



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería  
Informática**

**Creación de demos técnica y  
documentación asociada, para  
la detección/clasificación de  
materiales, utilizando un  
sensor radar a 60GHz  
Documentación Técnica**



Presentado por Martín Encabo Contreras  
en Universidad de Burgos — 16 de enero  
de 2023

Tutores:

José Francisco Díez Pastor  
Pedro Latorre Carmona



---

# Índice general

---

|   |            |
|---|------------|
| <b>Índice general</b>                                   | <b>i</b>   |
| <b>Índice de figuras</b>                                | <b>iii</b> |
| <b>Índice de tablas</b>                                 | <b>iv</b>  |
| <b>Apéndice A Plan de Proyecto Software</b>             | <b>1</b>   |
| A.1. Introducción . . . . .                             | 1          |
| A.2. Planificación temporal . . . . .                   | 1          |
| A.3. Estudio de viabilidad . . . . .                    | 6          |
| <b>Apéndice B Especificación de Requisitos</b>          | <b>9</b>   |
| B.1. Introducción . . . . .                             | 9          |
| B.2. Objetivos generales . . . . .                      | 9          |
| B.3. Catalogo de requisitos . . . . .                   | 10         |
| B.4. Especificación de requisitos . . . . .             | 11         |
| <b>Apéndice C Especificación de diseño</b>              | <b>17</b>  |
| C.1. Introducción . . . . .                             | 17         |
| C.2. Diseño de datos . . . . .                          | 17         |
| C.3. Diseño procedimental . . . . .                     | 18         |
| C.4. Diseño arquitectónico . . . . .                    | 19         |
| <b>Apéndice D Documentación técnica de programación</b> | <b>23</b>  |
| D.1. Introducción . . . . .                             | 23         |
| D.2. Estructura de directorios . . . . .                | 23         |
| D.3. Manual del programador . . . . .                   | 24         |

|  |           |
|--|-----------|
| D.4. Desarrollo del código . . . . .                             | 24        |
| D.5. Compilación, instalación y ejecución del proyecto . . . . . | 28        |
| D.6. Pruebas del sistema . . . . .                               | 29        |
| <b>Apéndice E Documentación de instalación</b>                   | <b>31</b> |
| E.1. Introducción . . . . .                                      | 31        |
| E.2. Instalación y ejecución en <i>Windows</i> . . . . .         | 31        |
| E.3. Instalación y ejecución en <i>Linux</i> . . . . .           | 34        |
| <b>Apéndice F Documentación de usuario</b>                       | <b>37</b> |
| F.1. Introducción . . . . .                                      | 37        |
| F.2. Requisitos de usuarios . . . . .                            | 37        |
| F.3. Manual del usuario . . . . .                                | 38        |
| <b>Bibliografía</b>  | <b>41</b> |

---

# Índice de figuras

---

|  |    |
|--|----|
| B.1. Diagrama de casos de uso. . . . .             | 11 |
| C.1. Diagrama de secuencia. . . . .                | 19 |
| C.2. Patrón de diseño fachada. . . . .             | 21 |
| D.1. Entorno Jupyter. . . . .                      | 24 |
| D.2. Función <i>get_data</i> . . . . .             | 25 |
| D.3. Función <i>get_modulo_fase</i> . . . . .      | 26 |
| D.4. Función <i>get_media</i> . . . . .            | 26 |
| E.1. Instalación de <i>Python</i> . . . . .        | 32 |
| E.2. Instalar librerías. . . . .                   | 33 |
| E.3. Instalar librería <i>Acconeer</i> . . . . .   | 33 |
| E.4. <i>RadarWave</i> en <i>Windows</i> . . . . .  | 34 |
| E.5. <i>RadarWave</i> en <i>Ubuntu</i> . . . . .   | 36 |
| F.1. Pantalla principal <i>RadarWave</i> . . . . . | 39 |

---

# Índice de tablas

---

|  |    |
|--|----|
| A.1. Costes de personal . . . . .                  | 6  |
| A.2. Coste del Hardware . . . . .                  | 7  |
| A.3. Licencias . . . . .                           | 8  |
| B.1. CU-02 Abrir fichero. . . . .                  | 12 |
| B.2. CU-02 Leer objeto. . . . .                    | 13 |
| B.3. CU-03 Clasificación de datos. . . . .         | 14 |
| B.4. CU-04 Guardar datos en fichero. . . . .       | 15 |
| B.5. CU-05 Modificar dimensiones interfaz. . . . . | 16 |

## *Apéndice A*

---

# **Plan de Proyecto Software**

---

### **A.1. Introducción**

En este apartado se va a detallar la planificación temporal del proyecto, especificando las tareas. También se realizarán estimaciones económicas y la viabilidad legal.

### **A.2. Planificación temporal**

Se ha decidido seguir la metodología *SCRUM* para la gestión del proyecto. Se han aplicado las siguientes características: El desarrollo se ha dividido en iteraciones (sprints), cada una de las cuales ha tenido una duración entre 1 y 2 semanas. Durante cada sprint se irán creando y realizando las diferentes tareas correspondientes a los objetivos fijados al inicio de cada sprint. Las tareas de cada sprint son elaboradas en la reunión que se establece al terminar la iteración anterior. Además de planificar las tareas del siguiente sprint se supervisan las tareas anteriores.

A continuación, se describen brevemente cada uno de los sprints ejecutados durante el desarrollo del proyecto.

#### **Sprint1 - 29 Octubre 2021 - 12 Noviembre 2021**

Sprint que da comienzo al TFG en la modalidad ONLINE.

Indicar que el tema escogido para la realización del proyecto se hizo en abril de 2021, cursando la modalidad presencial de la carrera. La causa de

que no se realizara ningún sprint en ese curso fue ocasionada por la carga lectiva más el acoplamiento de la realización de las prácticas en empresa.

Este Sprint comprende las tareas de:

- Familiarización con *GitHub* (milestone, commits, estadísticas...)
- Añadir *Readme* al repositorio.
- Desarrollo de la memoria del TFG.

## **Sprint2 - 13 Noviembre 2021 - 26 Noviembre 2021**

En el Sprint2 se seguirá con la documentación de la memoria y se barajará crear una interfaz alternativa a la actual con alguna herramienta drag and drop para código python.

Este Sprint comprende las tareas de:

- Conceptos teóricos memoria.
- Técnicas y herramientas memoria.
- Familiarización con los anexos de la memoria.
- Crear interfaz alternativa.

## **Sprint3 - 27 Noviembre 2021 - 10 Diciembre 2021**

En el Sprint3 se seguirá con la documentación de la memoria y además se refactoriza el código de los modelos clasificatorios Random Forest, Regresión Logística y KNN

Este Sprint comprende las tareas de:

- Aspectos relevantes del desarrollo del proyecto.
- Plan de proyecto software.
- Refactorización de código fuente.



### **Sprint4 - 11 Diciembre 2021 - 24 Diciembre 2021**

En el Sprint4 se envía una versión preliminar de la memoria a los tutores, tras una reunión realizada se decide modificar apartados de la memoria.

Este Sprint comprende las tareas de:

- Conceptos teóricos.
- Bibliografía.

### **Sprint5 - 25 Diciembre 2021 - 14 Enero 2022**

En el Sprint5 se seguirá con la documentación de la memoria y además se finaliza la interfaz gráfica que será la encargada de realizar las lecturas de los materiales.

Este Sprint comprende las tareas de:

- Desarrollo de interfaz.
- Añadir «acerca de» en interfaz
- Resumen memoria
- Desarrollo apartados memoria

### **Sprint6 - 15 Enero 2022 - 5 Febrero 2022**

En el Sprint6 se realiza la documentación de los anexos y se modifica la interfaz para implementar una barra de estado.

Este Sprint comprende las tareas de:

- Desarrollo de anexos.
- Añadir barra de herramientas (interfaz).
- Añadir barra de estado (interfaz).
- Crear ejecutable de la aplicación.

### **Sprint7 - 6 Febrero 2022 - 17 Febrero 2022**

El Sprint7 comprende el periodo final del desarrollo del proyecto.

