Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

Estimación Recursiva de Niveles RSS Dinámicos Mediante Crowdsourcing y Métodos Kernel en Python

Autor: Daniel Vela Calderón

Tutor: Juan José Murillo Fuentes

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Telecomunicación

Estimación Recursiva de Niveles RSS Dinámicos Mediante Crowdsourcing y Métodos Kernel en Python

Autor:

Daniel Vela Calderón

Tutor:

Juan José Murillo Fuentes

Catedrático Universidad

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Máster: Estimación Recursiva de Niveles RSS Dinámicos Mediante Crowdsourcing y Métodos Kernel en Python

Autor: Daniel Vela Calderón
Tutor: Juan José Murillo Fuentes

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

El guardar un espacio para agradecimientos siempre me ha parecido una de las mejores prácticas que se pueden realizar en torno a cualquier trabajo. Es por ello que me gustaría emplear este apartado para agradecer el esfuerzo que han realizado ciertas personas con el fin de que este trabajo, así como el resto de la titulación que enmarca a este proyecto, haya podido salir adelante satisfactoriamente.

Primeramente me gustaría agradecer al tutor de este proyecto, Juan José Murillo Fuentes por su ayuda a lo largo de los meses de duración del trabajo. Destaco en este punto la amabilidad mostrada, especialmente durante los primeros meses de la crisis sanitaria originada por el COVID-19. A pesar de no ser tiempos fáciles, Juan José siempré mostró la atención necesaria para que el trabajo se pudiera realizar dentro de los plazos marcados inicialmente. En este párrafo me gustaría mencionar a otros dos profesores, Juan Antonio Becerra Gónzalez e Hipólito Guzmán Miranda, por su ayuda y confianza depositada durante mis años de estudio. Sus consejos siempre fueron recibidos desde la absoluta admiración que tengo hacia ellos.

Por otro lado, agradezco a aquellos compañeros de clase los cuales después de tantos años han llegado a convertirse en **amigos**. Entre ellos, destaco a Guillermo Palomino, Abraham Pérez, Diego López e Ildefonso Jiménez por su cercanía en estos años de estudio. He aprendido mucho de todos ellos y estoy convencido de que tendrán una exitosa carrera profesional por delante.

Por último, me gustaría agradecer a mis padres, Diego y Mª Carmen por ser dos pilares tan fundamentales de mi vida, a mi hermano mayor Diego por su ejemplo y a mis dos hermanas pequeñas, Miriam y Ester por su enorme paciencia conmigo. No me quiero olvidar tampoco de aquellos amigos que han estado continuamente apoyándome y creyendo en mi y en mis capacidades y de entre los cuales, destaco a Abi, Dani, Miguel y Ana.

A todos vosotros, **muchas gracias**.

Daniel Vela Calderón
Alumno del Máster Universitario en Ingeniería de Telecomunicación

Sevilla, 2021

Resumen

Es bien sabido que el conocimiento de las comunicaciones inalámbricas supone uno de los principales esfuerzos a lo largo de la carrera de un ingeniero en telecomunicaciones. Se trata de un campo tan usado en la actualidad pero a la vez tan cambiante que hace que todo ingeniero dedicado a esta especialidad se encuentre en continua formación. Hoy en día, y desde hace algunos años, se está explorando cada vez con una mayor frecuencia la introducción del aprendizaje máquina, mayormente conocido como *machine learning* (ML), en este importante campo dentro de las comunicaciones. Este proyecto evalúa la **aplicación de cierto algoritmo de ML en un escenario concreto con el fin de estimar la señal recibida en distintos puntos de dicho escenario**, a partir de la información proporcionada por sensores de bajo coste cuyas medidas tienen una precisión muy baja. Las características del escenario de aplicación del algoritmo así como el modelo de propagación de señal que permite calcular la intensidad de campo recibido en los puntos del espacio son dos de los aspectos más interesantes en este proyecto. Para finalizar el apartado teórico, se mostrará las implicaciones y las consideraciones principales que tiene el algoritmo de ML seleccionado en este escenario.

Aparte del aspecto teórico, este trabajo incluye una gran aportación desde el punto de vista de la programación de las ecuaciones teóricas. Se trata de un capítulo muy importante ya que realiza una asociación entre las ecuaciones teóricas y el lenguaje de programación usado. Además, dentro de este apartado se potencia el uso de librerías ya realizadas previamente con el fin de facilitar la aplicación del algoritmo de ML usado en este escenario. Estas librerías son presentadas y analizadas a lo largo del mencionado capítulo.

Por último, este trabajo muestra las prestaciones del algoritmo de ML implementado a partir del cálculo del error entre las estimaciones realizadas y los valores de campo teóricos. A lo largo de la memoria se incluyen los aspectos más interesantes de las simulaciones así como los resultados obtenidos con el fin de valorar la fiabilidad y eficacia del algoritmo en este escenario concreto.

Abstract

It is well known that knowledge of wireless communications is one of the main efforts throughout the career of a telecommunications engineer. It is a field that is not only widely used nowadays but also so changing, that it means that every engineer dedicated to this specialty need to be in continuous training. Nowadays, and for some years now, the introduction of machine learning is being explored with increasing frequency in this important field within communications. This project assesses the **application of a certain ML algorithm in a specific scenario in order to estimate the received signal strength (RSS) at different points in that scenario**, from the information provided by low-cost sensors whose measurements have a very low precise. The characteristics of the algorithm application scenario as well as the signal propagation model that allows calculating the received field strength at points in space are two of the most interesting aspects of this project. To finish the theoretical section, the implications and main considerations of the ML algorithm selected in this scenario will be shown.

Apart from the theoretical aspect, this work includes a great contribution from the point of view of the programming of theoretical equations. This is a very important chapter since it makes an association between the theoretical equations and the programming language used. Furthermore, within this section, the use of libraries that are already made previously is promoted in order to facilitate the application of the ML algorithm used in this scenario. These libraries are presented and analyzed throughout the mentioned chapter.

Lastly, this work shows the performance of the ML algorithm implemented from the calculation of the mean square error (MSE) between the estimates made and the theoretical field values. Throughout the report, the most interesting aspects of the simulations are included as well as the results obtained in order to assess the reliability and effectiveness of the algorithm in this specific scenario.

Índice Abreviado

Índice

Notación

$\mathbf{A}^{\top}$	Transpuesto de $\mathbf{A}$
$\mathbf{A}^{-1}$	Inversa de la matriz $\mathbf{A}$
$\mathbf{A}^{\mathrm{H}}$	Transpuesto y conjugado de $\mathbf{A}$
$\mathbf{A}^{*}$	Conjugado
e	número e
$\mathrm{e}^{\mathrm{j}x}$	Exponencial compleja
$\mathrm{e}^{\mathrm{j}2\pi x}$	Exponencial compleja con 2π
$\mathrm{e}^{-\mathrm{j}x}$	Exponencial compleja negativa
$\mathrm{e}^{-\mathrm{j}2\pi x}$	Exponencial compleja negativa con 2π
C_{XY}	covarianza de dos variables aleatorias reales X e Y
R_{XY}	correlación de dos variables aleatorias reales X e Y
ρ_{XY}	Coeficiente de correlación de las variables aleatorias reales X e Y
$F_X(\cdot)$	Función de distribución de la variable aleatoria X
$f_X(\cdot)$	Función densidad de probabilidad de la variable aleatoria X
$\Pr(A)$	Probabilidad del suceso A
$E[X]$	Valor esperado de la variable aleatoria X
σ_X^2	Varianza de la variable aleatoria X
$\sim f_X(x)$	Distribuido siguiendo la función densidad de probabilidad $f_X(x)$
$\mathcal{N}\left(m_X, \sigma_X^2\right)$	Distribución gaussiana para la variable aleatoria X, de media m_X y varianza σ_X^2
$\mathbf{I}_n$	Matriz identidad de dimensión n
$\mathrm{diag}(\mathbf{x})$	Matriz diagonal a partir del vector $\mathbf{x}$
$\mathrm{diag}(\mathbf{A})$	Vector diagonal de la matriz $\mathbf{A}$
MSE	Minimum square error
$\mathbf{x}_i, i = 1,2,\ldots,n$	Elementos i, de 1 a n, del vector $\mathbf{x}$
$\leqslant$	Menor o igual
$\geqslant$	Mayor o igual
$\frac{a}{b}$	Fracción con estilo pequeño, a/b
Δ	Incremento
$\mathbf{C}_{\mathbf{x}}$	Matriz de covarianza de $\mathbf{x}$
$\mathbf{R}_{\mathbf{x}}$	Matriz de correlación de $\mathbf{x}$

1 Introducción

La ciencia es una forma de pensar, mucho más que un cuerpo de conocimientos.

Carl Sagan

A lo largo de este capítulo se va a llevar a cabo una introducción del proyecto realizado como Trabajo Fin de Máster. Es bien sabido que las comunicaciones inalámbricas han crecido de manera muy significativa en los últimos años, siendo por tanto el uso del espectro y la detección de las comunicaciones llevadas a cabo sobre determinadas frecuencias, dos de los campos más estudiados en telecomunicaciones. A lo largo de las próximas secciones se profundizará más en estos dos puntos. Además, se presentará la importancia que tiene actualmente el *Machine Learning* en las telecomunicaciones, así como en otros aspectos de la ingeniería actualmente. Finalmente, se lleva a cabo una breve introducción del trabajo realizado de una manera más específica, así como una presentación de los distintos capítulos que forma esta memoria.

1.1 Uso eficiente del espectro radioeléctrico

Aunque este trabajo está centrado en el segundo de los campos mencionados anteriormente, es necesario invertir algunas líneas presentando peculiaridades acerca del primero debido a su importancia en las telecomunicaciones. El espectro radioeléctrico es un recurso limitado y costoso por lo que se han invertido numerosos esfuerzos para poder explotar este recurso de la manera más eficiente posible. Las comunicaciones inalámbricas han formado parte de la rutina de los seres humanos desde hace algunos años, y actualmente no se podría concebir una sociedad desarrollada sin este tipo de comunicaciones. Es por ello que la demanda a la hora de acceder a este recurso se ha visto multiplicada en los últimos años, surgiendo de esta manera técnicas que han ido evolucionando a lo largo del tiempo y que han facilitado el acceso a este recurso de una manera eficiente. Por nombrar algunas de esta técnicas destaca la llamada *spatial spectrum* usada en comunicaciones de televisión digital terrestre [1] o la diversidad especial usada en comunicaciones móviles [2]. Aunque se podría profundizar más en estas técnicas y en otras muchas que ayudan a la explotación del espectro radioeléctrico, se ha querido únicamente mencionar alguna de ellas con el fin de dar importancia a la utilización eficiente de este limitado recurso.

1.2 Detección frecuencial o Spectrum Sensing

El término anglosajón con el que se refiere a este campo recibe el nombre de *spectrum sensing*, y adquiere una gran importancia cuando se quiere estudiar la distribución de las señales de radiofrecuencia dentro de un área específica. De esta manera se pueden tener numerosas emisiones en distintas frecuencias sin interferir unas con otras [3]. De entre las numerosas técnicas que existen actualmente para *spectrum sensing*, se encuentran aquellas que están basadas en medidas proporcionadas por un conjunto de sensores [4]. Por tanto, la precisión de las medidas proporcionadas por estos sensores es crucial para una correcta distribución frecuencial. Este proyecto está centrado en la estimación de campo recibido a partir de la información proporcionada por una serie de sensores, y es por ello que la utilidad de este trabajo se puede centrar en la ayuda a estas técnicas de *spectrum sensing* al proporcionar una estimación precisa de campo recibido.

La estimación de campo recibido a partir de sensores requiere en muchas ocasiones de una infraestructura en la cual los sensores están estratégicamente localizados a lo largo del área en la cual se quiere llevar a cabo una distribución espectral. Esta situación, además de poco realista, conlleva una inversión muy grande de dinero cuando estos sensores proporcionan medidas muy precisas ya que son dispositivos con precios muy elevados. Con el fin de encarar esta situación, apareció la técnica del *crowdsourcing* en la que los sensores usados son dispositivos los cuales proporcionan medidas menos exactas, pero cuyo precio decae considerablemente. Se pudo demostrar como a partir de la utilización esta técnica los resultados obtenidos son igualmente válidos provocando además una considerable reducción del dinero invertido [5]. Se trata de una técnica especialmente usada en campos como la localización interior gracias a comunicaciones vía WiFi [6].

El *spectrum sensing* no es la única técnica en la que las medidas proporcionadas por una serie de sensores (no necesariamente usando *crowdsourcing*) adquieren una gran importancia. Aplicaciones como el seguimiento de individuos o vehículos en interiores o exteriores [7, 8], el cálculo de distancias [9] o la navegación 3D a través de un interior [10], son sólo algunos de los campos en los que destaca la estimación de señal recibida.

1.3 Machine Learning

Para la estimación de campo recibido en un punto a partir de la información proporcionada por una serie de sensores se pueden usar técnicas de *machine learning* (ML). Las predicciones a partir de ML comenzaron a finales de los años cincuenta pero son especialmente importantes en la actualidad por su aportación a la sociedad en campos como la seguridad o los medios de comunicación entre otros, provocando de esta manera un incremento del bienestar en sociedades desarrolladas. Estas técnicas están basadas en modelos basados en datos, en los que el propio modelo *aprende* a partir de entradas y estima la salida basándose en la información proporcionada por las entradas.

La predicción a partir de algoritmos de ML permite obtener resultados más fiables respecto a los que una persona podría proporcionar. De acuerdo a algunos estudios [11], un ser humano necesitaría de 536 días para poder procesar lo equivalente a 1TB de memoria, y al final de esos días, no sería capaz de recordar ni la mitad de la información. Los sistemas de computación actuales están preparados para procesar esa cantidad de memoria en sólo algunos segundos, y además tienen la ventaja de que un mayor número de muestras implica una mayor precisión en las predicciones. No sólo en la capacidad de almacenamiento, sino en otros aspectos como la velocidad o la veracidad de las predicciones los ordenadores ofrecen mejores prestaciones que los seres humanos [12]. Por todo ello, los algoritmos de ML son altamente usados actualmente, gracias a la irrupción del *big bata*, entre otros factores. La capacidad de *aprender* gracias a grandes cantidades de datos se ha visto potenciada una vez las técnicas de *big data* eran capaces de proporcionar esta cantidad ingente de información necesaria para una estimación precisa. La gráfica mostrada en la Figura 1.1 muestra de una manera clara como la aparición del *big data* ha provocado un considerable aumento en el uso de algoritmos de ML. Es por ello que los algoritmos de ML están experimentando una "segunda juventud".

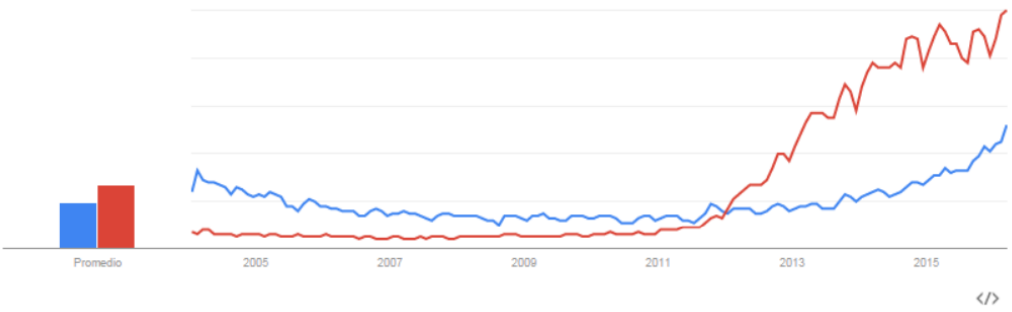

Figura 1.1 Crecimiento del ML (azul) y *Big Data* (rojo) en los últimos años [13].

