinutes = 2         # Minutes to split the .txt file

#### USB PORT ####
configPort_name = 'COM1'
dataPort_name = 'COM2'

```

- **level_sensing_UART_GUI.py:**

El más completo porque **permite obtener resultados de distancia y también la amplitud de los picos**. Los comandos de configuración incluyen mandar los valores de IQ para el gráfico FFT. Por lo tanto, el gráfico de picos (FFT) se puede visualizar si se selecciona.

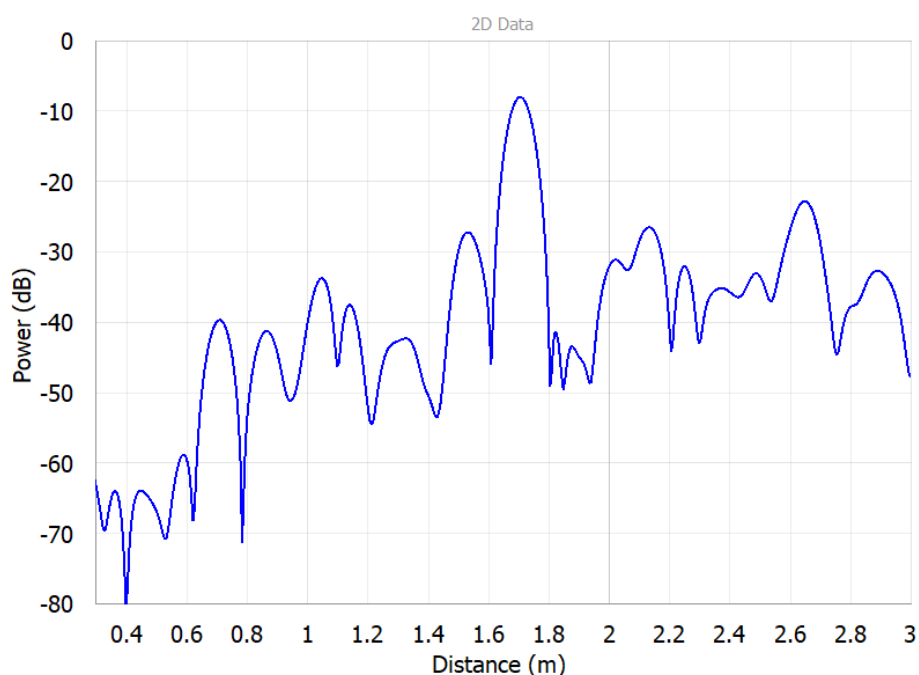
```

#### OUTPUT RESULTS ####
calculateAmplitude = True    # Calculate amplitude of the peaks besides range
saveResults = True          # Save results in .txt file
printResults = True         # Print results in the console
plotSpectrum = True         # Show Amplitude Vs Distance plot
numberOfPeaksToSave = 3     # Number of peaks to save (max = 3)
fileSizeMinutes = 2        # Minutes to split the .txt file

#### UART PORT ####
port_name = '/dev/serial0'

#### RESET ####
reset_pin_number = 6        # GPIO pin number of the Master Device
reset = True

```



- **level_sensing_USB_GUI.py:**

Similar al anterior, pero por USB. **Recuerda que cada vez que quieras lanzar el código otra vez, se debe resetear manualmente el radar con el botón físico de reset.**

```
#### OUTPUT RESULTS ####
calculateAmplitude = True # Calculate amplitude of the peaks besides range
saveResults = True # Save results in .txt file
printResults = True # Print results in the console
plotSpectrum = True # Show Amplitude Vs Distance plot
numberOfPeaksToSave = 3 # Number of peaks to save (max = 3)
fileSizeMinutes = 2 # Minutes to split the .txt file