Se han realizado la últimas tareas para la entrega del proyecto:

- Corrección de la documentación.
- Reestructurar repositorio.
- Entrega del proyecto.

Estas tareas finalmente no se cumplieron al estar el proyecto incompleto y se ha decidido retrasar la entrega para la convocatoria del siguiente curso.

### **Sprint8 - 28 Octubre 2022 - 10 Noviembre 2022**

El Sprint8 comprende el periodo inicial para retomar el proyecto en la modalidad presencial de la carrera. Como se han perdido las dos convocatorias del curso 2021/22 se ha retomado el TFG para el curso 2022/23 con vistas a la convocatoria de Enero.

Para recordar el desarrollo y comenzar el proyecto desde el principio se han decidido las siguientes tareas:

- Lectura del proyecto.
- Ajustar los parámetros al radar.
- Crear una nueva biblioteca de observaciones, es decir, volver a realizar todas las lecturas de los materiales.
- Crear nuevos modelos de entrenamiento (*RandomForest*, *KNN*, *SVM*)

### **Sprint9 - 11 Noviembre 2022 - 24 Noviembre 2022**

En el Sprint9 se ha querido introducir nuevos clasificadores actuales e innovadores como son:

- *auto-sklearn*
- *TabPFN*

Con estos dos clasificadores se ha querido mejorar las tasas de acierto obtenidas hasta ahora por los modelos clasificadores más convencionales.

### **Sprint10 - 25 Noviembre 2022 - 8 Diciembre 2022**

En el Sprint 10 nos vamos a centrar en el desarrollo de la memoria y anexos añadiendo los nuevos cambios y modificando en aquello que sea necesario.

### **Sprint11 - 9 Diciembre 2022 - 22 Diciembre 2022**

En periodo del Sprint 11 se ha decidido realizar las siguientes tareas:

- Establecer el patrón de diseño fachada en la aplicación.
- Decidir el modelo clasificador que ejecuta la aplicación.
- Hacer compatible la aplicación para *Ubuntu*.

### **Sprint12 - 22 Diciembre 2022 - 13 Enero 2023**

El Sprint 12 es un periodo largo debido a que comprende los días de Navidad. En este periodo se ha querido seguir con el enfoque en memoria y anexos para llegar a una versión cercana a la final. Además se ha creado un vídeo manual sobre la aplicación para incluir en el proyecto.

- Memoria
- Anexos
- Vídeo manual de uso aplicación.
- Versión final de la aplicación *RadarWave*
- Reestructurar repositorio.

### **Sprint13 - 14 Enero 2023 - 19 Enero 2023**

El Sprint 13 comprende el periodo final del desarrollo del proyecto.

Se han realizado la últimas tareas para la entrega del proyecto:

- Documentación declaración de entrega.
- Preparar tres ejemplares del TFG en soporte informático compatible.
- Copia impresa de la memoria del TFG.

- Dos copias en UBUVirtual de memoria y anexos del TFG.
- Presentación para el día de la defensa.
- Entrega del proyecto.

### A.3. Estudio de viabilidad

#### Viabilidad económica

En este apartado se explicarán los aspectos económicos del proyecto. Se va a tener en cuenta el coste de personal, el software empleado y los equipos utilizados.

#### Costes de personal

En el proyecto solo ha participado una persona por aproximadamente 6 meses. Se considera un salario de:

| Concepto                             | Coste    |
|--------------------------------------|----------|
| Salario mensual neto                 | 1224.75€ |
| Retención IRPF (12 %)                | 180.00€  |
| Seguridad social (Empleado) (6,35 %) | 95.25€   |
| Salario mensual bruto                | 1500.00€ |
| Salario total (6 meses)              | 8500.00€ |
| Seguridad social (Empresa) (29,9 %)  | 448.50€  |
| Coste total mensual de la empresa    | 1794.00€ |

Tabla A.1: Costes de personal

#### Costes de software

A nivel de software no tenemos gastos ya que las librerías son gratuitas y no necesitamos pagar ninguna licencia para el uso del radar.

#### Costes de hardware

En este apartado se describen los costes relacionados con el equipamiento hardware que se ha utilizado para el desarrollo del proyecto. Para calcular

el coste amortizado, se ha tenido en cuenta que el tiempo de uso coincide con la duración del proyecto (6 meses) y que su vida útil gira en torno a los 5 años.

| Concepto                  | Coste   | Coste amortizado |
|---------------------------|---------|------------------|
| Raspberry Pi 4            | 88.40€  | 8.84€            |
| Radar A111                | 52.73€  | 5.27€            |
| Placa XR112               | 176.78€ | 17.67€           |
| Cable flexible para XR112 | 7.99€   | 0.79€            |
| Tarjeta SD                | 4.90€   | 0.49€            |
| Teclado USB               | 9.90€   | 0.99€            |
| Ratón USB                 | 7.64€   | 0.76€            |
| Monitor con HDMI          | 79.99€  | 7.99€            |
| Cable HDMI                | 7.95€   | 0.79€            |
| Coste TOTAL               | 444.23€ | 44.42€           |

Tabla A.2: Coste del Hardware

## Viabilidad legal

Uno de los factores más importantes a tener en cuenta en el desarrollo de un proyecto es escoger el tipo de licencia con el que se distribuirá cada una de sus partes. De esta forma, se define el marco legal en el que se puede utilizar cada parte, es decir, lo que se autoriza a hacer y lo que no.

A la hora de determinar si una biblioteca es válida para nuestro proyecto, se tiene en cuenta si la licencia es compatible con *MIT*. La licencia *MIT* permite que el software sea redistribuido libremente. Se ha empleado esta licencia porque la aplicación donde se incluye este proyecto tiene esta licencia. Al igual que todas las demás licencias que hemos utilizado como *BSD* o *Apache 2.0 opensource* que son libres para su redistribución.

| <b>Librería</b> | <b>Licencia</b>               |
|-----------------|-------------------------------|
| Tensorflow      | Apache 2.0 opensource license |
| NumPy           | Nueva Licencia BSD            |
| Pandas          | Licencia BSD                  |
| Scikit-learn    | Licencia BSD                  |
| Auto-sklearn    | Licencia BSD                  |
| Tcl/Tk          | Licencia BSD                  |
| Socket          | Licencia MIT                  |
| Paramiko        | Licencia LGPL                 |
| matplotlib      | Licencia BSD                  |
| joblib          | Licencia BSD                  |
| TabPFN          | Licencia BSD                  |

Tabla A.3: Licencias

## *Apéndice B*

---

# Especificación de Requisitos

---

### B.1. Introducción

En este apéndice se especifican las necesidades del cliente o de los usuarios, los requisitos y la especificación de los mismos. Se analizan tanto los requisitos funcionales como los no funcionales.

### B.2. Objetivos generales

Los objetivos generales del proyecto se pueden describir en los siguientes puntos:

- El objetivo principal del proyecto es el desarrollo de una aplicación capaz de comunicarse con el radar y realizar lecturas de objetos para identificar distintos materiales.
- Documentar los diferentes elementos que componen el ensamblar y poner en funcionamiento un radar de 60GHz fabricado por *Acconeer* el cual es capaz de identificar, mediante un clasificador, el tipo de material al que pertenece un objeto.
- Crear un procedimiento capaz de extraer determinadas características de las lecturas de los objetos para entrenar un modelo de clasificación automática.
- Poder generar un gráfico de tasa de acierto, el cual sea capaz de mostrar la clasificación de los diferentes materiales empleados.

- Se establecerá una interfaz capaz de poder ser implementada en el sistema operativo Windows, buscando que la aplicación pueda ser implementada en cualquier dispositivo que cuente con este sistema.

## B.3. Catalogo de requisitos

### Requisitos funcionales

Los requerimientos funcionales son aquellos que describen cualquier actividad que deba realizar el sistema o *software*. Se relacionan con los casos de uso.