Se podría escribir mucho acerca de la enorme variedad de algoritmos de ML que existen en la actualidad, y es por ello que una correcta decisión es crucial para conseguir el objetivo buscado de manera satisfactoria. En [14] se muestra una especie de diagrama de flujo que ayuda a la decisión del mejor algoritmo posible

basándose en parámetros como el número de muestras, gasto computacional requerido, precisión en la predicción...

En lo que resta de sección se van a presentar tanto los problemas que son capaces de resolver las técnicas de ML, como los algoritmos que son usados con mayor frecuencia en la actualidad.

1.3.1 Problemas

Entre los problemas destacan los descritos a continuación:

- **Regresión**: La dinámica de este tipo de situaciones es muy simple y está basada en encontrar una función *f*, obtenida a partir de una serie de entradas *x*, que proporciona una salida *y*, la cual puede ser un valor particular o un conjunto de valores. Es por ello que para resolver este tipo de problemas será necesario enfocarse en la búsqueda de medidas o cantidades específicas [15].
- **Clasificación**: Estos problemas aparecen constantemente en campos como el reconocimiento por voz o clasificación de documentación, entre otros, y su propósito final es muy fácil de entender ya que se trata de situaciones en las que es necesario separar y categorizar un *set* de datos en múltiples clases las cuales pueden ser etiquetadas [16].
- **Agrupación**: El fin último de este tipo de problemas es muy parecidos al de las situaciones de clasificación, pero con la diferencia de que en estos casos las agrupaciones o clases no están "pre-etiquetadas". Es por ello que será necesario hacer uso de técnicas que estén preparadas para buscar algún tipo de relación entre muestras pertenecientes a un *set* de datos. Están basadas en medidas previas y en patrones o reglas que deberían de repetirse por lo que son problemas a resolver en muchos campos. Entre otros, el *web mining* o la bio-informática [17].
- **Detección de anomalías**: En muchas ocasiones es necesario partir de un *set* de datos y encontrar aquellos que se diferencian significativamente del resto. Estos datos se suelen relacionar con eventos *especiales* y son problemas que aparecen regularmente en campos como el fraude fiscal o las negligencias médicas. La identificación de estos datos anómalos o extraños es una labor complicada ya que puede ser provocada por un evento *especial* o por causa del mismo modelo [18].

1.3.2 Técnicas

Las técnicas presentadas a continuación fueron desarrolladas con el fin de encontrar las soluciones a los problemas descritos anteriormente:

- **Procesos gaussianos**: Esta técnica, la cual será presentada a lo largo de esta memoria, forma la base para entender el algoritmo usado en este trabajo. En un proceso gaussiano, las muestras son generadas a partir de una función media y una función de covarianza y es una técnica usada fundamentalmente en regresión y clasificación.
- **Algoritmos basados en árbol de decisión**: Son técnicas que usan la matemática y la estadística para formar una estructura en forma de árbol que ayuda a la toma de decisiones. Existe un gran número de cuestiones que los desarrolladores de estos algoritmos tienen que responder antes de proceder a usarlos como son: ¿Cuál es el coste computacional de elegir un camino en el árbol u otro?¿Hasta cuando se puede seguir dividiendo los caminos?¿Sobre qué parámetros descansa la decisión final: tiempo de computación, riesgo...? [19].
- **Redes neuronales**: A pesar de que pueda parecer un término muy actual, la primera red neuronal artificial data del año 1959, la cual estaba basada en un modelo para eliminar los ecos producidos en una comunicación por teléfono [20]. En la actualidad, es una técnica usada en muchos campos del "día a día" de la sociedad como puede ser la economía o la medicina. Está basado en el funcionamiento del cerebro humano, en el que tanto las neuronas como sus conexiones son emuladas a partir de capas y enlaces, formando diagramas de grafos. La estructura diseñada es el aspecto clave en este tipo de algoritmos [21].
- **Algoritmos para la reducción de dimensión**: El decremento del número de dimensiones da lugar a una serie de ventajas como pueden ser la reducción del gasto computacional o la reducción de la memoria usada. Algunos algoritmos más complejos dentro de este campo son capaces de encontrar no solo información redundante, sino también aquellas muestras que son especialmente relevante dentro del *set* de datos [22].

- **Aprendizaje profundo**: Más conocido por su término anglosajón -*Deep Learning* (DL)-, siempre ha estado muy ligado al ML. Estos algoritmos están basados en las ya mencionadas redes neuronales y son usados actualmente en múltiples campos como puede ser el procesamiento digital de imágenes en medicina, reconocimiento facial... La vida en este siglo no se podría concebir sin la explosión de este fenómeno: El reconocimiento facial de los móviles actuales o el reconocimiento por voz que usan *Siri* o *Alexa* están basados en algoritmos de DL [23].
- ***Support Vector Machine* (SVM)**: Se trata de una técnica usada ampliamente en clasificación aunque también puede ser usada en regresión. Básicamente esta técnica pretende encontrar un hiper-plano que consiga clasificar las muestras proyectadas en un espacio de dimensión mayor[24].

1.4 Ubicación y organización del trabajo

Una vez se han introducido de una forma muy general todos los campos que de alguna forma rodean a este proyecto, se va a presentar la ubicación del trabajo dentro de estos campos así como la organización y estructura de esta memoria.

1.4.1 Ubicación

Como se detallará a lo largo de esta memoria, este trabajo está basado en la estimación de señal recibida a partir de la información proporcionada por sensores, la cual no es especialmente precisa, y es por ello que este trabajo se encuentra principalmente ubicado dentro del campo de la estimación espectral o en inglés *Spectral Stimation* (SS). Como se ha comentado anteriormente, el incremento de las comunicaciones inalámbricas ha provocado una creciente necesidad en los ingenieros de telecomunicaciones de estimar la utilización del espectro de la manera más exacta posible. Es por ello que este trabajo presenta una técnica para llevar a cabo dicha estimación a partir de sensores de bajo coste.

El segundo campo presentado anteriormente en el que se ubica este proyecto es el del *machine learning* ya que para la estimación de la señal recibida a partir de los sensores, se ha usado una técnica de ML. De entre los múltiples algoritmos presentados, este trabajo se encuentra dentro del primero, el cual se trata de algoritmos de ML para **regresión**. Mas específicamente, el método usado para la estimación de señal recibida está basado en procesos gausianos, o en inglés, *Gaussian orocesses for regression* (GPR).

Tanto el método de propagación usado, como el desarrollo matemático en el que está basado la estimación de señal a partir de GPR, son presentados a lo largo de esta memoria, cuya organización se presenta a continuación.

1.4.2 Organización del trabajo

Se aprovechará este apartado para presentar la estructura en la que está divida esta memoria con el fin de ubicar al lector y hacer más sencilla su lectura.

En el próximo capítulo se va a presentar la motivación y objetivo de este trabajo. Una vez finalizado este capítulo se pasará a presentar los fundamentos matemáticos en los que está basado la técnica de GPR. Una vez entendidos los conceptos básicos, se pasará a detallar el artículo en el que está basado este proyecto. Aspectos como el modelo de propagación o los errores y consideraciones que se tienen que tener en cuenta son presentados en este capítulo. Finalizada la presentación de la base matemática, se mostrará los entornos de programación usados, así como las particularidades del código desarrollado. La simulación y análisis de los resultados se presentarán en un capítulo posterior antes de finalizar con las conclusiones y los trabajos futuros.

2 Motivación y objetivos

Para mí, nunca ha habido una mayor fuente de honores terrenales
o distinción mayor que la conexión con los avances de la ciencia.

Isaac Newton

Una vez finalizada la introducción de la memoria, se va a presentar en este capítulo tanto las causas que han motivado al autor de esta memoria a llevar a cabo este proyecto así como los objetivos buscados durante la realización del mismo. Se considera importante introducir este capítulo en la memoria final ya que ayuda a entender el por qué de la realización de este trabajo.

2.1 Motivación

Esta sección, junto con algunas partes del capítulo de conclusiones, son las dos únicas situaciones en las que voy a dar mi opinión o punto de vista, y es por lo que en ocasiones se puede producir un cambio en los tiempos verbales de la redacción, los cuales pueden pasar a la primera persona del singular.

Entrando en contenido, la principal causa que motivó la realización de este proyecto por mi parte reside en el desconocimiento que tenía hasta el momento de la elaboración del mismo, de las técnicas y algoritmos de *machine learning* que existen en la actualidad. Consideraba, que después de seis años de estudio, había adquirido conocimientos en muchos campos importantes pertenecientes a las telecomunicaciones, pero no había entrado nunca en este amplio sector. Además, la imposibilidad de cursar la única asignatura dedicada exclusivamente al ML en el máster, hizo que contactara con el tutor de este proyecto con el fin de poder realizar un trabajo el cual me acercase al mundo del ML.

Por mi parte, era consciente de la importancia que tienen estas técnicas en la actualidad, no sólo en el ámbito de las telecomunicaciones, sino en otros muchos aspectos de la sociedad actual. Es por ello que no quería terminar mi etapa de formación sin tener un acercamiento a este campo, ya que lo considero parte de la formación básica en una titulación de ingeniería de telecomunicaciones. En este punto me gustaría agradecer de nuevo al tutor de este proyecto, Juan José Murillo Fuentes, por brindarme la oportunidad de trabajar en este proyecto y por la ayuda proporcionado durante la realización del mismo.

Por otro lado, si se lleva a cabo un análisis de las referencias usadas para la realización del capítulo de introducción, se puede observar como todas ellas están basadas en **artículos realizados entre 2017 y 2020**, lo que muestra la gran importancia que tiene el ML en la actualidad. Un estudio llevado a cabo por la prestigiosa revista *forbes* [25], muestra de manera práctica y a través de gráficos la importancia que tiene el ML en los distintos ámbitos de la sociedad actual: economía, ingeniería, *marketing*... En las Figuras 2.1 y 2.2 se muestran dos de los gráficos presentados en dicho estudio donde se aprecia de manera clara la **dependencia** que ya se tiene en muchos ámbitos de las técnicas de ML. Me gustaría destacar sobre todo la Figura 2.1, en la que se muestra como los algoritmos de ML son especialmente cruciales en las telecomunicaciones actualmente, lo que viene a reforzar mi propuesta de introducir el *machine learning* en la formación básica de un ingeniero de telecomunicaciones.

Soy por tanto consciente de las ventajas competitivas que supone tener conocimientos dentro de este sector y cómo esta formación, aunque sea de manera introductoria, me puede abrir numerosas oportunidades a lo largo de mi carrera profesional. **Ésta ha sido por tanto mi mayor motivación**.

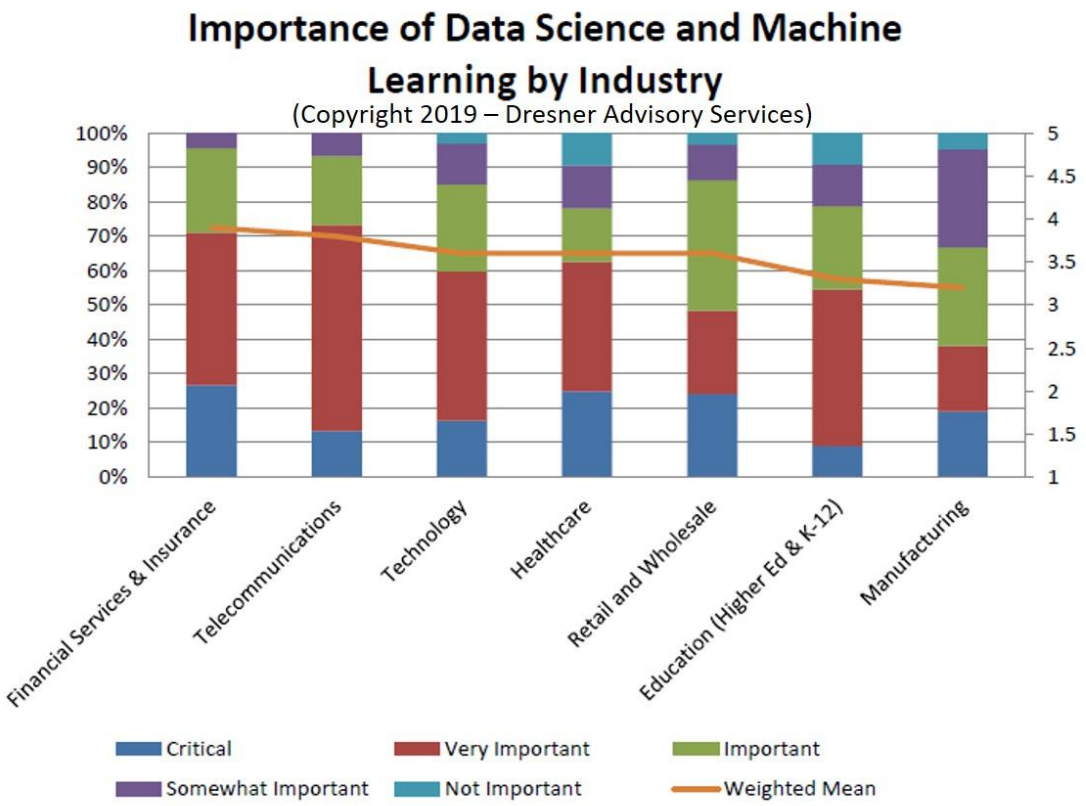

Figura 2.1 Importancia por sector industrial [25].

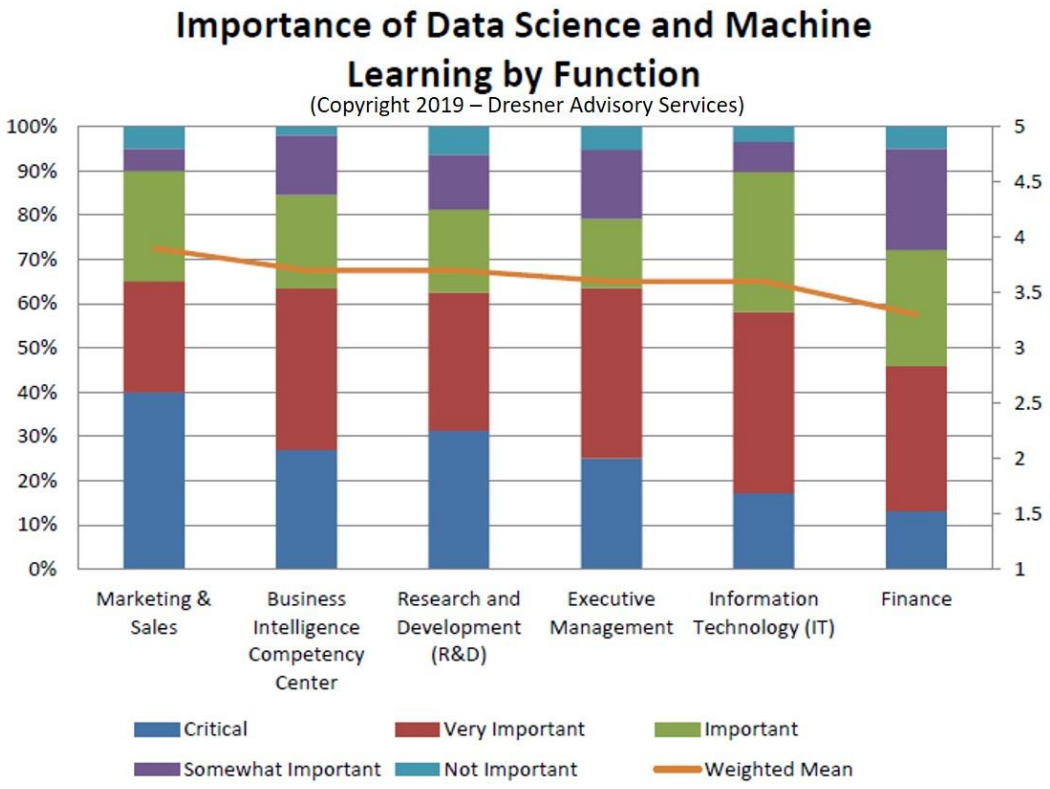

Figura 2.2 Importancia por función [25].