#### USB PORT ####
configPort_name = 'COM1'
dataPort_name = 'COM2'
```

- **level_sensing_function.py**

Este ejemplo es un script que muestra cómo crear una función simple para tomar una sola medida (promediada al número de muestras que se quiera) cada vez que se llama a la función. Por lo tanto, permite al usuario tomar solo una medida en el momento deseado.

- **level_sensing_UART.cpp**

Este programa de C++ es equivalente al archiva de Python *level_sensing_UART.py*. La diferencia en este programa es que, en lugar de definir los parámetros de configuración, debes introducirlos directamente (su número equivalente) en los comandos de configuración.

Los comandos de configuración, en el Código, son las siguientes líneas:

```
"flushCfg\n", \
"dfedataOutputMode 1\n", \
"channelCfg 1 1 0\n", \
"adcCfg 2 1\n", \
"adcbufCfg 0 1 1 1\n", \
"profileCfg 0 60 7 7 114.4 0 0 31.23 1 512 5000 0 0 48\n", \
"chirpCfg 0 0 0 0 0 0 0 1\n", \
"frameCfg 0 0 10 0 500 1 0\n", \
"lowPower 0 0\n", \
"guiMonitor 1 0 0 0 0 1\n", \
"RangeLimitCfg 2 1 0.1 12.0\n", \
"sensorStart\n"
```

Los 5 número relevantes están destacados en amarillo y corresponden a:

- frecuencia de inicio de acuerdo a **model**, 60 para IWR o 77 para AWR.
- chirp slope de acuerdo a **maximum_distance**, desde 1.87 (150 metros) a 31.23 (12 metros). Ambos valores se relacionan con la formula:

$$\text{chirp_slope [MHz}/\mu\text{s}] = \frac{374.7405725}{\text{maximum_distance [m]}}$$

- tiempo entre medidas de acuerdo a **sampling_rate**, desde 50 (20 muestras/s) a 500 (2 muestras/s). Ambos valores se relacionan con la formula:

$$\text{measurement_periodicity [ms]} = \frac{1000}{\text{sampling_rate}}$$

- **range_min** y **range_max**.

Además, un código similar llamado **uRAD_LevelSensing.ino** se proporciona para la plataforma Arduino.

Ejemplo con medidas reales

A continuación, se presenta un caso práctico real.

La configuración es la siguiente. El radar se coloca a una distancia de 1.7 metros de una chapa metálica. A mitad de distancia, se colocará un cristal. Se van a medir tres situaciones: sin cristal, con el cristal perpendicular al radar y con el cristal inclinado.



El radar está conectado a un ordenador. Se alimenta directamente por el USB por donde también se configura y recibe los datos.

El script que se utiliza es *level_sensing_USB_GUI.py* ya que queremos mostrar la gráfica de pico además de guardar los resultados.

La configuración es:

```
#### CONFIGURATION PARAMETERS ####
model = 'IWR'           # 'IWR' or 'AWR'
maximum_distance = 12  # maximum distance to measure. From
range_min = 0.3        # lower limit for the range of interest
range_max = 3          # upper limit for the range of interest
sampling_rate = 2      # min = 2, max = 20 samples per second
offset = 0              # apply an offset to all range measured values
```

Conectamos el radar y vemos los puertos COM que se han asignado. Configuramos el resto de los parámetros del script como sigue:

```
#### OUTPUT RESULTS ####
calculateAmplitude = False # Calculate amplitude of the peaks besides range
saveResults = True        # Save results in .txt file
printResults = True       # Print results in the console
plotSpectrum = True       # Show Amplitude Vs Distance plot
```