- **RF-1 Abrir lecturas:** la aplicación debe permitir al usuario abrir lecturas almacenadas para que sean procesadas.
- **RF-2 Reconocimiento de materiales:** la aplicación debe ser capaz de clasificar los datos extraídos de las lecturas.
- **RF-3 Conectar con el radar:** la aplicación debe ser capaz de crear una conexión estable entre el equipo y el radar.
- **RF-4 Iniciar lectura por radar:** la aplicación debe conseguir leer un objeto desde el radar.
- **RF-5 Guardar datos:** la aplicación debe permitir guardar los datos leídos por el radar o abiertos desde el explorador de archivos.
- **RF-6 Minimizar la interfaz:** se debe permitir minimizar la interfaz.
- **RF-7 Expandir la interfaz:** se debe permitir expandir la interfaz manteniendo el aspecto.
- **RF-8 Informe de error:** la aplicación debe informar de un error al usuario no se consiga establecer conexión con el radar.

### Requisitos no funcionales

Los requisitos no funcionales especifican los criterios a seguir, restricciones y condiciones que impone el cliente.

- **RNF-1 Rendimiento:** la aplicación tiene que ser fluida y tener un tiempo de respuesta bajo.



- **RNF-2 Usabilidad:** la aplicación debe ser intuitiva y fácil de entender y utilizar.
- **RNF-3 Mantenibilidad:** la aplicación debe ser fácilmente modificable.
- **RNF-4 Portabilidad:** la aplicación debe poder ejecutarse en distintos equipos con el sistema operativo Windows.

## B.4. Especificación de requisitos

### Diagramas de casos de uso

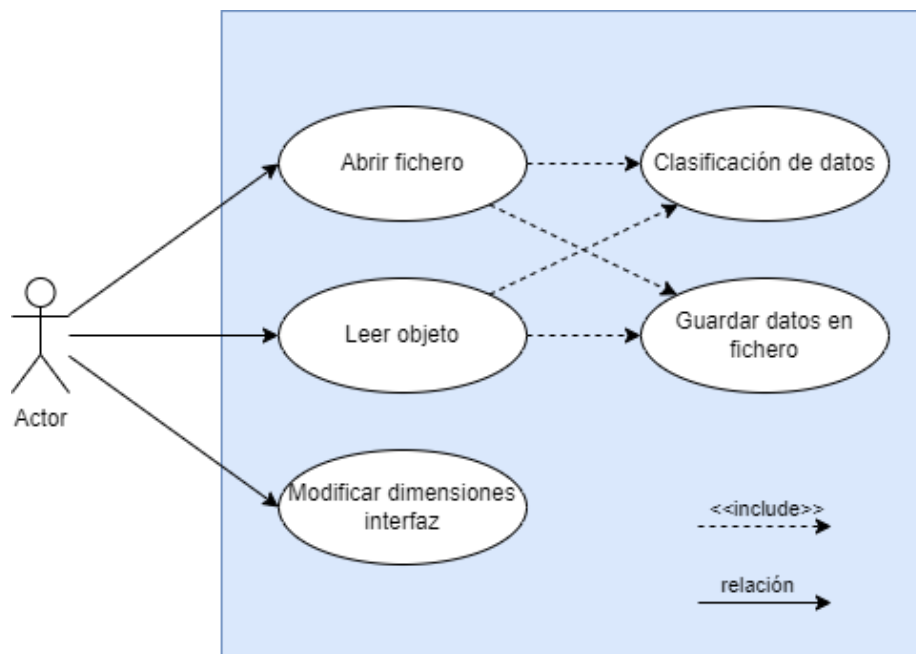


Figura B.1: Diagrama de casos de uso.

El actor será la persona que manejará el *software*.

## Casos de uso

|                             |   |
|-----------------------------|---|
| <b>CU-01</b>                | <b>Abrir fichero.</b>   |
| <b>Requisitos asociados</b> | RF-1  |
| <b>Descripción</b>          | Permite al usuario abrir ficheros con formato <i>.npy</i> , continen lecturas realizadas con anterioridad.  |
| <b>Precondición</b>         | El usuario debe tener iniciada la aplicación.<br>Debe existir un fichero de lecturas anteriores.  |
| <b>Acciones</b>             | <ol style="list-style-type: none"> <li>1. (Opcional) El usuario pulsa en la barra de herramientas en lectura.</li> <li>2. El usuario hace clic en <i>Leer archivo</i> en menu desplegable o icono de documento.</li> <li>3. Se abre una ventana de explorador de archivos.</li> <li>4. Se selecciona el fichero a abrir.</li> </ol> |
| <b>Postcondición</b>        | Se activan las posibilidades de <i>Verificar lectura</i> y <i>Guardar lectura</i> , se iluminan los botones.  |
| <b>Excepciones</b>          | Solo se pueden abrir ficheros con la extension <i>.npy</i>  |
| <b>Importancia</b>          | Alta  |

Tabla B.1: CU-02 Abrir fichero.

|                             |   |
|-----------------------------|---|
| <b>CU-02</b>                | <b>Leer objeto.</b>   |
| <b>Requisitos asociados</b> | RF-3, RF-4, RF-8  |
| <b>Descripción</b>          | Permite al usuario iniciar la conexión con el radar y obtener los datos de una lectura.   |
| <b>Precondición</b>         | El equipo debe estar conectado a la red de internet<br>El usuario debe tener iniciada la aplicación.<br>El radar debe estar conectado a la red eléctrica.<br>El radar debe estar conectado a la misma red de internet que el equipo por <i>Ethernet</i> o <i>Wifi</i>   |
| <b>Acciones</b>             | <ol style="list-style-type: none"> <li>1. (Opcional) El usuario pulsa en la barra de herramientas en lectura.</li> <li>2. El usuario hace clic en <i>Iniciar lectura</i> o icono de radar.</li> <li>3. Se inicia la conexión con el radar.</li> <li>4. Se arranca el servicio de lectura en el radar.</li> <li>5. Se obtienen los datos de la lectura.</li> </ol> |
| <b>Postcondición</b>        | Se activan las posibilidades de <i>Verificar lectura</i> y <i>Guardar lectura</i> , se iluminan los botones.  |
| <b>Excepciones</b>          | No se consigue realizar la conexión con el radar.   |
| <b>Importancia</b>          | Alta  |

Tabla B.2: CU-02 Leer objeto.

| CU-03                       | Clasificación de datos.  |
|-----------------------------|--|
| <b>Requisitos asociados</b> | RF-1, RF-2, RF-3, RF-4, RF-8   |
| <b>Descripción</b>          | Permite al usuario clasificar los datos extraídos de la lectura por radar o de un archivo.   |
| <b>Precondición</b>         | El equipo debe estar conectado a la red de internet<br>El usuario debe tener iniciada la aplicación.<br>El radar debe estar conectado a la red eléctrica.<br>El radar debe estar conectado a la misma red de internet que el equipo por <i>Ethernet</i> o <i>Wifi</i>  |
| <b>Acciones</b>             | <ol style="list-style-type: none"> <li>1. (Opcional) El usuario pulsa en la barra de herramientas en lectura.</li> <li>2. A. El usuario hace clic en <i>Leer archivo</i> o icono de fichero.<br/>B. El usuario hace clic en <i>Iniciar lectura</i> o icono de radar.</li> <li>3. A. Se inicia la conexión con el radar.<br/>B. Se abre el explorador de archivos.</li> <li>4. A. Se arranca el servicio de lectura en el radar.<br/>B. El usuario selecciona el archivo deseado.</li> <li>5. Se obtienen los datos..</li> <li>6. Se iluminan los botones de <i>Verificar lectura</i> y <i>Guardar lectura</i></li> <li>7. El usuario pulsa en <i>Verificar lectura</i></li> <li>8. Se procesan los datos.</li> <li>9. La pantalla principal se actualiza mostrando la pertenencia del objeto a un material.</li> </ol> |
| <b>Postcondición</b>        | Se desactivan las posibilidades de <i>Verificar lectura</i> y <i>Guardar lectura</i> , se bloquean los botones.  |
| <b>Excepciones</b>          | A. No se consigue realizar la conexión con el radar.<br>B. El fichero no tiene el formato correcto.  |
| <b>Importancia</b>          | Alta   |

Tabla B.3: CU-03 Clasificación de datos.