2.2 Objetivos

A raíz de lo comentado anteriormente, queda claro que el objetivo buscado a la hora de realizar este proyecto es principalmente llevar a cabo un acercamiento a una de las técnicas de ML más usadas en telecomunicaciones. Como también se ha mencionado, la base desde la que se partía era nula, por lo que cualquier aprendizaje que se haya podido llevar a cabo a lo largo de la realización de este proyecto es positivo.

Debido principalmente a la falta de *background* dentro de este campo, la complejidad del trabajo no es especialmente elevada a ojos de un experto en ML, pero para alguien que está empezando su andadura en estas técnicas, supone un reto el cual se revisará si ha sido superado satisfactoriamente o no en el apartado de conclusiones.

Dejando a un lado los objetivos a nivel personal, el fin último de este trabajo a nivel técnico se puede resumir en algunas líneas. Este proyecto está basado en un estudio realizado previamente por investigadores de la Universidad de Sevilla [34] (más adelante en la memoria se dedica un capítulo a presentar dicho estudio), el cual usaba el lenguaje de programación *MATLAB* para llevar a cabo la implementación de las ecuaciones matemáticas a nivel de simulación. El objetivo planteado por el tutor de este proyecto fue por tanto **convertir** el código desarrollado a un lenguaje de programación de *software* libre como puede ser *Python*, y usar librerías y entornos de libre acceso. Personalmente, y como defensor de los medios que ayudan a compartir los conocimientos *software* de manera gratuita, me pareció una gran idea. Las particularidades del código desarrollado serán presentadas en algún capítulo posterior en esta memoria, así como la introducción a las librerías que han sido utilizadas y que incluyen los algoritmos de GPR. Además, es bien sabido que actualmente el lenguaje de programación *Python* se encuentra en la lista de los más usados en mucho de los campos principales de la ingeniería, pero específicamente se trata de un lenguaje muy usado en aplicaciones de ML. Por tanto, no se concibe un aprendizaje sobre ML sin la adquisición al mismo tiempo de los conceptos de *Python* aplicados a ML. Por último, se propone como mejora al estudio original la utilización de técnicas para reducir la complejidad y el tiempo de ejecución de las simulaciones.

2.2.1 Problema a resolver

Ya con los objetivos del trabajo bien definidos cabe preguntarse, ¿en qué consiste ese estudio realizado previamente?, el código convertido a *Python*, ¿qué tipo de acciones realiza?. Básicamente, y como ya se ha comentado de pasada en un capítulo anterior, en este trabajo se pretender realizar una estimación de campo recibido en distintos puntos o nodos de un cierto *grid*, a partir de la información proporcionada por sensores de bajo coste. Estos sensores pueden ser terminales de usuario u otro tipo de dispositivo electrónico que requiera de comunicación inalámbrica. Además, tanto la posición del transmisor como de los sensores es desconocida, esto es, será necesario llevar a cabo un proceso previo en el que se estimen estas posiciones.

Las ecuaciones matemáticas deducidas para llevar a cabo la estimación del campo recibido usando un proceso gaussiano para regresión (GPR) son presentadas a lo largo de esta memoria, así como su implementación a nivel de código y las simulaciones y resultados obtenidos.

3 Procesos Gaussianos para Regresión

Después de todo, ¿qué es un científico entonces? Es un Hombre curioso que mira a través del ojo de una cerradura, la cerradura de la naturaleza, tratando de saber qué es lo que sucede.

JACQUES YVES COUSTEAU

En este capítulo, y una vez se tiene una idea general sobre el proyecto gracias a los capítulos anteriores, se va a presentar el algoritmo de ML presente en este trabajo. Debido a su contracción anglosajona, a lo largo de este capítulo y en los posteriores se va a usar la notación GPR para referirse a la técnica de *Gaussian processes for regression*.

Como ya se ha comentado en secciones anteriores, existen numerosos algoritmos de ML que son usados actualmente en el ámbito de las telecomunicaciones, de entre los cuales destacan aquellos que son dedicados a llevar a cabo una regresión de las muestras. Este capítulo comenzará presentando los fundamentos de los procesos gaussianos con el fin de introducir al lector en este campo antes de pasar a mencionar el algoritmo usado en este proyecto. Este capítulo sigue teniendo un carácter general e introductorio, y no será hasta los siguientes capítulos donde se mencionará las implicaciones que tiene este algoritmo en el proyecto y en los resultados finales obtenidos, así como las características *únicas* y *especiales* que tiene esta técnica dentro del proyecto y las consideraciones a tener en cuenta.

3.1 Procesos Gaussianos

De entre los múltiples bloques de algoritmos diferentes de ML que existen en la actualidad, los procesos gaussianos son especialmente usados en regresión, clasificación y reducción de la dimensión. Se trata de una técnica que necesita de un coste computacional muy elevado como será mencionado posteriormente en este capítulo, y por tanto, no fue hasta entrados el sigo XXI cuando se comenzó a implementar estas técnicas en ML, a pesar de que fue ya estudiada anteriormente [26].

Un proceso gaussiano se puede definir como una **generalización** de la distribución de probabilidad gaussiana, ya que puede verse como una colección de variables aleatorias, en las que cada subconjunto de ellas tiene una distribución gaussiana. Se puede concluir por tanto que, cuando se tiene una *variable aleatoria gaussiana*, las muestras obtenidas -que pueden ser un simple escalar o un vector- siguen una distribución gaussiana, mientras que cuando se parte de un *proceso gaussiano*, lo que se obtiene son **funciones** que siguen una distribución gaussiana [27]. Es por ello que son especialmente útiles en regresión ya que se asume que cada muestra proviene de la combinación de un conjunto de escalares obtenidos a partir de la misma función gaussiana.

Siguiendo una metodología que ayude a la comprensión de esta técnica, si para definir una distribución gaussiana es necesario un vector de medias y una matriz de covarianza, para definir un proceso gaussiano es necesario una *función de medias* y una *función de covarianzas* (que como se verá mas adelante recibe el nombre de *kernel*).

$$f(x) \sim GP(m(x), k(x,x')) \tag{3.1}$$

Donde:

$$m(x) = \mathbb{E}[f(x)] \tag{3.2}$$

$$k(x,x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] = Cov(x,x') \tag{3.3}$$

Siguiendo el desarrollo matemático presentado en [28], se va a estudiar las restricciones que deben de tener las mencionadas funciones de media y covarianza. Partiendo de un conjunto de puntos $x_{n=1}^{N}$ pertenecientes a $\mathbb{R}^D$ y apoyándose de la propia definición de proceso gaussiano, se puede concluir que:

$$(f(x_1), f(x_2), ..., f(x_N))^T \sim \mathcal{N}(m,K) \tag{3.4}$$

con $K_{nm} = k(x_n,x_m)$ y $m_n = m(x_n)$.

Se puede ver de manera clara como K, al ser una matriz de covarianzas, tiene que ser simétrica y semidefinida positiva. Por tanto, la función de covarianzas $k(x,x')$ debe de garantizar que se obtienen matrices simétricas y semidefindas positivas, y es por ello que recibe el nombre de función del núcleo o ***kernel***, cuyo término será usado con mayor frecuencia a partir de este momento. Para la función de media no se tiene ningún tipo de restricción, y únicamente se aconseja habitualmente asignarle un valor cero con el fin de facilitar los desarrollos matemáticos.

3.2 Procesos Gaussianos para Regresión (GPR)

Los conceptos básicos sobre GP introducidos en la sección anterior sirven como fundamentos para explicar el algoritmo de ML usado en este proyecto para realizar la estimación de señal a partir de información proporcionada por sensores. Este algoritmo está basado en el uso de los procesos gaussianos presentados en el anterior punto para de esta manera llevar a cabo una regresión a partir de las muestras de entrenamiento. Los modelos que usan los procesos gaussianos para regresión pueden verse desde dos puntos de vista diferentes como se muestra en [29], pero a lo largo de esta memoria van a estudiarse únicamente desde uno de ellos, el cual es descrito a continuación.

3.2.1 Punto de vista de los pesos

Este punto de vista dentro de GPR está basado en el ampliamente usado modelo en el que las salidas son obtenidas a partir de una combinación lineal de las entradas. Son modelos simples de entender e implementar y que además tienen una carga computacional muy baja, pero que no proporcionan resultados fiables cuando la relación entras las muestras de salida y de entrada tienen una relación no lineal.

Se partirá del modelo más básico de regresión lineal en que las salidas se calculan a partir de entradas x (de dimensiones $D \times 1$) las cuales son ponderadas usando ciertos parámetros w, cuyas dimensiones son exactamente iguales al vector de entrada, y a las que se le añade un ruido cuyas muestras siguen una distribución gaussiana de media cero y varianza σ^2:

$$y = x^T w + \mathcal{N}\left(0, \sigma^2\right) \tag{3.5}$$

A partir de este modelo se puede calcular de manera muy simple la función densidad de probabilidad de las salidas dado los parámetros, w, teniendo en cuenta que cada muestra de entrenamiento es independiente una de otra:

$$p(\mathbf{y}|X,\mathbf{w}) = \prod_{i=1}^{n} p(y_i|\mathbf{x_i},\mathbf{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(y_i - \mathbf{x_i}^T\mathbf{w})^2}{2\sigma^2}) = \mathcal{N}\left(X^T\mathbf{w}, \sigma^2 I\right) \tag{3.6}$$

Siendo las columnas de X las dimensiones de cada una de las muestras D), y las filas el número total de muestras de entrada (M). Además, será necesario especificar en este tipo de modelos una probabilidad *a priori* sobre los parámetros:

$$\mathbf{w} \sim \mathcal{N}\left(\mathbf{0}, \Sigma p\right) \tag{3.7}$$

En los modelos lineales, la inferencia está basada en la probabilidad *a posteriori* sobre dichos parámetros. Por tanto, y a partir del teorema de Bayes:

$$p(\mathbf{w}|\mathbf{y},X) = \frac{p(\mathbf{y}|X,\mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|X)} \tag{3.8}$$

Además, $p(\mathbf{y}|X)$ se define como:

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|X,w)p(\mathbf{w})d\mathbf{w} \tag{3.9}$$

Estos resultados forman la base para poder obtener la distribución de predicción (ver [29] para más detalles):

$$p(f_*|\mathbf{x}_*,X,\mathbf{y}) = \int p(f_*|\mathbf{x}_*,\mathbf{w})p(\mathbf{w}|X,\mathbf{y})d\mathbf{w} = \mathcal{N}\left(\frac{1}{\sigma^2}\mathbf{x}_*^T A^{-1} X\mathbf{y}, \mathbf{x}_*^T A^{-1}\mathbf{x}_*\right) \tag{3.10}$$

Donde:

$$A = \frac{1}{\sigma^2}XX^T + \Sigma_p^{-1} \tag{3.11}$$

Por tanto y la vista de las Ecuaciones (3.10) y (3.11) se puede sacar las siguientes conclusiones:

1. La distribución de predicción es también gaussiana.
2. La varianza tiene una dependencia cuadrática dada por el producto de las muestras de entrada y A, lo que demuestra que la incertidumbre en la predicción de las salidas crece a la vez que las dimensiones de las muestras de entrada, como era de esperar en un modelo lineal.

Como se ha comentado previamente, las predicciones calculadas por este modelo tienen una precisión muy escasa cuando las relaciones entre las muestras de entrada y salida no siguen una función lineal. Por tanto, y para mejorar este comportamiento, se propone proyectar las entradas en un espacio de gran dimensión cuyas bases son **funciones**, y aplicar directamente una regresión lineal en este espacio el cual recibe el nombre de *feature space*. Esto es, **se pretende realizar una regresión lineal de una transformación no lineal de la entrada**. La presentación de esta idea es muy importante ya que ayuda a entender la procedencia de uno de los aspectos claves de este trabajo, la función *kernel*. De esta manera, se define la función $\phi(x)$ la cual transforma un vector de dimensión D, en otro de dimensión N, donde $N > D$. Por tanto, y volviendo al desarrollo matemático presentado en esta sección:

$$f(\mathbf{x}) = \phi(\mathbf{x})^T\mathbf{w} \tag{3.12}$$

Es importante tener en cuenta en este punto de la demostración que las dimensiones de w son $N \times 1$ en lugar de $D \times 1$. Procediendo de la misma forma se llega a la siguiente distribución de predicción:

$$f_*|\mathbf{x}_*,X,\mathbf{y} \sim \mathcal{N}\left(\frac{1}{\sigma^2}\phi(\mathbf{x}_*)^T A^{-1}\phi\mathbf{y}, \phi(\mathbf{x}_*)^T A^{-1}\mathbf{x}_*\right) \tag{3.13}$$

Donde:

$$\phi = \phi(X) \tag{3.14}$$

Siendo las dimensiones de X ahora $N \times M$ en lugar de $D \times M$. Por último:

$$A = \frac{1}{\sigma^2}\phi\phi^T + \Sigma_p^{-1} \tag{3.15}$$

Se puede apreciar claramente como las Ecuaciones (3.13) y (3.10) son exactamente iguales y se llega por tanto a la misma solución, cambiando únicamente x por $\phi(x)$ y x_* por $\phi(x_*)$.

Para presentar la función *kernel*, se va hacer uso del teorema de inversión de matrices de Woodbury con el fin de obtener la siguiente distribución de predicción a partir de la Ecuación (3.13).

$$f_*|\mathbf{x}_*,X,\mathbf{y} \sim \mathcal{N}\left(\phi_*^T\Sigma_p\phi(K+\sigma^2 I)^{-1}\mathbf{y}, \phi_*^T\Sigma_p\phi_* - \phi_*^T\Sigma_p\phi(K+\sigma^2 I)^{-1}\phi^T\Sigma_p\phi_*\right) \tag{3.16}$$

Donde:

$$\phi(x_*) = \phi_* \tag{3.17}$$

Y

$$K = \phi^T \Sigma_p \phi \tag{3.18}$$

La Ecuación (3.16) es esencialmente la misma que (3.10) pero en esta nueva expresión se observa como las matrices a las que es necesario calcular su inversa cambian. En la Ecuación (3.10), las dimensiones de la matriz a invertir están definidas por las dimensiones de la entrada tras realizar la transformación no lineal, N, mientras que en la Ecuación (3.16) están definidas por el número de muestras, M. Si se tiene en cuenta que en algunas ocasiones las dimensiones de la entrada transformada es muy alta (incluso infinito en ocasiones), esta nueva expresión, obtenida a partir del llamado *kernel trick*, es esencial para estimar a partir de una solución viable.

Se puede observar en la Ecuación (3.16) como la expresión $\phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}')$ es igualmente válida y tiene en cuenta las muestras de entrenamiento, x, y las muestras de test, x'. Por tanto, la función *kernel* o *función de covarianza* se puede definir como:

$$k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}') \tag{3.19}$$

Obteniendo de esta manera las relaciones que presentan las muestras de entrenamiento con las muestras de test.