```

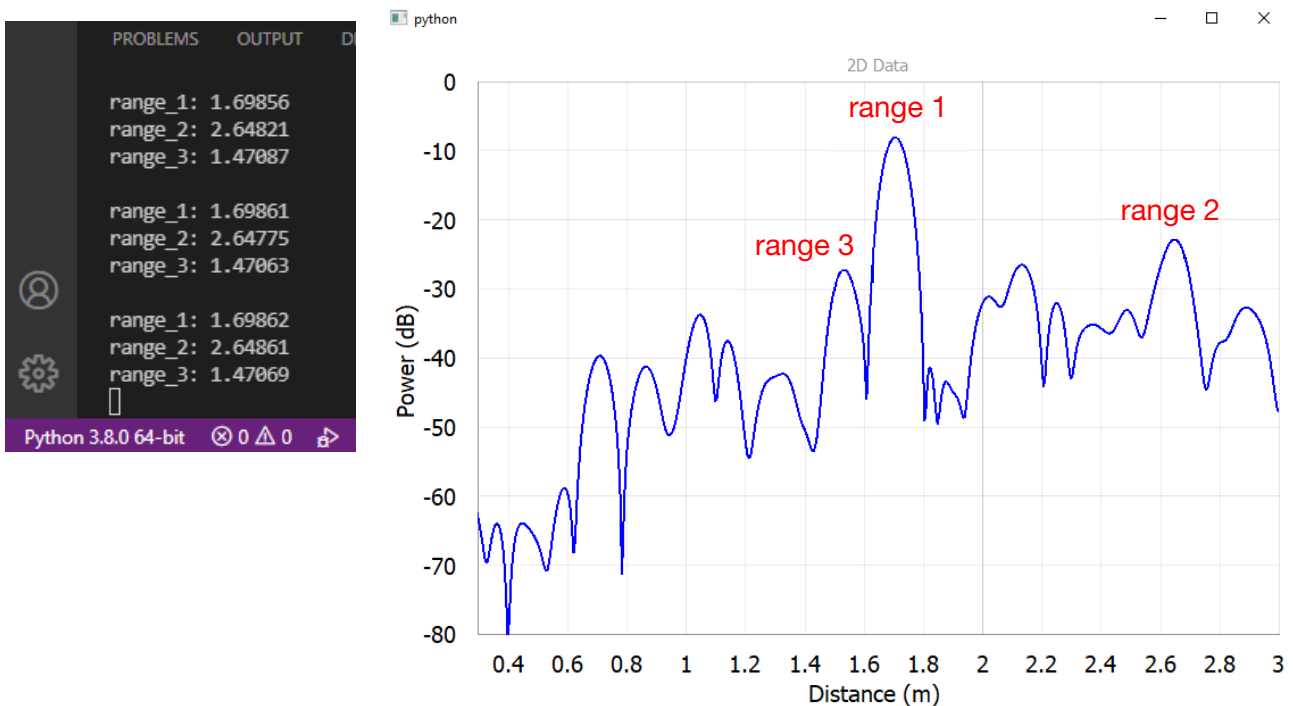
numberOfPeaksToSave = 3      # Number of peaks to save (max = 3)
fileSizeMinutes = 2         # Minutes to split the .txt file