|                             |   |
|-----------------------------|---|
| <b>CU-04</b>                | <b>Guardar datos en fichero.</b>  |
| <b>Requisitos asociados</b> | RF-1, RF-3, RF-4, RF-5, RF-8  |
| <b>Descripción</b>          | Permite al usuario guardar los datos extraídos de la lectura por radar o de un archivo.   |
| <b>Precondición</b>         | El equipo debe estar conectado a la red de internet<br>El usuario debe tener iniciada la aplicación.<br>El radar debe estar conectado a la red eléctrica.<br>El radar debe estar conectado a la misma red de internet que el equipo por <i>Ethernet</i> o <i>Wifi</i>   |
| <b>Acciones</b>             | <ol style="list-style-type: none"> <li>1. (Opcional) El usuario pulsa en la barra de herramientas en lectura.</li> <li>2. A. El usuario hace clic en <i>Leer archivo</i> o icono de fichero.<br/>B. El usuario hace clic en <i>Iniciar lectura</i> o icono de radar.</li> <li>3. A. Se inicia la conexión con el radar.<br/>B. Se abre el explorador de archivos.</li> <li>4. A. Se arranca el servicio de lectura en el radar.<br/>B. El usuario selecciona el archivo deseado.</li> <li>5. Se obtienen los datos.</li> <li>6. Se iluminan los botones de <i>Verificar lectura</i> y <i>Guardar lectura</i></li> <li>7. El usuario pulsa en <i>Guardar lectura</i></li> <li>8. Se abre el explorador de archivos.</li> <li>9. El usuario debe indicar el nombre del fichero.</li> <li>10. El usuario debe guardar el fichero.</li> </ol> |
| <b>Postcondición</b>        | Se desactivan las posibilidades de <i>Verificar lectura</i> y <i>Guardar lectura</i> , se bloquean los botones.   |
| <b>Excepciones</b>          | A. No se consigue realizar la conexión con el radar.<br>B. El fichero no tiene el formato correcto.   |
| <b>Importancia</b>          | Alta  |

Tabla B.4: CU-04 Guardar datos en fichero.

|                             |   |
|-----------------------------|---|
| <b>CU-05</b>                | <b>Modificar dimensiones interfaz.</b>  |
| <b>Requisitos asociados</b> | RF-6, RF-7  |
| <b>Descripción</b>          | Permite al usuario modificar las dimensiones de la interfaz o minimizarla.  |
| <b>Precondición</b>         | El usuario debe tener iniciada la aplicación.   |
| <b>Acciones</b>             | <ol style="list-style-type: none"> <li>1. A. EL usuario desplaza los bordes de la interfaz.<br/>B. El usuario pulsa en el icono de minimizar.</li> <li>2. El usuario establece la dimensión final.</li> </ol> |
| <b>Postcondición</b>        | -   |
| <b>Excepciones</b>          | -   |
| <b>Importancia</b>          | Media   |

---

 Tabla B.5: CU-05 Modificar dimensiones interfaz.

## *Apéndice C*

---

# Especificación de diseño

---

## C.1. Introducción

En este apartado se tratan los aspectos referentes al diseño de la aplicación cumpliendo los requisitos establecidos.

## C.2. Diseño de datos

En el presente proyecto no ha sido necesaria una base de datos por lo que los datos se han almacenado en ficheros y en estructuras de datos en los objetos que se van instanciando en la aplicación.

### Lecturas de objetos

Todas las lecturas iniciales realizadas por el radar se han almacenado en ficheros *Numpy* (*NPY*). El formato de archivo *NPY* es un archivo de datos asociado al lenguaje de programación *Python*. *Numpy* es uno de los paquetes importantes necesarios para ejecutar análisis de datos y cálculos científicos, sobre todo utilizando matrices para su almacenamiento. Los archivos *NPY* almacenan los datos de la matriz en un formato de archivo binario que es recuperable para su transformación, utilización, etc, regenerando la matriz tal cual estaba.

## Conjuntos de *Entrenamiento* y *Test*

Tanto el conjunto de *Entrenamiento* como de *Test* se han almacenado en ficheros *Numpy*. Estos conjuntos fueron creados a partir de 300 archivos *Numpy* que almacenaban las lecturas de cada vista de un objeto determinado.

## Modelo clasificador

La aplicación actualmente está desarrollada para que siempre utilice el mismo modelo ya entrenado, es un modelo creado a partir del clasificador *TabPFN*. El fichero que almacena los datos *modeloTabPFN.pkl* es archivo *PKL* binario creado por la librería *Pickle*, los *pickles* de *Python* representan un objeto *Python* como una cadena de bytes que puede ser almacenada en ficheros o en una base de datos.

El archivo *pickle*, por tanto, es útil para varios propósitos, como el almacenamiento de resultados para que sea utilizado por otro programa *Python* o para crear copias de seguridad.

## Guardar lecturas

Desde la aplicación todas las lecturas que hagamos sobre un objeto se puede almacenar para su posterior utilización tanto en esta aplicación como para otro fin.

Al igual que indicábamos en la subsección *Lecturas de objetos*, toda las lecturas realizadas por el radar se almacenan en ficheros *Numpy* (*NPY*).

## C.3. Diseño procedimental

En este apartado se utiliza un diagrama de secuencia para explicar el funcionamiento de la aplicación.



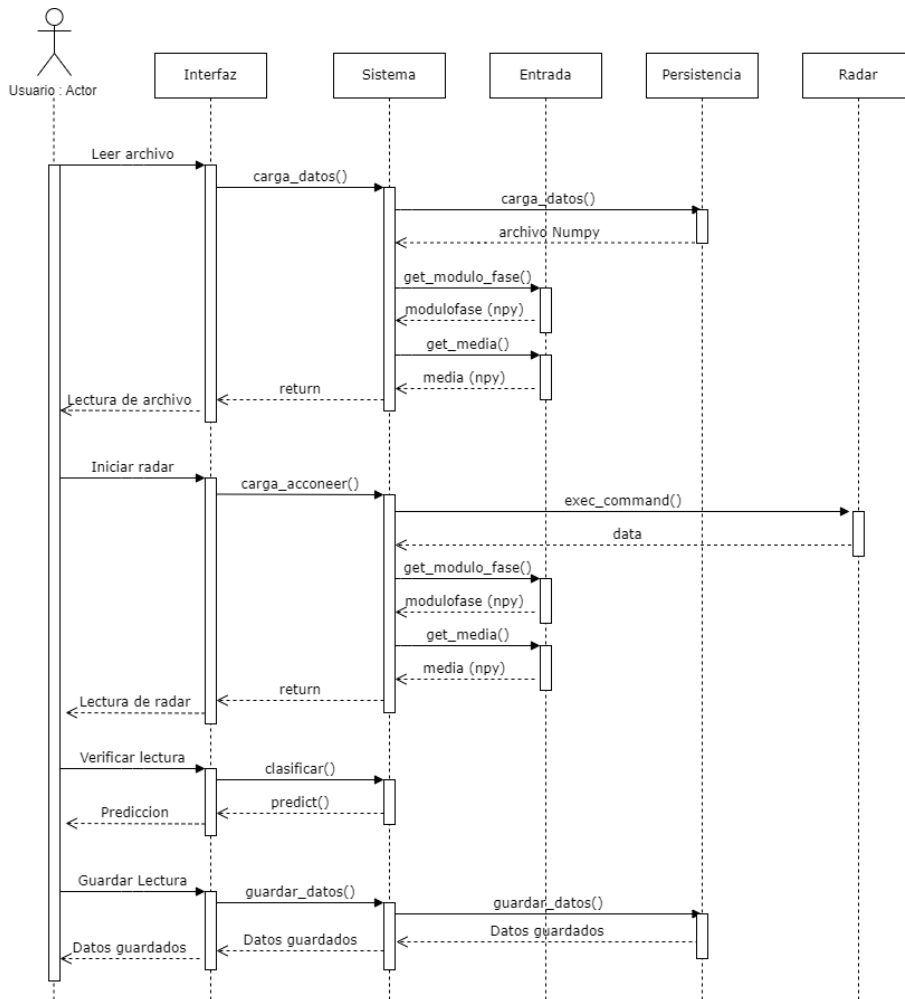


Figura C.1: Diagrama de secuencia.