3.2.2 Ejemplo básico

Con el fin de aclarar más todavía este punto se va a presentar un ejemplo de código [30] donde se aplican los conceptos presentados a lo largo de esta sección de una manera práctica. Aunque en este proyecto se usan librerías para acceder a los algoritmos de GPR, se considera necesario presentar este ejemplo en este punto de la memoria para empezar a mostrar aplicaciones prácticas sencillas de estas técnicas, y de esta manera facilitar la comprensión de aplicaciones más complejas como pueden ser la de este trabajo.

Como se ha ido explicando, los procesos gaussianos son usados para definir una probabilidad *a priori* sobre las muestras de entrada la cual es convertida en una probabilidad *a posteriori* usando la información proporcionada por las muestras de entrenamiento.

Entrando de una manera más exacta en el cálculo de la probabilidad *a posteriori* a partir de la probabilidad *a priori* obtenida de las observaciones, se define en primer lugar una función densidad de **probabilidad conjunta** entre: (1) los datos observados (muestras de entrenamiento) y (2) las salidas de la función (muestras de test):

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right) \tag{3.20}$$

Como era de esperar al usar procesos gaussianos, la función densidad de probabilidad conjunta es una gaussiana de media cero cuya matriz de covarianza se divide en distintos bloques calculados a partir del *kernel*. El término $K(X,X)$ se obtiene al aplicar la función *kernel* a las muestras de entrenamiento y muestra las relaciones que tienen éstas observaciones con ellas mismas. De igual forma se calcularía los términos $K(X,X_*)$ y $K(X_*,X_*)$ aplicando la función *kernel* a las muestras de entrenamiento o a las muestras de *test* según corresponda, proporcionando información sobre las similitudes entre muestras.

De esta forma, y para obtener el calculo de la probabilidad *a posteriori*, será necesario *eliminar* aquellas funciones que no concuerden con las observaciones. Para ello, habrá que obtener la función densidad de probabilidad **condicionada**, $p(f_*|f)$, a partir de la probabilidad conjunta, $p(f_*,f)$. Se puede demostrar [29] que se llega a una expresión como la siguiente:

$$f_*|X_*,X,f \sim \mathcal{N}\left(K(X_*,X)K(X,X)^{-1}f, K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*)\right) = \mathcal{N}(\mu_*,\Sigma_*) \tag{3.21}$$

Esta expresión vuelve a ser equivalente a (3.10) y (3.16) (con y igual a f) pero usa una notación más propia de los procesos gaussianos para regresión y será, por tanto, la que se use a lo largo de la memoria.

Finalmente, las funciones muestras pueden ser generadas usando las matrices media y varianza de la Ecuación (3.21). La generación de muestras se puede llevar a cabo de manera trivial en el caso de ser una

distribución normal con una sola variable. Para el caso de disponer de múltiples variables las muestras tienen que ser generadas como sigue (para un caso general), donde

$$f_*|X_*,X,f \sim \mu_* + B\mathcal{N}(0,I) \tag{3.22}$$

Donde es necesario usar la descomposición de Cholesky para encontrar una matriz B tal que: $BB^T = \Sigma_*$.

Para el ejemplo se ha considerado como *kernel* una función exponencial cuadrática que también puede recibir el nombre de función *kernel* gaussiana:

$$k(x_p,x_q) = exp(-\frac{1}{2}|x_p - x_q|^2) \tag{3.23}$$

Se trata de uno de los *kernels* más usados en GPR y proporciona valores de covarianza cercanos a uno cuando las muestras de entrada están muy próximas. De igual forma, el valor de la covarianza decrece a medida que aumenta la distancia entre muestras.

En primer lugar, se mostrará las muestras generadas a partir de la función *kernel*, los valores de entrada y la Ecuación (3.22). Por tanto, para generar las muestras se sigue:

$$f \sim \mathcal{N}(0,K(X_*,X_*)) \tag{3.24}$$

Este procedimiento se observa en el siguiente fragmento de código:

Código 3.1 Muestras de las funciones a priori.

```
# Cálculo de la matriz de covarianza
K_ss = kernel(Xtest, Xtest, param)
# Obtiene la descomposición de Cholescky
L = np.linalg.cholesky(K_ss + 1e-15*np.eye(n))
# Uso de la fórmula: f_{*} \sim \mu + BN(0,I) para genera las muestras
f_prior = np.dot(L, np.random.normal(size=(n,3)))
```

Donde se aprecia de manera clara, el calculo de la matriz de covarianza, y la obtención de muestras a partir de la descomposición de Cholesky. Una vez se tienen las muestras, la representación es trivial y da lugar a la Figura 3.1

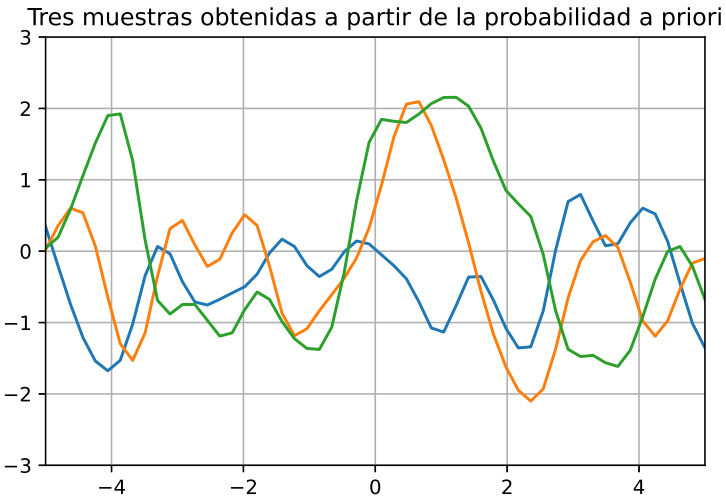

Figura 3.1 Obtención de muestras para función a priori (Ecuación (3.24)).

La Figura 3.1 representa la probabilidad *a priori* determinada por la matriz de covarianza calculada a partir del *kernel* donde se especifica cómo se parece la salida de dos entradas para una distancia dada. Por tanto, las muestras contiguas de las funciones representadas se parecen según se indique en el *kernel*.

La función a aprender en este ejemplo es un seno, por tanto se va a generar algunas muestras de entrenamiento con el fin de *forzar* a la salida de las funciones a pasar por estas muestras. Únicamente implementando en el código las fórmulas especificadas en esta sección, especialmente las Ecuaciones (3.21) y (3.22), se obtienen unos resultados que se muestran en la Figura 3.2 donde se incluyen tres posibles salidas o predicciones. De esta forma, en dicha imagen se aprecia de manera clara como las tres funciones obtenidas *a posteriori* (amarillo, azul y verden) pasan por las muestras de entrenamiento (marcadas como un cuadrado azul) obtenidas a partir de la función original (negro a rayas) incrementando la precisión de la estimación en esos puntos, y en los cercanos a estos. En los intervalos intermedios, donde no hay muestras de entrenamiento, el algoritmo no sabe cómo proceder por lo que la salida es directamente la función *a priori*. Un aspecto interesante se aprecia en la media de la predicción (roja a rayas), la cual es cero en los tramos donde no hay muestras de entrenamiento ya que se especificó en las consideraciones *a priori* (Ecuación (3.24)) que la información sobre la media era nula.

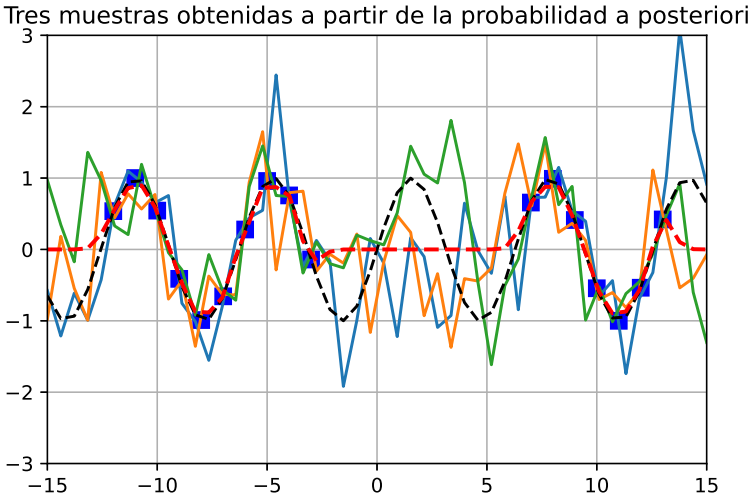

Figura 3.2 Obtención de muestras para función *a posteriori* (Ecuación (3.22)).

3.3 Consideraciones adicionales sobre GPR

Todo lo comentado a lo largo de la secciones anteriores corresponde únicamente con una breve introducción acerca de los procesos gaussianos aplicados a la regresión. Se podría invertir un tiempo mucho mayor en presentar más detalles desde el punto de vista matemático o estadístico, pero se considera que está fuera de los límites correspondientes al alcance de este trabajo. Sin embargo, algunas consideraciones especiales sobre GPR se detallan a lo largo de esta sección, ya que se trata de conceptos que son convenientes de estudiar debido a su aparición en el desarrollo de este proyecto.

3.3.1 Muestras con ruido

En una situación real, las muestras de entrenamiento no son salidas de una función objetivo como tal, sino que se encuentran influenciadas por un ruido que modifica su valor original. Con el fin de mostrar el caso más simple, se considera un ruido gaussiano cuyas muestras son independientes e idénticamente distribuidas. Por tanto, la probabilidad *a priori* de las muestras viene marcada por una covarianza:

$$cov(y_p, y_q) = k(x_p, x_q) + \sigma^2 \delta_{pq} \tag{3.25}$$

Teniendo en cuenta que δ_{pq} es una delta de Kronecker, el ruido en las muestras provoca la aparición de una matriz diagonal la cual modela este ruido. Por tanto, la función densidad de probabilidad conjunta quedaría como sigue:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) + \sigma^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right) \tag{3.26}$$

Procediendo de igual forma que para el caso de las muestras sin ruido, la función densidad de probabilidad condicionada obtenida a partir de la conjunta sería:

$$f_*|X_*,X,f \sim \mathcal{N}(\mu_*,\Sigma_*) \tag{3.27}$$

donde:

$$\mu_* = K(X_*,X)[K(X,X)+\sigma^2 I]^{-1} f \tag{3.28}$$

$$\Sigma_* = K(X_*,X_*) - K(X_*,X)[K(X,X)+\sigma^2 I]^{-1} K(X,X_*) \tag{3.29}$$

Aunque parece un cálculo muy sencillo, tiene numerosas implicaciones que pueden verse en [29]. De una manera práctica, y con el fin de modelar las muestras con ruido, únicamente hay que añadir la potencia, σ^2, de ruido a la predicción de la varianza.

3.3.2 Hiperparametrización

Si se analiza de manera exhaustiva el código 3.1 (lo cual no es un trabajo muy exigente debido a la simplicidad del mismo), se puede observar como uno de los argumentos que aparecen en la llamada a la función *kernel* en la línea 2 no ha sido estudiado. Se trata del argumento *param* y no es más que uno de los **hiperparámetros** del *kernel*.

El *kernel* de un proceso gaussiano puede verse como un parámetros fijo, pero también como una función variable cuyos hiperparámetros ayudan a encontrar el *mejor kernel* de acuerdo a las muestras o escenario en el que se aplique. En el ejemplo más básico del *kernel* RBF, la Ecuación (3.23) una vez se han añadido los hiperparámetros queda como sigue:

$$k(x,x') = \sigma_f^2 \exp(-\frac{1}{2l^2}|x-x'|^2) \tag{3.30}$$

Se observa, que tanto σ_f y l son considerados como hiperparámetros además del propio ruido, σ, que puede ser tomando igualmente como hiperparámetro.

Se verá en el capítulo correspondiente como estos hiperparámetros serán optimizados con el fin de encontrar el *kernel* que mejor funcione en este proyecto basándose en la información proporcionada por las muestras. La discusión en torno a la elección del *kernel* es uno de los aspectos fundamentales cuando se trabaja con procesos gaussianos y es por ello que la decisión final deberá estar justificada.

En uno de los capítulos posteriores se nombrará algunas características adicionales sobre los hiperparámetros y especialmente se considerará el parámetro l el cual recibe el nombre de factor de escala o *lengthscale*.

3.3.3 Media distinto de cero

Hasta ahora y por simplicidad en el desarrollo matemático se ha considerado que la función media del proceso gaussiano es cero. Teniendo en cuenta que esta suposición no concuerda con el modelo estudiado en este proyecto (el cual será presentado en un capítulo posterior) será necesario aclarar los conceptos adicionales que aparecen cuando se trabaja con un proceso gaussiano de media distinta de cero.

Teniendo en cuenta una función media distinta de cero, la densidad de probabilidad conjunta queda de la siguiente forma:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(\hat{X}) \end{bmatrix}, \begin{bmatrix} K(X,X)+\sigma^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right) \tag{3.31}$$

De igual forma que para el caso anterior se puede desarrollar dicha probabilidad conjunta y alcanzar una función densidad de probabilidad condicionada, la cual es una distribución gaussiana cuya media es en este caso:

$$\bar{\mathbf{f}}_* = m(\hat{X}) + K(\hat{X},X)(K(X,X)+\sigma^2)^{-1}(\mathbf{f}-m(X)) \tag{3.32}$$

Teniendo en cuenta que el término $m(X)$ se desconoce, será necesario estimarlo a partir de los datos. Se parte por tanto de la siguiente expresión:

$$f(x) = g(x) + \mathbf{h}(x)^T \beta + \sigma^2 I \tag{3.33}$$

Donde $g(x)$ es un proceso gaussiano de media cero como cualquiera de los vistos a lo largo de este capítulo, $\mathbf{h}(x)$ es un conjunto fijo de bases y β son parámetros adicionales. Esta visión hace que β se optimice a la misma vez que los hiperparámetros del *kernel* [28]. Otra posible opción se aprecia en [29], donde $\beta \sim \mathcal{N}(\mathbf{b}, B)$, obteniéndose de esta forma un nuevo proceso gaussiano:

$$f(x) \sim GP(\mathbf{h}(x)^T\mathbf{b}, k(x,x') + \mathbf{h}(x)^T B\mathbf{h}(x')) \tag{3.34}$$

Al igual que para el resto de procesos gaussianos vistos en este capítulo, se puede difinir una función densidad de probabilidad conjunta la cual da lugar a una probabilidad condicionada con una media y una matriz de covarianza. Se puede acceder a la referencias para ver los resultados finales así como el desarrollo matemático, los cuales no son plasmados en esta memoria ya que se considera que están fuera del alcance de este trabajo. Lo más importante a destacar en este punto es que se genera un nuevo proceso gaussiano el cual se puede expresar a partir de la media y covarianza de un proceso gaussiano de media cero, a los que se suma una serie de términos que aparecen cuando se supone que la media del proceso gaussiano es distinta de cero.

En un caso práctico, la media del proceso gaussiano es cero cuando no hay ningún conocimiento *a priori*, pero distinta de cero cuando se conoce algo sobre la media. Por ejemplo, si se sabe *a priori* que la media es una recta con una pendiente determinada, se incluiría esta información como media del GPR dejando el valor final de la pendiente como hiperparámetro.

3.3.4 Estimación Recursiva

En muchas aplicaciones, y al igual que ocurre en este proyecto, podría darse el caso en el que el escenario no fuera fijo, y por tanto, variase a lo largo del tiempo, siendo necesario calcular una nueva función densidad de probabilidad condicionada cada vez que se recibe una nueva muestra.