#### USB PORT ####
configPort_name = 'COM1'
dataPort_name = 'COM2'

```

- **Sin cristal**

Tras lanzar el script, obtenemos los siguientes resultados medidos:

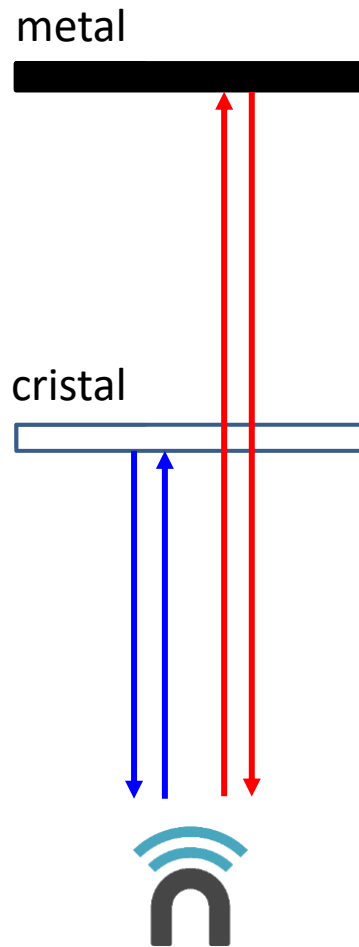


Observando los resultados tanto en la consola de Python como en la gráfica, vemos los tres picos que se identifican. Los resultados se ordenan de mayor a menor amplitud del pico.

A la distancia de 1.69856 metros se ha medido la chapa metálica con una amplitud muy por encima de los otros dos picos, que en realidad no se corresponden con ninguna superficie de interés.

El resto de picos del espectro, incluido range 2 y 3, se producen por el propio procesado de la FFT y de rebotes del entorno.

- Cristal colocado perpendicularmente



Los resultados obtenidos son:

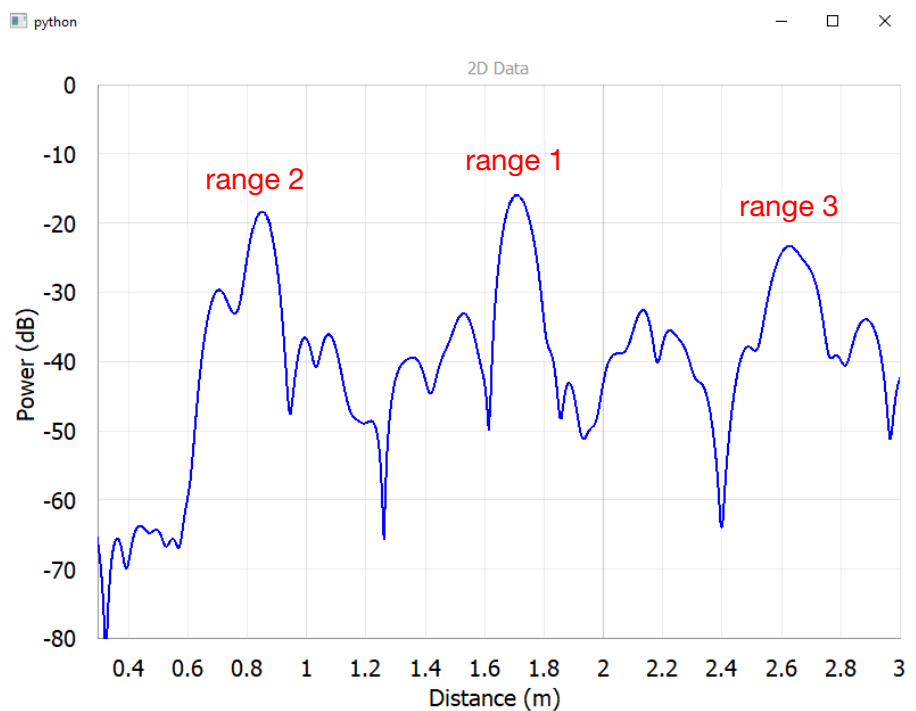
```

PROBLEMS  OUTPUT

range_1: 1.69573
range_2: 0.84838
range_3: 2.55751

range_1: 1.69562
range_2: 0.84762
range_3: 2.56129

range_1: 1.69587
range_2: 0.84762
range_3: 2.56228
    
```

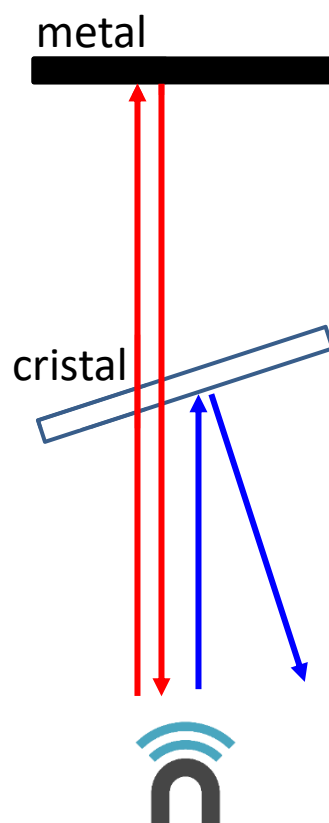


En este caso se observa como aparece el cristal a una distancia de 0.84838 metros y el metal sigue apareciendo a una distancia de 1.69573 metros. En este caso, el nivel de amplitud de ambos picos es muy similar, por lo que el valor de range 1 podría estar saltando entre ambos valores o incluso atribuir el pico de mayor amplitud al cristal. Además, se observa el pico correspondiente al metal ha bajado de amplitud como consecuencia de la atenuación de la onda a través del cristal.

- **Cristal colocado con ángulo**

En este caso, inclinamos el cristal para observar cómo su pico desaparece. Con una inclinación de 20-30 grados es suficiente para que el pico desaparezca.

Esto puede ser muy útil, por ejemplo, en situaciones reales donde se requiera aislar con una tapa el radar o donde es inevitable atravesar varios materiales. En ese caso, una inclinación en esa superficie mitigará el efecto del rebote.



Al inclinar el cristal, la mayor parte de la potencia reflejada no incide sobre el radar, como se muestra en la imagen anterior.