## C.4. Diseño arquitectónico

En esta sección se explican las consideraciones tomadas al diseñar la arquitectura del proyecto.

Se ha decidido utilizar el patrón de diseño *Facade*<sup>[2]</sup> o *Fachada*, utilizado habitualmente en aplicaciones escritas en *Python*. Es de especial utilidad al trabajar con bibliotecas y *API*<sup>1</sup> complejas.

El nombre de *Fachada* hace referencia a lo que ve el usuario de la aplicación, la interfaz. El patrón de diseño por fachada divide en varias

<sup>1</sup>Interfaz de programación de aplicaciones.

clases el desarrollo de una aplicación. Hay una clase que es la que hace referencia a la fachada, esta se encarga, sobre todo, de implementar la interfaz y las funciones necesarias. El resto de clases de la aplicación se encuentran detrás de la fachada y son conocidas como *Subsistemas*, estas clases comprenden el resto de funciones que harán falta en los diferentes procesos de un programa.

Se aplica cuando:

- Se tiene que acceder a parte de la funcionalidad con un desarrollo complejo.
- Hay tareas que ejecutan el mismo código muy frecuentemente y es conveniente simplificar el código.
- Necesitamos que nuestro desarrollo sea mas legible.
- Hay que simplificar el acceso a las *APIs*.

En la aplicación desarrollada se ha decidido crear una clase fachada y dos clases subsistemas. Las clases son:

- **FacadeRadarWave:** Clase *Fachada* que implementa la interfaz de la aplicación.
- **TratamientoDatos:** Clase que comprende las funciones para el tratamiento de datos escogidos para su clasificación.
- **Verificacion:** Clase empleada para realizar verificaciones básicas en el inicio de la aplicación.

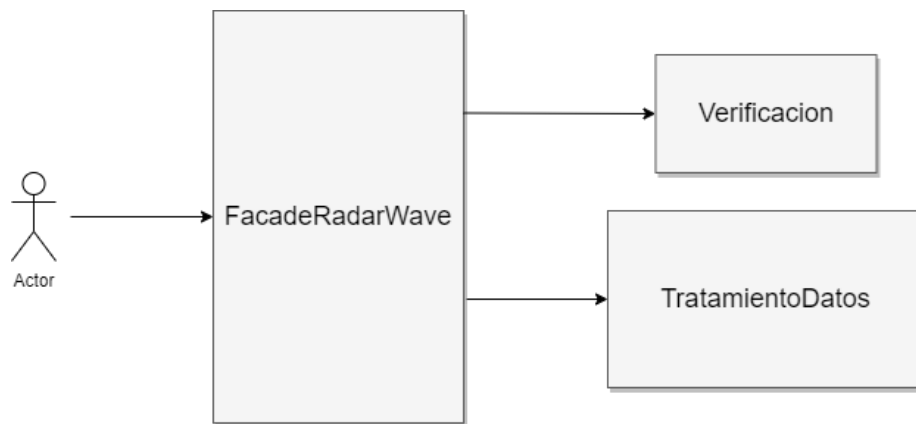


Figura C.2: Patrón de diseño fachada.



## Apéndice *D*

---

# Documentación técnica de programación

---

## D.1. Introducción

En este anexo se detalla la estructura de directorios del proyecto y los conceptos necesarios para la programación del sistema así como para su instalación y ejecución.

## D.2. Estructura de directorios

- `/`: raíz el proyecto, fichero de licencia, modelo clasificador *k-NN*, *interfaz RadarWave.py*, las librerías necesarias *requirements.txt* y archivo comprimido *RadarWave\_v1.0.zip* con la primera versión de la aplicación.
- `/acconeer-python-exploration/`: esta carpeta incluye la librería e interfaz de *Acconeer*.
- `/iconos/`: contiene los iconos de la interfaz.
- `/Latex/`: documentación del proyecto memoria y anexos.
- `/Latex/img/`: imágenes utilizadas en la documentación.
- `/Latex/ltx/`: documentación en formato  $\text{\LaTeX}$ .
- `/scripts/`: contiene los ficheros que se han utilizado para leer los materiales, crear las particiones de datos, los conjuntos de entrenamiento y test, todos los modelos clasificadores utilizados en el proyecto.
- `/scripts/iconos/`: contiene los iconos de la interfaz.
- `/scripts/Materiales/`: contiene todas las lecturas de los materiales.

- `/scripts/Materiales/Fotos/`: fotos que identifican los objetos leídos.

### D.3. Manual del programador

Los entornos para el desarrollo del código durante la realización de proyecto han sido las siguientes:

- *Jupyter*
- *Colab*

#### Entorno de programación

El proyecto se ha desarrollado en los cuadernos de *Jupyter Notebook* y *Colab* mediante código y librerías *Python*.



Figura D.1: Entorno Jupyter.

Se ha decidido utilizar este entorno debido a que *Jupyter* o *Colab* siempre será 100 % de código abierto, de uso gratuito para todos. Es un entorno informático interactivo basado en la web para crear documentos de *Jupyter Notebook*.

Además este entorno ya resultaba familiar debido a su uso en la Universidad de Burgos.

### D.4. Desarrollo del código

En este apartado se muestra el desarrollo creado para la aplicación.

## Transformación de las características

Una vez extraídas las características de los materiales se han realizado una serie de transformaciones en las lecturas, estas devuelven datos mostrados como números reales.

Se ha creado una función que a partir del nombre del fichero *npz* cargue los datos, los redimensione y los devuelva.

Función *get\_data* devuelve un array de 2 dimensiones (2D) al que llamaremos *datos\_bruto*.

Los test de esta función son:

- Comprobar que devuelve un array 2D
- Comprobar que la segunda dimensión tiene tamaño 291
- Comprobar que la matriz tiene una parte real y que tiene una parte imaginaria

```
def get_data(url):  
    try:  
        diccionario = np.load(url, allow_pickle=True).item()  
        data = diccionario.get('sweep_data').get('data')  
        data = data.reshape(data.shape[0], data.shape[2])  
    except:  
        data = np.load(url)  
    return data
```

Figura D.2: Función *get\_data*

Una función *get\_modulo\_fase* que a partir del array 2D anterior, obtenga un array 2D, con el doble de anchura. Por ejemplo, si *datos\_bruto* es de  $300 \times 291$ , *modulo\_fase* será de  $600 \times 291$ .

Esa función obtendrá el módulo del número complejo y la fase del número complejo. El módulo será una matriz de  $300 \times 291$  y la fase también. Será necesario concatenarlas "horizontalmente".

```
def get_modulo_fase(data):
    modulo = abs(data)
    fase = []
    for i in data:
        fila = []
        for j in i:
            fila.append(phase(j)) #Fase
        fase.append(fila)
    fase = np.asarray(fase)
    modulo_fase = np.concatenate((modulo, fase), axis=0)
    return np.asarray(modulo_fase)
```

Figura D.3: Función *get\_modulo\_fase*

A partir de los datos del modulo y la fase se obtiene la media. La comprobación sería que devuelva un array de 1 dimensión de tamaño 582(291x2).

```
def get_media(modulo_fase):
    traspuesta = np.transpose(modulo_fase) #Obtenemos la matriz traspuesta(291x600)

    #Separamos la traspuesta en modulo y fase
    t_modulo = traspuesta[:,int(traspuesta.shape[1]/2)]
    t_fase = traspuesta[:,int(traspuesta.shape[1]/2):]

    media = []
    for i in t_modulo:
        media.append(i.mean())
    for j in t_fase:
        media.append(j.mean())

    return np.asarray(media) #Devuelve medias modulo y fase
```

Figura D.4: Función *get\_media*

Finalmente se debían crear las particiones para el entrenamiento del modelo. Para ello se crea una función encargada de obtener las siguientes particiones separadas en entrenamiento (*part\_train*) y test (*part\_test*).