Para afrontar este punto y los que vienen a partir de ahora, será necesario hacer un cambio de mentalidad, y ver las fórmulas de media y matriz de covarianza obtenidas en la Ecuación (3.27) a nivel de muestra. Para ello se va a realizar el siguiente cambio de notación:

$$\begin{aligned} K(X,X) + \sigma_n^2 I \equiv C_n &= K_n + \sigma_n^2 I_n \\ \mu_* \equiv \mu_n &= K_n C_n^{-1} y_n \\ \Sigma_* \equiv \Sigma_n &= K_n - K_n C_n^{-1} K_n \end{aligned} \tag{3.35}$$

Además, a partir de ahora se tendrá en cuenta un modelo para la ecuación de regresión con la siguiente notación: $y = f(x) + \sigma$. Las muestras de entrenamiento serían por tanto cada y_n mientras que las muestras de test serán cada x_n.

De esta manera, cada vez que se recibe una nueva muestra de entrenamiento y_{n+1} y una nueva muestra de test x_{n+1}, los valores de media y matriz de covarianza se pueden actualizar como sigue [26]:

$$\mu_{n+1} = \begin{bmatrix} \mu_n \\ \mu_{f(x_{n+1})} \end{bmatrix} - \frac{\mu_{f(x_{n+1})} - y_{n+1}}{\sigma^2_{y_{n+1}}} \begin{bmatrix} h_{n+1} \\ \sigma^2_{f(x_{n+1})} \end{bmatrix} \tag{3.36}$$

$$\Sigma_{n+1} = \begin{bmatrix} \Sigma_n & h_{n+1} \\ h_{n+1}^T & \sigma_{f(x_{n+1})} \end{bmatrix} - \frac{1}{\sigma^2_{y_{n+1}}} \begin{bmatrix} h_{n+1} \\ \sigma^2_{f(x_{n+1})} \end{bmatrix} \begin{bmatrix} h_{n+1}^T & \sigma^2_{f(x_{n+1})} \end{bmatrix} \tag{3.37}$$

Donde $h_{n+1} = \Sigma_n K_n^{-1} k_{n+1}$.

Por tanto, y a diferencia de lo que se pudiera pensar en primera instancia, no es necesario calcular en cada iteración una nueva función densidad de probabilidad, sino que únicamente habrá que actualizar los valores de media y matriz de covarianza, reduciendo de manera muy considerable la carga computacional. Además, no es necesario almacenar las dos matrices sino que únicamente bastará con actualizar una de ellas y calcular la otra a partir de la primera.

3.3.5 Factor de olvido o *Forgetting factor*

Se trata de un concepto muy importante cuando las muestras varían con el tiempo y por tanto se tiene que dar una mayor importancia a las muestras recientes. De esta manera, y gracias a este parámetro, se consigue

olvidar la información contenida en las muestras más antiguas. Para ello, y después de actualizar las matrices en cada iteración, se realiza el siguiente paso:

$$\begin{aligned} \mu &\leftarrow \sqrt{\lambda}\mu \\ \Sigma &\leftarrow \lambda\Sigma + (1-\lambda)\Sigma \end{aligned} \tag{3.38}$$

El factor de olvido λ puede tomar valores comprendidos entre cero y uno ($0 < \lambda \leq 1$), no teniendo ningún efecto cuando vale uno ya que como se aprecia en la Ecuación (3.38), los nuevos valores de media y matriz de covarianza se estarían calculando a partir de la muestra actual. Por tanto, valores cercanos a uno darán más importancia a las muestras actuales que a las antiguas, mientras que conforme el valor se acerca más a cero, esta importancia va decayendo. De esta manera se lleva a cabo una predicción en la cual se tiene en cuenta tanto la muestra actual como las pasadas, pero cuya importancia en la predicción está ponderada por un parámetros λ.

Para una mayor información se puede acceder a la bibliografía y especialmente a [26] donde se demuestra como este paso en realidad no es visto como un paso adicional en la predicción sino como parte de un procedimiento más completo en el que el *kernel* tiene una visión espacio-temporal.

3.4 *Sparse GP*. Una mejora del tiempo de ejecución

Una desventaja común a todas las técnicas basadas en ML está relacionada con los tiempos de ejecución de los algoritmos implementados. En la mayoría de los escenarios de aplicación de estos algoritmos, los tiempos de ejecución rondan valores bastantes altos siendo estos en algunos casos del orden de días o semanas, incluso después de hacer uso del *kernel trick*. En este proyecto, esta desventaja aparece también ya que, al no ser *Python* un lenguaje de programación orientado a matrices, los tiempos de computación se disparan.

Por tanto, se trató de encontrar una alternativa que redujera el tiempo de ejecución siendo además la pérdida de precisión mínima. Con el fin de encontrar este comportamiento es habitual usar la técnica del *sparse GP* [31] la cual es explicada a continuación.

3.4.1 Explicación básica

La idea detrás de este decremento en el tiempo de ejecución está relacionada con una reducción en el número de muestras usadas en el *kernel* para estimar una nueva salida. Es trivial pensar que, un menor número de muestras implica una reducción en el tiempo de ejecución total y más cuando este ronda las 24 horas como ocurre en este trabajo.

La base teórica de esta técnica se considera que se encuentra fuera del alcance de este trabajo, pero se puede acceder a la bibliografía en el caso de querer profundizar en este tema [32], [33]. Básicamente, este subconjunto de muestras usadas para la estimación no son exactamente parte de las muestras de entrenamiento, sino que se parte del conjunto total de muestras, las cuales son tratadas con el fin de obtener un subconjunto de muestras que reúnan la información acerca del conjunto total de muestras. De esta manera, se puede tener teóricamente la misma información pero en un número menor de muestras, por lo que las operaciones matriciales necesitarán de menos tiempo de computación, reduciendo por tanto el tiempo total empleado para llevar a cabo la estimación. En la práctica, se verá como este subconjunto de muestras realmente no tienen la información completa, obteniendo resultados menos precisos.

4 Presentación del Escenario

En cuestiones de ciencia, la autoridad de miles no vale más que el humilde razonamiento de un único individuo.

GALILEO GALILEI

En el capítulo anterior se pudo desarrollar de una manera matemática y estadística los detalles a cerca del algoritmo de ML que se va a aplicar en este proyecto. Por tanto, y una vez presentada la base teórica del algoritmo, en este capítulo se va a introducir el **escenario** en el que se va a aplicar el mencionado algoritmo. Se van a dar detalles sobre el modelo de propagación usado, así como las características principales del algoritmo dentro de este escenario.

Es muy importante destacar que este capítulo está basado en el artículo en el cual descansa los aspectos fundamentales de este proyecto [34]. Como ya se comentó en el capítulo de introducción y en el de objetivos, mi aportación al estado del arte es *simplemente* convertir el código que fue desarrollado para elaborar dicho artículo a un lenguaje de programación de *software* libre. Por tanto, el lector debe acceder a este capítulo considerando que lo expuesto en él procede de investigaciones llevadas a cabo por personal ajeno al autor de esta memoria.

Por otro lado, se considera importante apartar este espacio para hacer un inciso en la notación, la cual puede llegar a ser una fuente de incertidumbre en este capítulo. A diferencia de la notación usada en [34], para indicar la dependencia temporal de los datos no va a usarse un superíndice sino que va a indicarse como se hace habitualmente. Esto es, si la variable x depende del tiempo, aparecerá en las ecuaciones como $x(t)$. Por otro lado, cuando se vaya a tratar con las ecuaciones que modelan el algoritmo para la regresión de las muestras (GPR), se va a usar la notación a nivel de muestra como se hizo a partir de la Sección 3.3.4.

Finalmente, este capítulo cuenta con tres partes bien diferenciadas: (1) Una primera donde se presenta el modelo, (2) una segunda parte donde se estudia la estimación de los parámetros que se desconocen dentro de ese modelo, y (3) por último una sección donde se presenta la aplicación del algoritmo de ML dentro de este escenario.

4.1 Escenario de aplicación de GPR

Como ya se comentó en el capítulo de introducción y en el de objetivos, este proyecto está basado en la estimación recursiva de campo recibido a partir de la información proporcionada por una serie de sensores de bajo coste los cuales se encuentran repartidos por una cierta zona espacial. La grandes preguntas que surge en este momento, y que animan a la realización de esta sección es: ¿en qué consiste esa información proporcionada por las medidas de los sensores?¿cómo podemos modelar esas medidas?¿es necesario estimar algún parámetro dentro de ese modelo? Todas esas preguntas serán respondidas a lo largo de este capítulo.

Antes de nada, se considera necesario mostrar un pequeño gráfico en el que se representa el problema a solucionar en este trabajo. Por tanto, en la Figura 4.1 se muestra de manera gráfica la distribución de los nodos y sensores con el fin de clarificar el escenario base.

Se parte de un espacio de 500×500m en el que se encuentran un total de M (1088) nodos distribuidos uniformemente en una rejilla, y en cuyas posiciones se tratará de estimar la señal recibida. Con el fin de llevar a cabo la estimación de la forma más exacta posible, se va a usar como muestras de entrenamiento

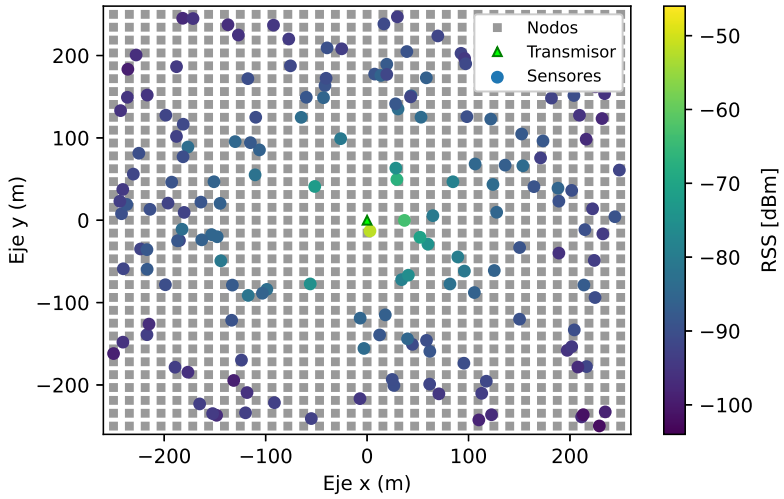

Figura 4.1 Escenario de aplicación del algoritmo.

la información de señal recibida proporcionada por un total de N (218) sensores distribuidos de manera aleatoria a lo largo del espacio de aplicación. En la Figura 4.1 se aprecia además como el triángulo representa al transmisor, el cual está situado en el centro del *grid* pero cuya localización se desconoce, los cuadrados corresponden a las posiciones de los nodos en las que se pretende estimar la señal recibida, y los círculos de colores representan a los sensores distribuidos aleatoriamente cuyas posiciones se conoce pero de forma aproximada. Se puede apreciar como el color de los sensores depende de la intensidad de señal recibida en su posición, siendo los sensores coloreados con tonos más claros aquellos que se encuentran más cerca del transmisor y en los cuales se recibe un valor más alto de intensidad de señal, mientras que los sensores con tonos más oscuros se encuentran más alejados del transmisor siendo la señal recibida en estos más débil.

A continuación, y una vez se tiene una idea general del problema, se va discutir acerca del desarrollo matemático presente en este escenario. Se partirá del modelo más simple en el que no existe ningún tipo de error en la posición de los sensores, para posteriormente ir añadiendo más complejidad al modelo base con el fin de modelar por completo el escenario en el que se encuentra este trabajo. Estos sensores comparten información acerca del **campo recibido** en ciertos puntos del espacio y por tanto, esa información sigue un modelo de propagación el cual está basado en un **modelo de pérdidas por camino** (*log-normal path loss model*). Este tipo de modelos dependen de numerosos parámetros entre los que se encuentran la localización de los sensores, el exponente de pérdidas por camino, la potencia transmitida y la localización de los puntos muertos (*shadowing*). La elección de este modelo está basada en la suposición de que tanto la posición del transmisor como la de los sensores, así como los efectos de *shadowing*, afectan al modelo de propagación.

De esta manera, y a partir de la aplicación del modelo ya mencionado, el campo recibido en su manera más simple puede expresarse como sigue:

$$\mathbf{z}(t) = \mathbf{1}P(t) - 10\alpha(t)\log_{10}(\mathbf{d}(t)) + \mathbf{v}(t) + \mathbf{w}(t) \tag{4.1}$$

En la Tabla 4.1, se muestra el significado de cada uno de los parámetros del modelo de manera resumida.

Conviene destacar algunos aspectos importantes que han sido mencionados en la Tabla 4.1 antes de añadir una mayor complejidad al modelo. En primer lugar, para el cálculo de la distancia se ha usado la siguiente fórmula:

$$d_{ij}(t) = \sqrt{(\mathbf{x}_i(t) - \mathbf{x}_j(t))^T(\mathbf{x}_i(t) - \mathbf{x}_j(t))} \tag{4.2}$$

En la que los términos x_{ij} corresponde a la posición de los N sensores que proporcionan la información necesaria, así como de los M puntos del espacio en los que se quiere calcular el campo recibido. Estas posiciones están plasmadas en un plano XY como se verá más adelante en este documento, por lo que únicamente cuenta con dos valores (uno para la posición respecto al eje x y otro para la posición respecto al eje y).

Tabla 4.1 Defínción de parámetros de Ecuación (4.1).

Parámetro	Definición
z(t) [dBm]	Medida de campo recibido en cada uno de los sensores.
P(t) [dBm]	Se trata de la potencia isotrópica radiada equivalente (PIRE) del transmisor el cual se encuenta en la posición x_0. Tanto la posición como la potencia transmitida son dos de los parámetros que será necesario estimar.
$\alpha(t)$	Es el exponente de pérdidas por camino en el instante t y tiene un valor que necesita ser estimado.
d(t) [m]	Se trata de la distancia entre los sensores y el transmisor. También puede aparecer el parámetro $\mathbf{d}_g$ en el que se indica la distancia entre el transmisor y los puntos en los que se quiere calcular el campo recibido.
v(t)	Este parámetro modela la atenuación debido al efecto del *shadowing* y se trata de una distribución Gaussiana de media cero y matriz de covarianza: Σ_v.
w(t)	Parámetro que añade un ruido de tipo Gaussiano a las muestras de campo recibida y sigue una distribución Gaussiana de media cero y matriz de covarianza: $\sigma_w^2\mathbf{I}_N$.

Por otro lado, cada término de la matriz de covarianza Σ_v que forma parte del modelado de la atenuación por efectos de *shadowing* se calcula como sigue:

$$Cov(v_i(t)v_j(t)) = \sigma_v^2 \exp\left(\frac{d_{ij}(t)}{D_{corr}}\right) \tag{4.3}$$

Donde el término D_{corr} modela la correlación entre las muestras, y cuyo valor será especificado más adelante.

Por último, y en relación al término de ruido añadido $\mathbf{w}(t)$, comentar que la matriz diagonal de covarianza tiene como valores el parámetros σ_w, el cual tiene un valor de $\sigma_w = \sqrt{7}$ (obtenido a partir de las referencias).

4.1.1 Imprecisión en la posición de los sensores

El anterior modelo está basado en una suposición en la que la posición de los sensores es perfectamente conocida. Esto podría darse en algunos escenarios de aplicación en el que los sensores son fijos y proporcionan medidas muy precisas. En cambio, en este proyecto se realiza una estimación a partir de un número muy elevado de medidas, cuya precisión es muy baja, las cuales son proporcionadas por sensores de bajo coste como pueden ser *smartphones*, cuya posición no sólo no es conocida de forma precisa sino que además puede variar a lo largo del tiempo (*crowdsourcing*). Es por ello necesario añadir al modelo original (Ecuación (4.1)) esta incertidumbre en la posición de los sensores con el fin de modelar el campo recibido de la manera más precisa posible.