```

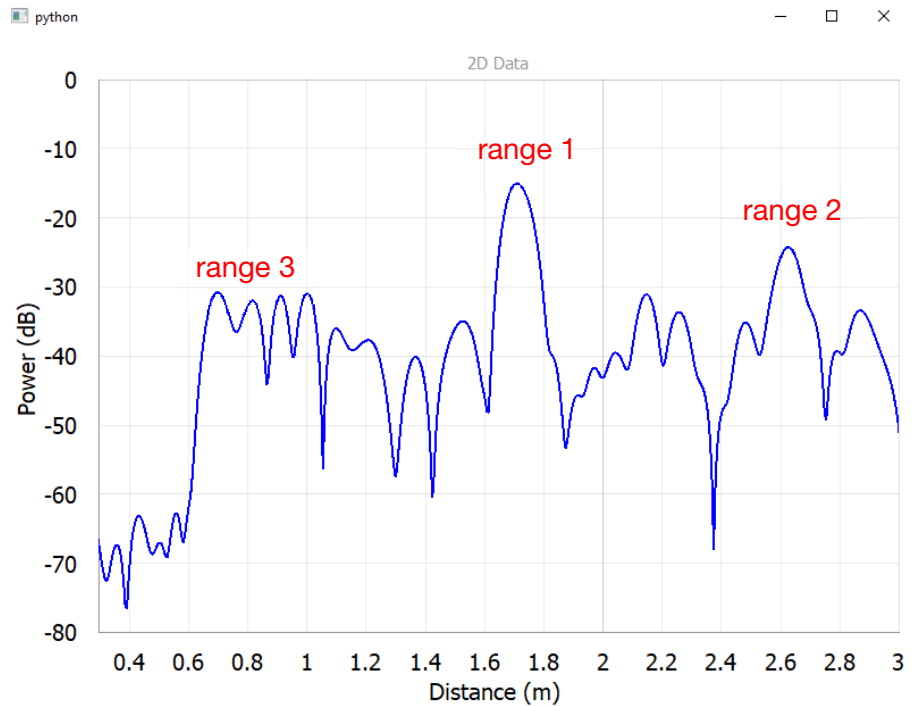
PROBLEMS OUTPUT
range_1: 1.69778
range_2: 2.63228
range_3: 1.00076

range_1: 1.69745
range_2: 2.63226
range_3: 1.00006

range_1: 1.69750
range_2: 2.62830
range_3: 0.69223

Python 3.8.0 64-bit 0 0

```



Se observa como el pico del cristal se ha mitigado completamente. En esta nueva situación, range 1 se identifica inequívocamente con el metal, siendo su amplitud muy superior al resto de picos.

El archivo de texto generado con los resultados se guarda en la carpeta *output files*. El formato es el siguiente:

```

1.69778 2.63228 1.00076 1622787713.261
1.69745 2.63226 1.00006 1622787713.762
1.69750 2.62830 0.69223 1622787714.261
1.69761 2.63123 0.69210 1622787714.762
1.69776 2.63211 1.00037 1622787715.260

```

Se guarda en cada fila los valores de cada medida, un frame de cada media, ordenando por columnas:

```
range1 range2 range3 time_stamp
```