#### Partición 1

Entrenamiento: Carton Vista2-Vista10 + Plastico Vista2-Vista10  
+ Cristal Vista2-Vista10

Test: Carton Vista1 + Plastico Vista1 + Cristal Vista1

#### Partición 2

Entrenamiento: Carton Vista1,Vista3-Vista10 + Plastico Vista1,Vista3-Vista10  
+ Cristal Vista1,Vista3-Vista10



Test: Carton Vista2 + Plastico Vista2 + Cristal Vista2

...

Partición 10

Entrenamiento: Carton Vista1-Vista9 + Plastico Vista1-Vista9  
+ Cristal Vista1-Vista9

Test: Carton Vista10 + Plastico Vista10 + Cristal Vista10

Testamos esta función:

- Sumando el número de filas de test1 y train1 sea igual que sumando el número de filas de test2 y train2.
- Se comprueba el tamaño de varias vistas igual a 582.

Dentro del repositorio de GitHub del proyecto hay un fichero llamado *GenerarParticiones.ipynb* que incluye el desarrollo de todas estas funciones así como comentarios interesantes.

Una vez obtenidos los datos de las lecturas se comienza con la creación de varios modelos clasificadores, estos son:

- *Random Forest*
- *KNN*
- *SVM*
- *auto-sklearn*
- *TabPFN*

Se ha decidido optar por *KNN* para generar el modelo que utilizará la interfaz desarrollada. Esta decisión se debe a la falta de compatibilidad de *auto-sklearn* y *TabPFN* (basado en *auto-sklearn*) con el sistema operativo *Windows*. Actualmente *auto-sklearn* es compatible con sistemas como *Linux* y el desarrollo creado se quiere utilizar tanto en *Windows* como en *Linux*.

Finalmente, con el clasificador escogido creamos el modelo que emplea la aplicación a la hora de clasificar nuevos objetos leídos por el radar.

## Interfaz

El desarrollo del código de la interfaz lo podemos encontrar en el cuaderno *RadarWave.ipynb*.

Para la generación de la interfaz se ha utilizado la librería *Tkinter*, es una librería dedicada generalmente para el desarrollo de interfaces.

La aplicación generada para el proyecto ha sido llamada ***RadarWave***, debe su nombre a las ondas<sup>1</sup> generadas por los radares.

Para el desarrollo del código se ha decidido utilizar el patrón de diseño *Facade*[2] o *Fachada*, utilizado habitualmente en aplicaciones escritas en *Python*. Es de especial utilidad al trabajar con bibliotecas y *API*<sup>2</sup> complejas.

En la aplicación desarrollada se ha decidido crear una clase fachada y dos clases subsistemas. Las clases son:

- **FacadeRadarWave**: Clase *Fachada* que implementa la interfaz de la aplicación.
- **TratamientoDatos**: Clase que comprende las funciones para el tratamiento de datos escogidos para su clasificación.
- **Verificacion**: Clase empleada para realizar verificaciones básicas en el inicio de la aplicación.

Se ha decidido no incorporar el desarrollo del código de la interfaz en los anexos, se cree que es más comprensible su lectura y razonamiento dentro del cuaderno de *Jupyter RadarWave.ipynb*.

## D.5. Compilación, instalación y ejecución del proyecto

### Compilación

Al ser una aplicación *beta* el archivo donde se encuentra desarrollada la interfaz *RadarWave.ipynb* se ha exportado en formato *.py* de *Python*. Además se ha creado un archivo llamado *requirements.txt* que será utilizado para instalar las librerías necesarias en nuestro equipo.

---

<sup>1</sup>Onda en inglés es wave.

<sup>2</sup>Interfaz de programación de aplicaciones.

Todos los archivos necesarios se han comprimido para su exportación en el fichero *RadarWave\_v1.0.zip*

## Instalación y ejecución en *Windows*

Consultar la sección [E.2](#).

## Instalación y ejecución en *Linux*

Consultar la sección [E.3](#).

## D.6. Pruebas del sistema

Durante el desarrollo del código del programa se fueron implementando un conjunto de pruebas para comprobar el correcto tratamiento de los datos de entrenamiento.

Estas pruebas las podemos encontrar en el fichero *GenerarParticiones.ipynb*, aquí se crearon varias pruebas para comprobar que los resultados eran los correctos. En la sección **Desarrollo del código** del anexo actual hemos hablado de las pruebas o test que se han formulado.

Los test realizados ha sido:

```
#comprobar que devuelve un array 2D
assert(len(get_data('Materiales\C01_V01.npy')).shape) == 2)

#comprobar que la segunda dimensión tiene tamaño 291
assert(get_data('Materiales\C01_V01.npy').shape[1] == 291)

#comprobar que la matriz tiene una parte real
assert((get_data('Materiales\C01_V01.npy').real.mean) != None)

#comprobar que la matriz tiene una parte imaginaria
assert((get_data('Materiales\C01_V01.npy').imag.mean) != None)

assert(get_media(modulo_fase).shape[0] == 582) #Dimensión de tamaño = 582

assert(len(get_media(modulo_fase).shape) == 1) #Array de 1 dimensión

#Train1 60 + Test1 30 = Train2 60 + Test2 30
```

```
assert(len(train[0]) + len(test[0])) == (len(train[1]) + len(test[1]))

#Train1 60 + Test1 30 = Train10 60 + Test10 30
assert(len(train[0]) + len(test[0])) == (len(train[9]) + len(test[9]))

#582 son las características de una vista
assert(len(train[0][0]) == 582)

#Las vistas tienen todas un tamaño de 582 (291 x 2)(modulo x fase)
assert(len(train[0][0]) == len(train[0][9]) == len(test[5][7]))
```

## Apéndice *E*

---

# Documentación de instalación

---

### E.1. Introducción

En este apartado se explica el proceso de instalación y ejecución de la aplicación desarrollada para el proyecto. Se incluye el manual de instalación tanto para *Windows* como para Linux.

### E.2. Instalación y ejecución en *Windows*

La siguiente instalación se ha realizado en un equipo con las siguientes características:

- **SISTEMA OPERATIVO:** *Microsoft Windows 10 Pro*
- **PROCESADOR:** *Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz*
- **MEMORIA RAM:** 12 GB
- **TARJETA GRÁFICA:** *NVIDIA GeForce 930M*

En realidad la aplicación *RadarWave* no hace falta que se instale, pero si necesitamos instalar las librerías utilizadas por el programa.

Antes de proceder con la instalación de las librerías vamos a verificar si tenemos instalado Python, de no ser así procedemos a su instalación.

- Descargamos el instalador desde la web de *Python*.
- Ejecutamos el archivo descargado.

- En la pantalla de inicio de la instalación marcamos la opción de añadir *Python* al *PATH*, si no luego se tendrá que hacer manualmente.
- Pulsamos en instalar ahora (*install now*)

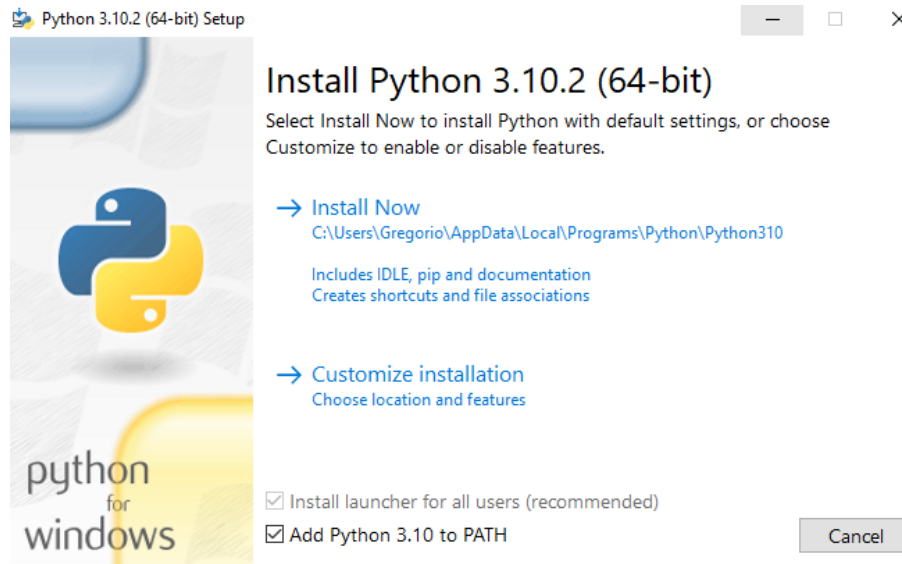
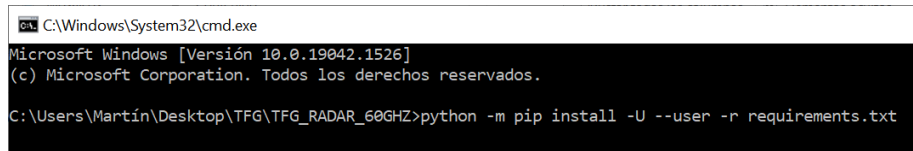


Figura E.1: Instalación de *Python*.