Para indicar que la posición no es completamente conocida se usará la siguiente notación: $\hat{\mathbf{X}}(t) = [\hat{x}_1(t), \hat{x}_2(t)...\hat{x}_N(t)]$. De esta manera, la distancia respecto al transmisor tendrá también un valor poco preciso y será por tanto expresada de la siguiente forma: $\hat{\mathbf{d}}(t) = [\hat{d}_1(t), \hat{d}_2(t)...\hat{d}_N(t)]$.

Esta imprecisión en la posición de sensores se puede modelar a partir de la siguiente ecuación:

$$\hat{d}_i(t) = d_i(t) + \varepsilon_d \tag{4.4}$$

Se puede apreciar como a la distancia original se añade un término de *ruido* el cual sigue una distribución Gaussiana de media cero y varianza σ_d^2. Este valor de varianza es conocido ($\sigma_d = 13.16$m), por lo que se llega de manera sencilla a la siguiente expresión:

$$\hat{d}_i(t) \sim \mathcal{N}(d_i(t), \sigma_d) \tag{4.5}$$

Este modelado de posición de los sensores que refleja la incertidumbre de su localización hace que el modelo de la Ecuación (4.1) evolucione hacia un escenario más realista:

$$\mathbf{z}(t) = \mathbf{1}P(t) - 10\alpha(t)log_{10}(\hat{\mathbf{d}}(t)) + \mathbf{u}(t) + \mathbf{v}(t) + \mathbf{w}(t) \tag{4.6}$$

En el que se puede observar la aparición de un término nuevo: $\mathbf{u}(t)$. Este parámetro refleja dicha imprecisión y tiene la siguiente expresión:

$$\mathbf{u}(t) \sim \mathcal{N}\left(0, \rho_u^2(t)\hat{\mathbf{D}}(t)\right) \tag{4.7}$$

La procedencia del parámetro $\rho_u^2(t)$ [mdB] puede verse detallada en el apéndice de [34], donde además se demuestra de igual forma la aparición del término $\mathbf{u}(t)$ en la Ecuación (4.6). Los dos parámetros con los que se calcula la matriz de covarianza en la Ecuación (4.7) tienen las siguientes expresiones:

$$\rho_u^2(t) = 10\alpha(t)\sigma_d \log_{10}(e) = 200 \tag{4.8}$$

$$\hat{\mathbf{D}}(t) = diag(1/\hat{d}_1^2(t), 1/\hat{d}_2^2(t)...1/\hat{d}_N^2(t)) \tag{4.9}$$

4.1.2 Ejemplo con imprecisión en la posición

Una vez se ha avanzado en la formulación matemática del modelo incluyendo la imprecisión en la posición de los sensores, en esta sección se va a presentar un ejemplo gráfico donde se muestra la importancia de añadir este término de imprecisión en el modelo original.

Para ello, se va a realizar una simulación *acotada*, en la que los parámetros son perfectamente conocidos y las fórmulas son más simples que el modelo de propagación propuesto en la Ecuación (4.6). Por tanto, se va a realizar un estudio que relaciona disminución de la potencia con la distancia. La ecuación base de este estudio es la siguiente:

$$E(z_i) = P - 10\alpha \log_{10} d_i \tag{4.10}$$

En la que los valores de α y P son, respectivamente 3.5 y 0 dBm.

Esta gráfica incluirá distintas curvas correspondientes a situaciones en las que (1) la distancia es perfectamente conocida (roja), (2) la distancia tiene un error de 10 metros (azul) y (3) el error en la distancia está condicionado al valor de ρ_u (negra a rayas), el cual es de 200 mdB como ya se comentó anteriormente. Estas curvas se muestran en la Figura 4.2.

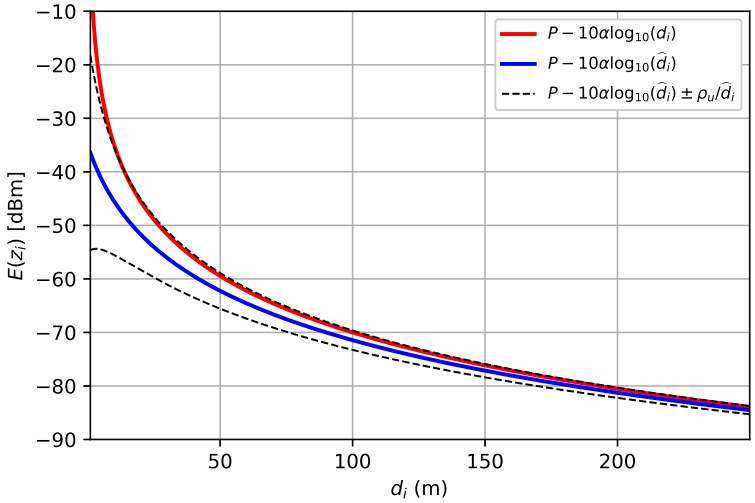

Figura 4.2 Nivel de potencia respecto a la distancia.

La principal conclusión obtenida a partir del análisis de la Figura 4.2 es la siguiente: conforme más cerca se encuentran los sensores de la posición del transmisor, mayor será el efecto del error de posición de los sensores. Para demostrar esta conclusión de manera empírica y no visual se muestra la Figura 4.3 donde se ha representado el porcentaje de error en el cálculo de la potencia debido al error de posición de los sensores. En dicha gráfica se aprecia de manera más que clara como el porcentaje de error es muy elevado en distancias cercanas al transmisor respecto a distancias más alejadas. A partir de una distancia en torno a 50 metros el porcentaje de error es prácticamente nulo por lo que el efecto de la imprecisión en la posición empieza a ser despreciable. Por tanto, habrá que tener muy en cuenta estas consideraciones a la hora de estudiar la información proporcionada por sensores que se encuentre en torno al transmisor en distancias muy cercanas.

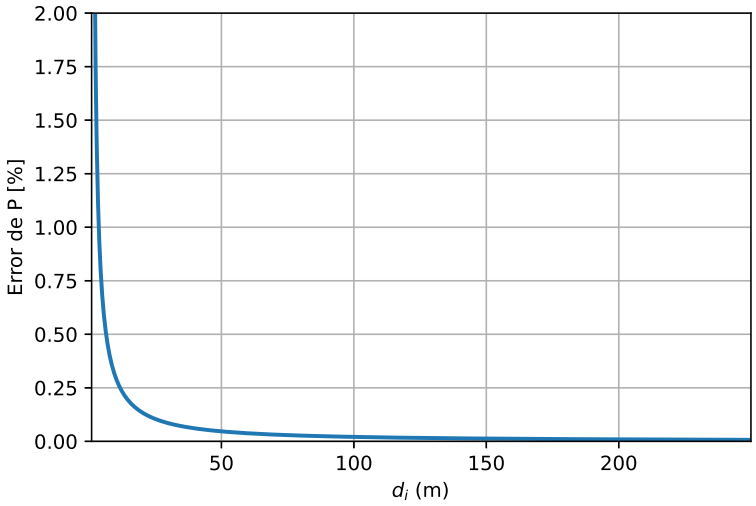

Figura 4.3 Porcentaje de error en la potencia.

4.2 Estimación de parámetros del modelo

Como se ha comentado en el estudio de las Ecuaciones (4.1) y (4.6) y la Tabla 4.1, existen una serie de parámetros que han de ser estimados con el fin de modelar el campo recibido en los sensores de manera correcta. Estos parámetros son: la posición del transmisor, el exponente de pérdidas por camino y la potencia isotrópica radiada equivalente (PIRE) recibida en los sensores. Por tanto se dedicará un espacio en esta memoria al estudio de la estimación de estos parámetros.

4.2.1 Posición del transmisor

Se comenzará presentando el estudio acerca de la estimación de la posición del transmisor. Para ello, se ha usado un algoritmo de localización basado en la fuerza del campo recibido (RSS), el cual es ampliamente usado en localización para aplicaciones inalámbricas. Cabe destacar que, por lo general, los algoritmos de localización basados en medidas de RSS son bastante imprecisos y por tanto, el error en la posición estimada es bastante elevado. Esto es debido principalmente a las situaciones a la que se enfrenta normalmente cualquier sistema de comunicaciones inalámbricas, como puede ser la degradación por efectos de *shadowing* o las los desvanecimientos de señal. Estas circunstancias hacen que el algoritmo de localización proporcione resultados muy poco precisos y variantes con el tiempo en función de las características de la señal recibida [35]. Aún así, y como se ha comentado anteriormente, son algoritmos ampliamente usados debido a su bajo coste y complejidad [36].

Es por ello que para la localización del transmisor se propone un algoritmo en el que se añaden unos pesos a la medidas proporcionadas por los sensores y, de esta manera, y como se podrá comprobar a partir de las fórmulas, se dará una mayor importancia a la información proporcionada por sensores con mayor RSS. Este algoritmo recibe el nombre de *weighted centroid approach* [37].

Adicionalmente, en este trabajo se ha añadido una variante al algoritmo de localización añadiendo una dependencia temporal a las muestras para, de esta forma, incluir la información proporcionada por muestras anteriores en el cálculo de la posición actual.

En relación a la notación, y al ser un parámetro estimado, la posición del transmisor cambiará su representación a: $\hat{x}_0$. Por tanto, la distancia entre los sensores y el transmisor, la cual es necesaria de calcular como se aprecia en la Ecuación (4.6), se estudiará a partir de la siguiente expresión:

$$\hat{\mathbf{d}}_i(t) = |\hat{\mathbf{x}}_i(t) - \hat{\mathbf{x}}_0(t)| \tag{4.11}$$

Entrando de manera más analítica en el cálculo de la posición, ésta se realiza a partir de la siguiente expresión:

$$\hat{\mathbf{x}}_0(t) = \frac{1}{\hat{w}(t)} \left(\sum_{i=1}^{N} w_i(t)\hat{\mathbf{x}}_i(t) + \hat{w}(t-1)\hat{\mathbf{x}}_0(t-1) \right) \tag{4.12}$$

En la que los valores de los pesos (w) se calculan como sigue:

$$w_i(t) = 10^{\frac{z_i(t)}{10}} \tag{4.13}$$

$$\hat{w}(t) = \hat{w}(t-1) + \sum_{i=1}^{N} w_i(t) \tag{4.14}$$

Como se mencionó al comienzo de esta sección, los algoritmos de localización basados en RSS proporcionan medidas muy imprecisas por lo que, además de añadir los pesos y la dependencia temporal, se puede llevar a cabo una serie de cálculos que perfeccionen aún más la posición estimada. Más adelante en el documento, se mostrarán los resultados de las simulaciones llevadas a cabo con refinamiento de la posición y sin refinamiento con el fin de mostrar con un mayor detalle el efecto de este cálculo.

La idea detrás de este refinamiento se encuentra en el algoritmo propuesto en [36] para localización. En este interesante artículo se propone un algoritmo en el que, tanto el exponente de pérdidas por camino como la potencia transmitida son estimadas en lugar de ser consideradas como fijas. Se menciona igualmente que, un error habitual consiste en considerar como fijos estos parámetros (3.5 y 0 dBm respectivamente) a lo largo del tiempo, consideración la cual no modela de manera eficaz un escenario real. Como en este proyecto, estos dos parámetros son igualmente estimados, y se ha tomado como referencia este estudio para llevar a cabo una mejora de la estimación. Por tanto, y una vez estimadas la potencia y el exponente de pérdidas por camino como se explicará en las siguientes secciones, la posición del transmisor se puede re-estimar como sigue:

$$\hat{\mathbf{x}}_0(t) = \min_{x_0} \sum_{i=1}^{N} \left(z_i(t) - \hat{\mu}_P(t) + 10\hat{\mu}_\alpha(t) \log_{10} \sqrt{(\mathbf{x}_i(t) - \mathbf{x}_0)^T (\mathbf{x}_i(t) - \mathbf{x}_0)} \right)^2 \tag{4.15}$$

Este cálculo proporciona una estimación de la posición más precisa y además permite en una segunda iteración, re-estimar los valores de exponente de pérdidas por camino y de potencia transmitida a partir de este valor más exacto de posición del transmisor proporcionando unas estimaciones igualmente más precisas de estos dos parámetros. La estimación en primera instancia de estos dos parámetros se detalla en la siguiente sección.

4.2.2 Exponente de pérdidas por camino (α) y potencia transmitida (PIRE)

Para los dos parámetros restantes del modelo que no se conocen se va a realizar una estimación bayesiana. Ya que es necesario usar la información conjunta de ambos parámetros para llevar a cabo la estimación, esta sección va a estar enfocada en la presentación de los cálculos necesarios para estimar los dos parámetros, en lugar de realizar dos secciones por separado.

Como se ha comentado al comienzo de la sección, la estimación de estos parámetros está basada en el Teorema de Bayes, es por ello que será necesario usar el método de Bayes empírico (*empirical Bayes method* [38]) para de esta manera llevar a cabo la estimación correspondiente. Estos parámetros estimados forman parte de los hiperparámetros del modelo, y aunque más adelante se va a profundizar un poco más en el modelo de GPR y sus hiperparámetros, este punto de la memoria sirve como introducción a cuatro de ellos. Estos cuatro hiperparámetros son: $\boldsymbol{\Theta} = [\mu_\alpha, \sigma_\alpha, \mu_P, \sigma_P]$. Donde μ_α y μ_P son la media de la estimación del exponente de pérdidas por camino y potencia transmitida respectivamente, y σ_α y σ_P son la varianza de dicha estimación.

Cabe destacar en relación a la notación, la eliminación de la dependencia temporal con el fin de relajar un poco las ecuaciones que se desarrollan en esta sección. Es importante mencionar este aspecto ya que estos parámetros estimados tienen una dependencia temporal.

La estimación realizada por este tipo de técnicas parten de una distribución *a priori* estimada a partir de una serie de muestras. En este caso, los datos para llevar a cabo la estimación serán las medidas proporcionadas

por los sensores así como la posición del transmisor. Por tanto, y usando el teorema de Bayes, se puede hallar la función densidad de probabilidad conjunta:

$$p(\alpha,P|\mathbf{z},\hat{\mathbf{X}},\boldsymbol{\Theta}) = \frac{p(\mathbf{z}|\alpha,P,\hat{\mathbf{X}})p(\alpha|\boldsymbol{\Theta})p(P|\Theta)}{p(\mathbf{z}|\hat{\mathbf{X}},\boldsymbol{\Theta})} \tag{4.16}$$

Si se parte del supuesto en el que los parámetros α y P son distribuciones Gaussianas de media μ_α, μ_P y varianza σ_α, σ_P, la expresión $p(\mathbf{z}|\alpha,P,\hat{\mathbf{X}})$ (obtenida del primer término del numerador de la Ecuación (4.16)) es igualmente Gaussiana con media:

$$\mu_{z|\alpha,P} = \mathbf{1}P - \hat{\mathbf{q}}\alpha \tag{4.17}$$

y matriz de covarianza:

$$\boldsymbol{\Sigma}_{z|\alpha,P} = \rho_u^2\hat{\mathbf{D}} + \boldsymbol{\Sigma}_v + \sigma_w^2\mathbf{I}_N \tag{4.18}$$

Por otro lado, el término del denominador $p(\mathbf{z}|\hat{\mathbf{X}},\boldsymbol{\Theta})$ es también Gaussiano y tiene una media:

$$\mu_z = 1\mu_P - \hat{q}\mu_\alpha \tag{4.19}$$

y una matriz de covarianza que tiene la siguiente expresión:

$$\boldsymbol{\Sigma}_z = \boldsymbol{\Sigma}_{z|\alpha,P} + \sigma_\alpha^2\hat{\mathbf{q}}\hat{\mathbf{q}}^T + \sigma_P^2\mathbf{1}_{NxN} \tag{4.20}$$

Una vez ya se conocen todos los términos de la ecuación (4.16), se puede aplicar el método de Bayes empírico para de esta manera estimar la media y varianza de los parámetros. Como es bien sabido, y a partir de un análisis de las referencias, se puede concluir que para estimar estos parámetros será necesario llevar a cabo una minimización de la siguiente función para el calculo de la media:

$$\begin{bmatrix}\hat{\mu}_P\\ \hat{\mu}_\alpha\end{bmatrix} = \min_{\mu_P,\mu_\alpha}\left|\left|[\mathbf{1}-\hat{\mathbf{q}}]\begin{bmatrix}\mu_P\\ \mu_\alpha\end{bmatrix} - \mathbf{z}\right|\right|^2 \tag{4.21}$$

El cálculo de la media no es tan trivial y hay que tener en cuenta una serie de consideraciones que incluyen a la media y varianza de las muestras.

- Primeramente la media de las medidas de RSS proporcionada por los sensores se puede calcular a partir de: $\mu_z = \mathbf{1}\hat{\mu}_P - \hat{\mathbf{q}}\hat{\mu}_\alpha$
- Además, y una vez calculada la medidas de RSS, se puede calcular la matriz de covarianza de esas medidas como sigue: $\boldsymbol{\Sigma}_z = (\mathbf{z}-\hat{\mu})(\mathbf{z}-\hat{\mu})^T$

Con esto mente, la función que será necesario minimizar con el fin de encontrar la varianza de los parámetros es:

$$\begin{bmatrix}\hat{\sigma}_P^2\\ \hat{\sigma}_\alpha^2\end{bmatrix} = \min_{\sigma_P^2,\sigma_\alpha^2}\left|\left|[\mathbf{1}-diag(\hat{\mathbf{q}}\hat{\mathbf{q}}^T)]\begin{bmatrix}\sigma_P^2\\ \sigma_\alpha^2\end{bmatrix} - diag(\boldsymbol{\Sigma}_z - \boldsymbol{\Sigma}_{z|\alpha,P})\right|\right|^2 \tag{4.22}$$

Por último hay que tener en cuenta lo mencionado en la sección anterior ya que, como se aprecia en las Figuras 4.2 y 4.3, la información obtenida a partir de los sensores que se encuentran en posiciones más próximas al transmisor tienen un mayor error. Por tanto, habrá que darle un menor peso a estas medidas a la hora de realizar la estimación de estos hiperparámetros (a partir de ahora y de cara a la siguiente sección se denominarán de esta forma). Para ello la ecuación (4.21) se modifica de la manera que se describe a continuación:

$$\begin{bmatrix}\hat{\mu}_P\\ \hat{\mu}_\alpha\end{bmatrix} = \min_{\mu_P,\mu_\alpha}\left|\left|\left[\mathbf{1}\sqrt{\hat{\mathbf{D}}}^{-1} - \hat{\mathbf{q}}\sqrt{\hat{\mathbf{D}}}^{-1}\right]\begin{bmatrix}\mu_P\\ \mu_\alpha\end{bmatrix} - \mathbf{z}\sqrt{\hat{\mathbf{D}}}^{-1}\right|\right|^2 \tag{4.23}$$

En este punto es importante recordar que: $\hat{D} = diag(1/\hat{d_1^2}, 1/\hat{d_2^2}...1/\hat{d_N^2})$.

4.3 GPR aplicado en este escenario

Por último, y una vez se ha llevado a cabo una introducción tanto del escenario como del modelo de propagación empleado en este proyecto, así como de los mecanismos de estimación de los parámetros desconocidos, en esta sección se va a estudiar el algoritmo de GPR empleado para estimar valores de RSS en distintos puntos del espacio. Esta sección es considerada por el autor de esta memoria como la más importante desde el punto de vista teórico, ya que reúne la mayoría de conceptos presentados en secciones y capítulos anteriores. Esto es, **se va a hacer uso del algoritmo de GPR explicado a continuación para estimar valores de RSS, partiendo del modelo de la Ecuación** (4.6) **y usando los hiperparámetros estimados a partir de las ecuaciones de la sección anterior**.

Como ya se hizo en la sección donde se introducían los conceptos básicos sobre GPR, se va a diferenciar el estudio de este algoritmo aplicado en este escenario bajo dos supuestos: (1) el campo recibido es estático por lo que únicamente se toma la muestra actual y (2) el campo recibido varía a lo largo del tiempo por lo que habrá que añadir el factor de olvido con el fin de dar una mayor importancia a las muestras más recientes.

4.3.1 Campo constante a lo largo del tiempo

En esta primera sección se va a considerar que el campo recibido en los sensores es constante a lo largo del tiempo, y por tanto no se va a usar información de instantes previos para llevar a cabo la estimación de RSS en los distintos nodos. Además, y como se lleva comentando a lo largo de toda la memoria, se va a hacer uso de la técnica de GPR para poder llevar a cabo dicha estimación. Por tanto, habrá que *alterar* el modelo propuesto en la Ecuación (4.6) para que sea representado a partir de un proceso Gaussiano más un ruido:

$$\mathbf{z}(t) = f(\hat{\mathbf{X}}(t)) + \mathbf{n}(t) \tag{4.24}$$

El ruido se trata de un ruido Guassiano el cual modela los siguientes términos de la Ecuación (4.6):

- La imprecisión en la posición de los sensores: $\mathbf{u}(t)$.
- El ruido propio de las muestras observadas: $\mathbf{w}(t)$.

De esta manera y de acuerdo a lo presentado en la Sección 4.1 acerca de los dos términos anteriores, el ruido queda modelado como una distribución Gaussiana de media cero y matriz de covarianza: $\Sigma_n = \sigma_w \mathbf{I}_N + \rho_u^2 \hat{\mathbf{D}}(t)$.

Por otro lado, el resto de términos del modelo se agrupan formando el proceso Gaussiano. Como fue comentado en su sección correspondiente, todo proceso Gaussiano queda perfectamente definido a partir de una función media y una función de covarianza o *kernel*, cuyas expresiones se muestran a continuación:

$$\mathbf{m}_{\hat{\mathbf{X}}}(t) = \mathbf{1}\hat{\mu}_P - \hat{\mathbf{q}}(t)\mu_\alpha(t) \tag{4.25}$$

$$\mathbf{K}_{\hat{\mathbf{X}}}(t) = \mathbf{K}(\hat{\mathbf{X}}(t), \hat{\mathbf{X}}(t)) \tag{4.26}$$

La elección del *kernel* es uno de los retos fundamentales cuando se trabaja con algoritmos de GP y, para este trabajo se ha usado una combinación de dos *kernel* más un término *bias* como se detalla a continuación:

$$k(x_i, x_j) = \sigma_k \exp\left(-\frac{\sqrt{(x_i - x_j)(x_i - x_j)^T}}{2l^2}\right) + \sigma_\alpha^2 \hat{q}(x_i)\hat{q}(x_j) + \sigma_P^2 \tag{4.27}$$

donde:

$$\hat{q}(x_i) = 10\log_{10}\sqrt{(x_i - \hat{x}_0)(x_i - \hat{x}_0)^T} \tag{4.28}$$

De la expresión del *kernel* expuesta en la Ecuación (4.27) se pueden realizar los comentarios que se enumeran a continuación:

1. El primer término se asemeja mucho al *kernel* RBF en el que los valores de los términos de la matriz de covarianza son cercanos a uno cuando las muestras están muy próximas, pero añadiendo un término una raíz cuadrada al resultado de la distancia con el fin de obtener unos resultados más conservadores cuando las muestras están distantes. Este término recibe el nombre de *non-squared exponential* (NSE) y se uso está menos extendido que el *kernel* RBF. En este apartado se aprecian dos términos, σ_k y l los

cuales no han sido mencionados anteriormente. Se trata de dos valores que pueden ser añadidos como parte del conjunto final de hiperparámetros, y por tanto, su valor es optimizado con el fin de encontrar el *kernel* que mejor se adapte a las muestras de entrenamiento.

2. El segundo término modela la incertidumbre debido a la estimación del exponente de pérdidas por camino y recibe el nombre de *kernel* LOG. Se trata también de una expresión muy poco extendida. Tanto este término como el anterior no suelen encontrarse entre los *kernel* predefinidos por defecto en las librerías de GP, por lo que tuvieron que ser implementados *hard-code* por parte del autor de esta memoria. Este proceso se detallará en un capítulo posterior.

3. El tercer y último término modela, al igual que el segundo, la incertidumbre debido a la estimación de la potencia transmitida. Es un término que ayuda a desplazar el resultado final en torno a un valor concreto.

Tal como se enfatizó en la sección de presentación de GPR, la función densidad de probabilidad de la estimación se obtiene a partir de la distribución *a posteriori* calculada en función de una distribución *a priori*, la cual es una función densidad de probabilidad conjunta entre las muestras de entrenamiento y la muestras de *test*. Primeramente, la función densidad de probabilidad *a priori* se muestra a continuación:

$$p(\mathbf{f}_g(t)|\boldsymbol{\Theta}(t)) \sim \mathcal{N}(\mathbf{m}_{\mathbf{X}_g}(t), \mathbf{K}_{\mathbf{X}_g}(t)) \tag{4.29}$$

Donde las expresiones de función media y función matriz de covarianza quedan:

$$\mathbf{m}_{\mathbf{X}_g}(t) = \mathbf{1}\hat{\mu}_P - \mathbf{q}_g(t)\mu_\alpha(t) \tag{4.30}$$

$$\mathbf{K}_{\mathbf{X}_g}(t) = \mathbf{K}(\mathbf{X}_g(t), \mathbf{X}_g(t)) \tag{4.31}$$

Y además, aparece un término que no había aparecido antes: $\mathbf{q}_g = [10\log_{10}(\hat{d}_{g1})...10\log_{10}(\hat{d}_{gM})]^T$. Donde se aprecia el subíndice $\hat{d}_{gM}$ indicando la distancia entre los M nodos donde se quiere calcular el campo recibido y la posición del transmisor. De ahí que, aunque la posición de los nodos sea perfectamente conocida, al no ser la del transmisor, se añade el *gorro* que indica que se trata de un valor estimado e impreciso.

Una vez conocida la distribución *a priori*, la expresión de la función densidad de probabilidad conjunta se calcula como en cualquier aplicación de GPR:

$$\begin{bmatrix}\mathbf{z}(t)\\ \mathbf{f}_g(t)\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{m}_{\hat{\mathbf{X}}}(t)\\ \mathbf{m}_{\mathbf{X}_g}(t)\end{bmatrix}, \begin{bmatrix}\mathbf{K}_{\hat{\mathbf{X}}}(t)+\boldsymbol{\Sigma}_n(t) & \mathbf{K}_{\mathbf{X}_g,\hat{\mathbf{X}}}(t)\\ \mathbf{K}_{\mathbf{X}_g,\hat{\mathbf{X}}}(t) & \mathbf{K}_{\mathbf{X}_g}(t))\end{bmatrix}\right) \tag{4.32}$$

Se trata de la expresión común de GPR que aparecía constantemente en el capítulo de introducción del algoritmo pero cuya notación y expresiones han sido adaptadas a este escenario. Por tanto y como último paso, únicamente habrá que calcular la función densidad de probabilidad condicionada a las muestras, la posición estimada de los sensores y los valores estimados de los hiperparámetros para de esta manera encontrar la media y la matriz de covarianza que modela la distribución a partir de la cual se puede calcular el campo recibido en los M nodos:

$$p(\mathbf{f}_g(t)|\mathbf{X}_g,\hat{\mathbf{X}}(t),\mathbf{z}(t),\hat{\boldsymbol{\Theta}}) \sim N(\mu_g(t),\boldsymbol{\Sigma}_g(t)) \tag{4.33}$$

Donde los términos necesarios para calcular la media y la covarianza se detallan a continuación:

$$\mu_g(t) = \mathbf{m}_{\mathbf{X}_g}(t) + \mathbf{K}_{\mathbf{X}_g,\hat{\mathbf{X}}}(t)\left(\mathbf{C}_{\hat{\mathbf{X}}}(t)\right)^{-1}(\mathbf{z}(t) - \mathbf{m}_{\hat{\mathbf{X}}}(t)) \tag{4.34}$$

$$\boldsymbol{\Sigma}_g(t) = \mathbf{K}_{\mathbf{X}_g}(t) - \mathbf{K}_{\mathbf{X}_g,\hat{\mathbf{X}}}(t)\left(\mathbf{C}_{\hat{\mathbf{X}}}(t)\right)^{-1}\mathbf{K}_{\mathbf{X}_g,\hat{\mathbf{X}}}(t) \tag{4.35}$$

$$\mathbf{C}_{\hat{\mathbf{X}}}(t) = \mathbf{K}_{\hat{\mathbf{X}}}(t) + \boldsymbol{\Sigma}_n(t) \tag{4.36}$$

Estas tres ecuaciones son las más importantes de esta sección y será necesario acudir a ellas cuando se quiera realizar la estimación del campo recibido. Se verá en su capítulo correspondiente como son implementadas exactamente de la misma forma en el código de *Python* que lleva a cabo dicha estimación.

4.3.2 Campo variante a lo largo del tiempo

Como se ha mencionado varias veces durante el trascurso de esta memoria, se pretende que el modelo final trate de simular de la forma más realista posible el escenario de aplicación de GPR. Por tanto, en una situación real, el campo recibido en los sensores no va a ser constante a lo largo del tiempo, sino que la señal sufrirá de desvanecimientos, los sensores desplazarán su posición o el transmisor variará su potencia de transmisión, lo que provoca que el valor de RSS en los sensores varíe conforme pasa el tiempo.

La dinámica que va a ser usada para tener en cuenta este fenómeno será tomar la distribución *a posteriori* calculada, como distribución *a priori* en el siguiente instante de tiempo. Además, se van a tener en cuenta las muestras anteriores pero con algunas consideraciones, ya que será necesario dar una mayor relevancia a las muestras más recientes. Para ello, se va a hacer uso del factor de olvido (*forgetting factor*) estudiado en una sección anterior. De esta manera se tiene una combinación lineal entre la muestra actual y las muestras pasadas pero ponderadas a partir del factor de olvido. Por último, habrá que tener en cuenta que tanto la potencia transmitida como el exponente de pérdidas por camino tienen que ser estimados nuevamente en cada iteración.

Las ecuaciones con el factor de olvido añadido se muestran a continuación:

$$\mu_g(t) = \mathbf{m}_{\mathbf{X}_g}(t) + (1-\lambda)\mu_{prior}(t) + \lambda\mu_{post}(t) \tag{4.37}$$

$$\mathbf{\Sigma}_g(t) = \mathbf{K}_{\mathbf{X}_g}(t) - \left((1-\lambda)\mathbf{\Sigma}_{prior}(t) + \lambda\mathbf{\Sigma}_{post}(t)\right) \tag{4.38}$$

Donde cada término se desglosa a continuación:

$$\mu_{post}(t) = \mathbf{K}_{\mathbf{X}_g,\hat{\mathbf{X}}}(t)\left(\mathbf{K}_{\hat{\mathbf{X}}}(t) + \mathbf{\Sigma}_n(t)\right)^{-1}(\mathbf{z}(t) - \mathbf{m}_{\hat{\mathbf{X}}}(t)) \tag{4.39}$$

$$\mathbf{\Sigma}_{post}(t) = \mathbf{K}_{\mathbf{X}_g,\hat{\mathbf{X}}}(t)\left(\mathbf{K}_{\hat{\mathbf{X}}}(t) + \mathbf{\Sigma}_n(t)\right)^{-1}\mathbf{K}_{\mathbf{X}_g,\hat{\mathbf{X}}}(t) \tag{4.40}$$

$$\mu_{prior}(t) = \mu_g(t-1) - \mathbf{m}_{\mathbf{X}_g}(t-1) \tag{4.41}$$

$$\mathbf{\Sigma}_{prior}(t) = \mathbf{K}_{\mathbf{X}_g}(t-1) - \mathbf{\Sigma}_g(t-1) \tag{4.42}$$

Al igual que para el caso estático, estas ecuaciones tendrán una gran importancia a la hora de programar el algoritmo de estimación de campo recibido en los nodos y, por tanto, se retomará su estudio en un capítulo posterior.