Una vez instalado *Python* lo primero que haremos será descargar y descomprimir el fichero *RadarWave\_v1.0.zip*, se puede descargar desde [https://github.com/mecyc/TFG\\_RADAR\\_60GHZ/blob/main/RadarWave\\_v1.0.zip](https://github.com/mecyc/TFG_RADAR_60GHZ/blob/main/RadarWave_v1.0.zip)

Procedemos a instalar las librerías, están incluidas en el archivo *requirements.txt*. Primero instalamos *setuptools* que facilita el empaquetado de proyectos de *Python* y a continuación el resto, para ello debemos abrir la consola y lanzar las siguientes instrucciones.

```
python -m pip install -U --user setuptools wheel
python -m pip install -U --user -r requirements.txt
```



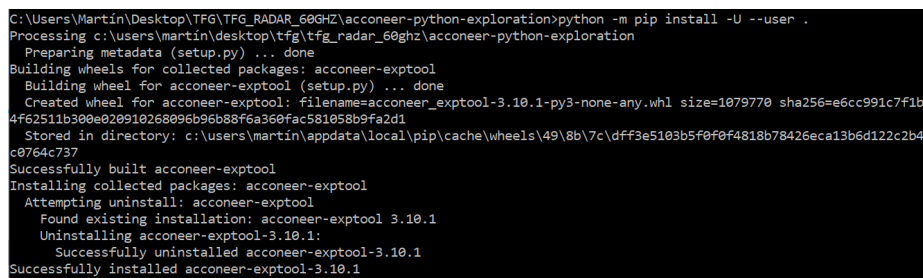
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19042.1526]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Martín\Desktop\TFG\TFG_RADAR_60GHZ>python -m pip install -U --user -r requirements.txt
```

Figura E.2: Instalar librerías.

Finalmente se debe instalar la librería de *Acconeer*, para ello descargamos del repositorio del TFG la carpeta *aconeer-python-exploration*. Una vez descargada abrimos el terminal dentro de la carpeta y ejecutamos la siguiente orden.<sup>[1]</sup>

```
python -m pip install -U --user .
```

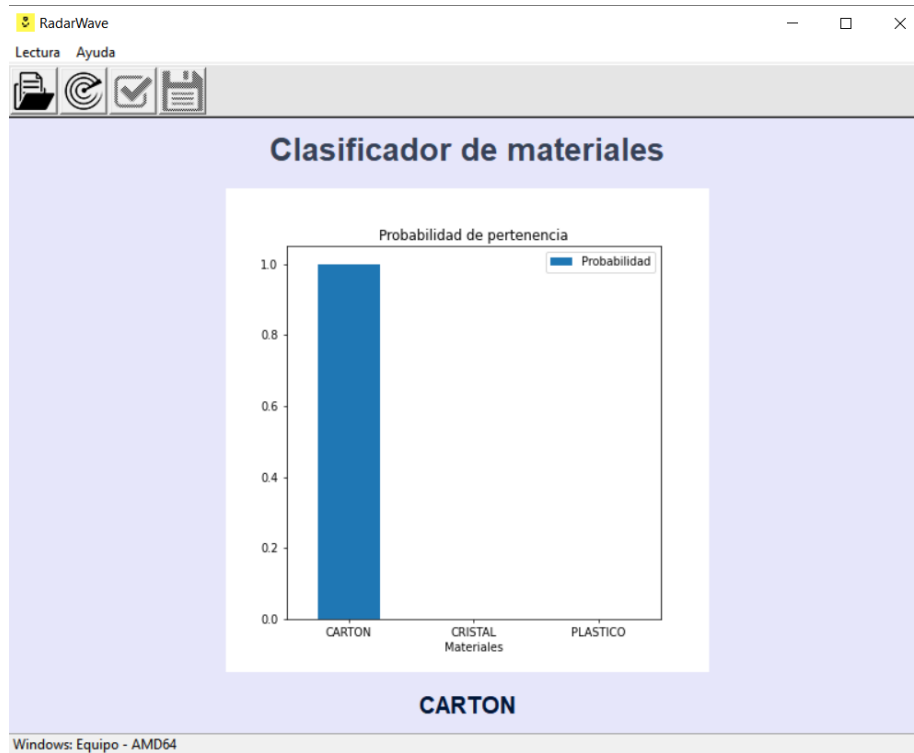


```
C:\Users\Martín\Desktop\TFG\TFG_RADAR_60GHZ\aconeer-python-exploration>python -m pip install -U --user .
Processing c:\users\martin\desktop\tfg\tfg_radar_60ghz\aconeer-python-exploration
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: acconeer-exptool
  Building wheel for acconeer-exptool (setup.py) ... done
  Created wheel for acconeer-exptool: filename=acconeer_exptool-3.10.1-py3-none-any.whl size=1079770 sha256=e6cc991c7f1b4f62511b300e020910268096b96b88f6a360fac581058b9fa2d1
  Stored in directory: c:\users\martin\appdata\local\pip\cache\wheels\49\8b\7c\dff3e5103b5f0f0f4818b78426eca13b6d122c2b4c0764c737
Successfully built acconeer-exptool
Installing collected packages: acconeer-exptool
  Attempting uninstall: acconeer-exptool
    Found existing installation: acconeer-exptool 3.10.1
    Uninstalling acconeer-exptool-3.10.1:
      Successfully uninstalled acconeer-exptool-3.10.1
Successfully installed acconeer-exptool-3.10.1
```

Figura E.3: Instalar librería *Acconeer*.

Una vez instaladas las librerías procedemos a ejecutar el programa haciendo doble clic en el fichero *RadarWave.py* donde está el programa.

Programa ejecutado y probado en *Windows*:

Figura E.4: *RadarWave* en *Windows*.

### E.3. Instalación y ejecución en *Linux*

La siguiente instalación se ha realizado en un equipo virtualizado en *Oracle VM VirtualBox 6.1* con las siguientes características:

- **SISTEMA OPERATIVO:** *Ubuntu 22.04 LTS*
- **PROCESADOR:** *Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz*
- **MEMORIA RAM:** 4 GB

En realidad la aplicación *RadarWave* no hace falta que se instale, pero si necesitamos instalar las librerías utilizadas por el programa.

El sistema operativo que utilizamos es *Ubuntu*, una distribución Linux, por lo que *Python* está integrado por defecto y no hace falta su instalación. Se recomienda tener actualizado *Python* a la última versión.



Antes de instalar las librerías necesarias instalamos *pip*, es un sistema de gestión de paquetes que nos ayuda a instalar las librerías, y *setuptools*, facilita el empaquetado de proyectos de Python.

```
sudo apt install python3-pip
pip3 install setuptools-rust
```

Una vez terminada la instalación procedemos a descargar y descomprimir el fichero *RadarWave\_v1.0.zip*, se puede descargar desde [https://github.com/mecyc/TFG\\_RADAR\\_60GHZ/blob/main/RadarWave\\_v1.0.zip](https://github.com/mecyc/TFG_RADAR_60GHZ/blob/main/RadarWave_v1.0.zip)

A continuación se debe instalar las librerías que usa la interfaz desarrollada. Indicar que para ejecutar el siguiente comando nos debemos situar en la dirección o *PATH* donde se encuentra el archivo *requirements.txt*, lo encontramos dentro del fichero que acabamos de descomprimir.

El comando sería el siguiente:

```
pip3 install -r requirements.txt
```

Si existiera algún problema al instalar la librería de *Tkinter* (tk) se deben lanzar los siguientes comandos:

```
sudo apt-get install python3-tk
sudo apt-get install python3-pil python3-pil.imagetk
```

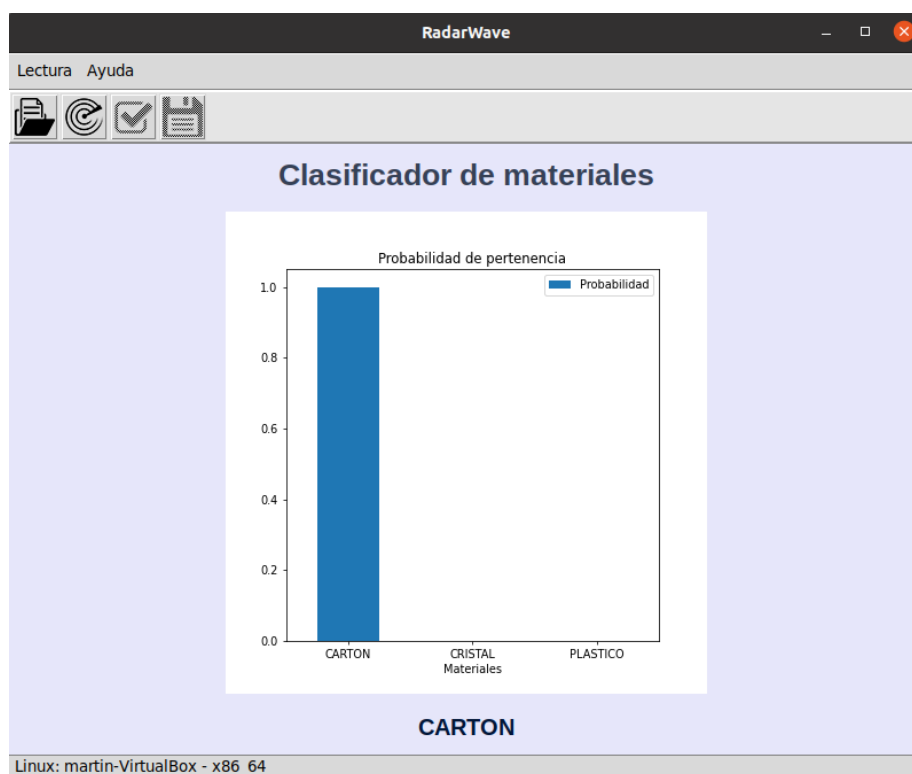
Finalmente se debe instalar la librería de *Acconeer*, para ello descargamos del repositorio del TFG la carpeta *aconner-python-exploration*. Una vez descargada abrimos el terminal dentro de la carpeta y ejecutamos la siguiente orden.

```
python3 setup.py install
```

Si queremos ejecutar *RadarWave* en *Ubuntu* deber ser lanzando el siguiente comando en la misma ruta del archivo *RadarWave.py*.

```
python3 RadarWave.py
```

Programa ejecutado y probado en *Ubuntu*:

Figura E.5: *RadarWave* en *Ubuntu*.

## *Apéndice F*

---

# Documentación de usuario

---

### F.1. Introducción

En este apartado se explican los requerimientos de la aplicación para ser ejecutada, el proceso de instalación y el manual de usuario con las indicaciones para utilizar correctamente la aplicación.

### F.2. Requisitos de usuarios

En esta sección se indican los requisitos para utilizar la aplicación.

Si vamos a clasificar lecturas sin radar necesitamos:

- Lecturas guardadas en el equipo en formato *.npy*

Si deseamos realizar lecturas desde radar necesitamos el siguiente hardware:

- Raspberry Pi 4
- Radar A111
- Placa XR112
- Cable flexible para XR112
- Tarjeta SD
- Teclado USB

- Ratón USB
- Monitor con HDMI
- Cable HDMI
- Adaptador mini HDMI a HDMI

Y además necesitamos la siguiente configuración:

- Equipo conectado a red *A*.
- Radar conectado a la misma red *A*.
- Antes de realizar una lectura por radar necesitamos conectarle a la red eléctrica unos 20 segundos antes de iniciar la lectura.

## Instalación y ejecución en *Windows*

Consultar la sección [E.2](#).

## Instalación y ejecución en *Linux*

Consultar la sección [E.3](#).

## F.3. Manual del usuario

En esta sección indicaremos al usuario cómo usar la aplicación.

Para comenzar debemos iniciar la aplicación abriendo el archivo *Radar-Wave.py*.

Nos muestra la siguiente pantalla:

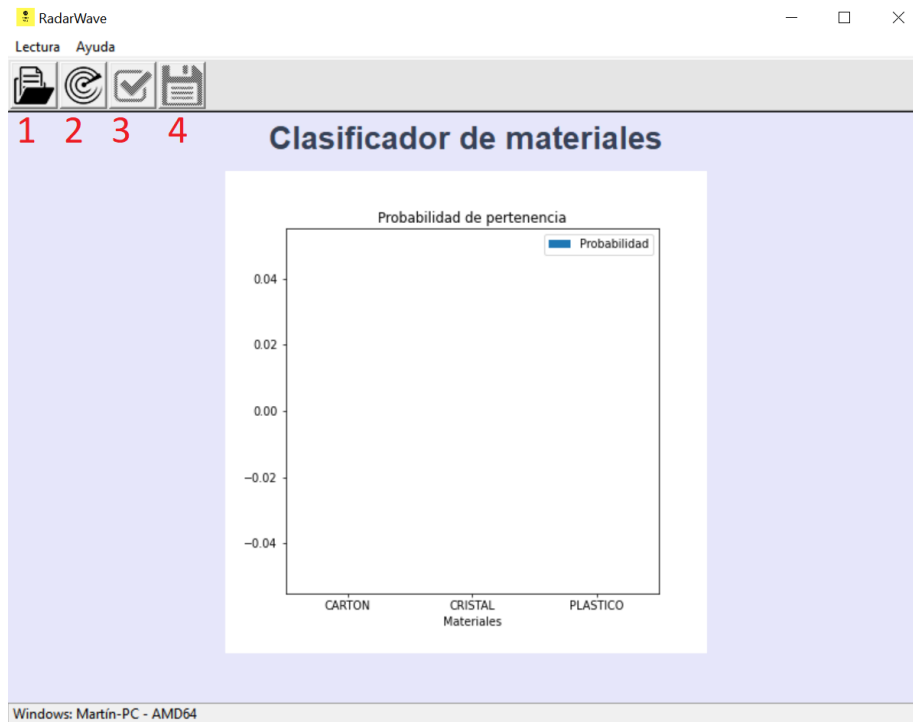


Figura F.1: Pantalla principal RadarWave.

El uso de la aplicación es muy sencillo, tenemos cuatro botones en la parte superior, de izquierda a derecha son:

- 1. Abrir archivo de lectura
- 2. Iniciar lectura por radar
- 3. Clasificar la lectura
- 4. Guardar datos de lectura.

Si pulsamos en el botón 1 se abre el explorador de archivos para seleccionar una lectura a clasificar.

Abrimos el fichero y ya tenemos los datos dentro de la aplicación, para clasificarlos pulsamos el botón 3. Si lo que queremos es guardarlos pulsamos en el botón 4.

Para realizar una lectura por radar necesitamos iniciar la *Raspberry* junto con el radar 20 segundos antes de lanzar la lectura, una vez hecho

esto pulsamos en el botón 2. Tras esto se iluminan los botones 3 y 4. Para clasificarlos pulsamos el botón 3. Si lo que queremos es guardarlos pulsamos en el botón 4.

**Vídeo elaborado como demostración y manual de usuario de la aplicación RadarWave:** [https://youtu.be/le\\_5lstr7-I](https://youtu.be/le_5lstr7-I)

---

# Bibliografía

---

- [1] Radar sensor introduction — acconeer-python-exploration documentation, Jan 2021. [Online; accessed 11. Jan. 2022].
- [2] Facade en Python / Patrones de diseño, December 2022. [Online; accessed 16. Dec. 2022].