4.4 Variación de la potencia transmitida a lo largo del tiempo

Una situación lógica que se podría dar en este escenario está relacionada con la variación de la potencia transmitida. La mayoría de protocolos inalámbricos modernos incluyen unos mecanismos de reducción de potencia transmitida con el fin de ahorrar batería cuando sea necesario. Por tanto, en esta sección se va a estudiar la precisión de las estimaciones en estas situaciones particulares y además se va a hacer uso del concepto de *Bayesian Cramer-Rao bound* (BCRB) con el fin de obtener una cota en la estimación.

A lo largo de este apartado, el estudio se va a centrar en la estimación de campo recibido en un nodo aislado del *grid*, para el cual se estima una media (μ_{g_i}) y una varianza ($\sigma^2_{g_i}$). El límite inferior para esta estimación es obtenido de manera muy similar a como se hace en [50] y [51], a partir del cálculo de BCRB como sigue:

$$\mathbb{E}_{z,f_{g_i}}[(f_{g_i} - \hat{f}_{g_i})^2] \geq \left(\mathbb{E}_{z,f_{g_i}}\left[\left(\frac{\partial}{\partial f_{g_i}}\ln p(\mathbf{z}, f_{g_i})\right)^2\right]\right)^{-1} \tag{4.43}$$

Hay que tener en cuenta que el error no procede únicamente de la estimación del campo recibido sino también de la estimación de la posición del transmisor y de la potencia transmitida y exponente de pérdidas. Es por ello que en el trabajo original [34] se propone un nuevo método que recibe el nombre de *hybrid Cramer-Rou bound* (HCRB) y que permite introducir el error de todas las estimaciones realizadas en el trabajo. No se va a hacer mucho énfasis en la demostración matemática de la técnica, pero se puede acceder a

la bibliografía con el fin de estudiar la procedencia de las siguientes expresiones, a partir de las cuales se pueden calcular estos límites.

$$\mathbb{E}_{z,f_{g_i}} \geq \sigma_{g_i}^2 + \mathbf{g}_{g_i}^{-1}\mathbf{M}_{g_i}^{\top}\mathbf{g}_{g_i} \tag{4.44}$$

De esta ecuación base se puede demostrar que cada término se calcula a partir de las siguientes expresiones:

$$\mathbf{g}_{g_i} = \begin{bmatrix} 1 \\ -10\log_{10}(d_{g_i}) \\ \frac{c}{d_{g_i}^2}(\hat{\mathbf{x}}_0 - \mathbf{x}_{g_i}) \end{bmatrix} - \begin{bmatrix} \mathbf{1} & -\hat{\mathbf{q}} & \mathbf{A} \end{bmatrix} \left(C_{\hat{\mathbf{X}}}\right)^{-1} \mathbf{k}_{\mathbf{x}_{g_i},\hat{\mathbf{X}}}^{\top} + \begin{bmatrix} 0 \\ 0 \\ \mathbf{m}_{\hat{\mathbf{X}}}^{\top}\left(C_{\hat{\mathbf{X}}}\right)^{-1}\mathbf{A}_1\left(C_{\hat{\mathbf{X}}}\right)^{-1}\mathbf{k}_{\mathbf{x}_{g_i},\hat{\mathbf{X}}}^{\top} \\ \mathbf{m}_{\hat{\mathbf{X}}}^{\top}\left(C_{\hat{\mathbf{X}}}\right)^{-1}\mathbf{A}_2\left(C_{\hat{\mathbf{X}}}\right)^{-1}\mathbf{k}_{\mathbf{x}_{g_i},\hat{\mathbf{X}}}^{\top} \end{bmatrix} \tag{4.45}$$

$$\mathbf{M}_{g_i} = \begin{bmatrix} \mathbf{1} & -\hat{\mathbf{q}} & \mathbf{A} \end{bmatrix} \left(C_{\hat{\mathbf{X}}}\right)^{-1} \begin{bmatrix} \mathbf{1} & -\hat{\mathbf{q}} & \mathbf{A} \end{bmatrix} \tag{4.46}$$

Donde:

$$c = -10\mu_{\alpha}\log_{10}(e) \tag{4.47}$$

$$\mathbf{A} = c\hat{\mathbf{D}}(\mathbf{J}_{Nx2}diag(\hat{\mathbf{x}}_0) - \hat{\mathbf{X}}) \tag{4.48}$$

$$\mathbf{A}_1 = -2\sigma_u^2 diag\{\hat{\mathbf{D}}^2(\mathbf{1}\hat{\mathbf{x}}_0(1) - \hat{\mathbf{X}}(:,1))\} \tag{4.49}$$

$$\mathbf{A}_2 = -2\sigma_u^2 diag\{\hat{\mathbf{D}}^2(\mathbf{1}\hat{\mathbf{x}}_0(2) - \hat{\mathbf{X}}(:,2))\} \tag{4.50}$$

Estas ecuaciones serán añadidas al código desarrollado con el fin de obtener los márgenes entre los que se van a mover las estimaciones en los casos en los que varíe la potencia transmitida.

5 Python e implementación de GPR

La ciencia es el padre del conocimiento, pero las opiniones son las que engendran la ignorancia.

HIPÓCRATES

Una vez se han presentado las ecuaciones matemáticas deducidas para este proyecto con el fin de estimar el campo recibido a partir de la técnica de GPR, el siguiente paso en la memoria sería la explicación e introducción del código desarrollado en este proyecto. Pero antes de presentar el código en el capítulo siguiente, se va a hacer uso de este capítulo para presentar tanto el entorno de programación usado, así como las librerías que han sido utilizadas a lo largo del código.

La principal finalidad de este capítulo reside en proporcionar al lector una idea general del entorno de programación usado con el fin de facilitar el entendimiento del código que va a ser presentado en un capítulo posterior. Este código utiliza numerosas librerías que, de no haber sido presentadas con anterioridad, podrían ser fuente de malentendidos.

Este capítulo tiene dos secciones: (1) una primera donde se presentan las librerías usadas a lo largo del código, mostrando especial interés en aquellas con una mayor relevancia en el proyecto y (2) una segunda sección donde se introduce el entorno virtual creado.

5.1 Librerías usadas

La elección de librerías es uno de los aspectos más importantes cuando se trabaja con cualquier lenguaje de programación, adquiriendo una importancia aún mayor en aquellos que son de código abierto como *Python*. Por tanto, en este apartado se van a listar las librerías usadas en este proyecto, realizando además una breve descripción de ellas con el fin de justificar su presencia en este trabajo. Dicha lista se muestra a continuación:

- **numpy**: Es la librerías más usada e imprescindible dentro de este proyecto ya que da soporte para vectores y matrices, haciendo que el uso de estos sea más cómodo. Es una librería que consigue *igualar* de alguna manera las facilidades de *MATLAB* en cuanto a la utilización de vectores y matrices, convirtiéndose de este modo en la librería más usada a lo largo de este proyecto. Tanto las operaciones matriciales como la inicialización y uso de vectores se llevan a cabo de manera muy simple gracias a esta librería [43].
- **matplotlib**: Esta librería es usada principalmente para la representación de gráficos a partir de datos contenidos en vectores, por lo que soporte el formato de matrices y vectores de *numpy*. Su uso es bastante similar a *MATLAB* lo que hace que la representación sea llevada a cabo de manera muy simple, y su aparición en el código está centrada en la Clase *graficas* [44].
- **scipy**: Se trata de otra librería ampliamente usada ya que también soporta el formato de vectores y matrices de *numpy*. Es una librería la cual tiene implementados en su código fuente numerosos algoritmos para optimización, así como módulos para álgebra lineal, procesamiento de imagen... Dentro de este proyecto, su uso se reduce a la utilización de los algoritmos de minimización necesarios para llevar a cabo la estimación de los parámetros α y P. La función *lsq_linear* que aparece en la línea 6 del Código 6.3 es un ejemplo de método implementado en esta librería y aplicado en este proyecto [45].

5.1.1 Librería GPy

[46] Debido a su importancia dentro de este proyecto se va a dedicar un apartado exclusivamente a esta librería. El aprendizaje en torno al uso de librerías que implementen algoritmos de ML es uno de los objetivos principales de este proyecto. Tras el uso del algoritmo de GPR de esta librería, el autor de este proyecto se encuentra en disposición de usar otros algoritmos dentro de la misma, así como directamente otras librerías que incluyan algoritmos de GP. Por tanto, el correcto uso de esta librería está relacionado directamente con el cumplimiento de uno de los objetivos de este trabajo.

Aunque su uso ya apareció en algunos de los códigos de la sección anterior, en este apartado se va mostrar de manera muy breve su utilización en un ejemplo muy simple [47]. Se va a tratar de estimar las salidas correspondientes a una función sencilla ($y = 0.25x^2$), añadiendo además ruido a las muestras de entrenamiento. De igual forma, se va a analizar el efecto del parámetro de escala o *lengthscale*. El ejemplo se reduce a las líneas del Código 5.1.

Código 5.1 Código básico de ejemplo de GPy.

```
1  # Creación del kernel
2  kernel = GPy.kern.RBF(input_dim=1, lengthscale=.5)
3  # Creación del modelo a partir de los datos de entrenamiento
4  m = GPy.models.GPRegression(X,Y,kernel)
5  # Representación para lengthscale=1
6  print (m)
7  m.plot()
8
9  # Optimización
10 m.optimize()
11 # Optimización para lengthscale óptimo
12 print (m)
13 m.plot()
```

Para la primera simulación se va a usar un valor de factor de escala igual a 0.5. Es bien sabido en estos casos que conforme menor es el factor de escala, menor es también la suavidad a la hora de ajustar las muestras, como se puede apreciar en la comparación entre las Figuras 5.1 y 5.2. Un valor muy pequeño de factor de escala hará que los cambios en la función que ajusta las muestras sean muy bruscos obteniendo de esta manera una solución poco precisa. La misma situación pero a la inversa se puede apreciar en la gráfica obtenida para un valor de factor de escala muy grande, en la que los cambios en la función son tan suaves que difícilmente se llega a ajustar de manera adecuada a las muestras.

En esta situación es cuando cobra una gran importancia la optimización de hiperparámetros, ya que será necesario encontrar el valor de *lengthscale* que mejor se ajuste a estas muestras. Para ello, únicamente habrá que seguir la línea 10 en la que gracias al método *optimize* implementado en la librería GPy se consigue obtener unos valores adecuados de factor de escala y de varianza. Este método hace uso de la técnica de máxima verosimilitud para ajuste de hiperparámetros, la cual aparece en [29]. La aportación de la varianza es similar al del factor de escala pero a la inversa, ya que unos valores mayores harán que la función se ajuste a las muestras de una manera más exacta, y viceversa. En la Figura 5.3 se muestran los resultados para los valores óptimos de factor de escala y varianza, y se puede apreciar de manera clara como la estimación es sustancialmente mejor que en las anteriores figuras. Los valores óptimos obtenidos se muestran en la Tabla 5.1.

Tabla 5.1 Valores obtenidos tras optimización de hiperparámetros.

Parámetro	Valor óptimo
Factor de escala	3.344647559
Varianza	28.900546413

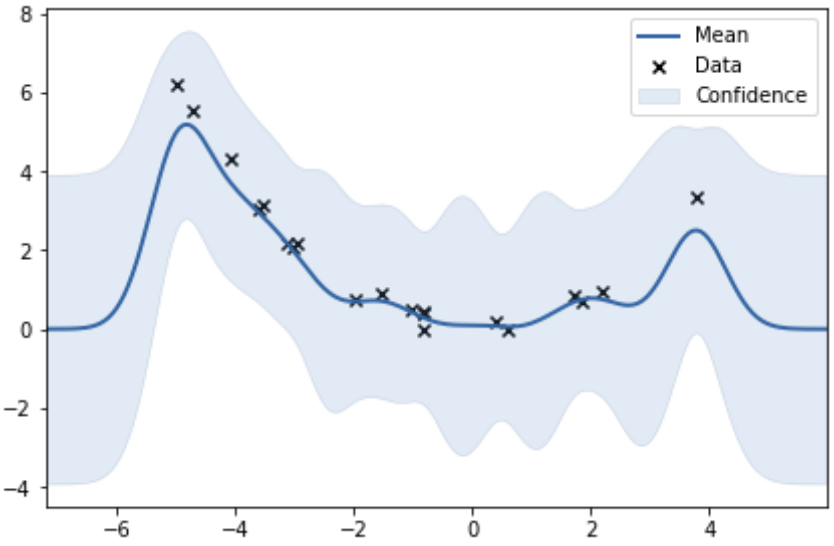

Figura 5.1 Simulación para factor de escala igual a 0.5.

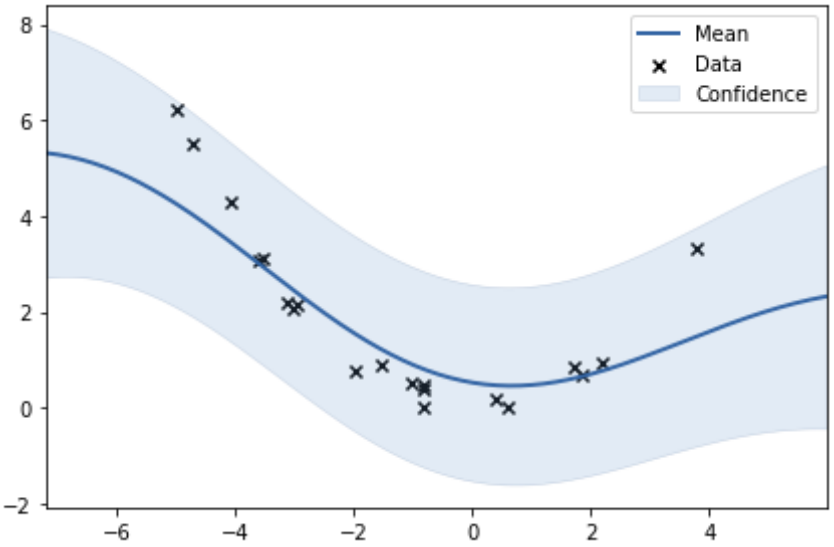

Figura 5.2 Simulación para factor de escala igual a 5.

5.1.2 Mejora de la librería GPy

Un aspecto interesante respecto al uso de esta librería dentro de este proyecto está relacionado con la implementación de la función *kernel* de la Ecuación (4.27). En la línea 2 del Código 5.1 se aprecia como la función *kernel* RBF se puede inicializar de manera muy simple usando uno de los métodos de la librería. En cambio, la implementación de los *kernels* NSE y LOG que aparecen en la Ecuación (4.27) no se podían llevar a cabo usando directamente los métodos de la librería ya que estos no venían implementados.

Por tanto, fue necesario iniciar un proceso de *ingeniería inversa* con el fin de investigar la forma en la que está desarrollado tanto el *kernel* RBF, así como otros *kernels* dentro de la librería. No fue un proceso especialmente largo ya que una vez encontrada la manera en la que estaban implementados estos *kernels*, únicamente fue necesario cambiar las fórmulas correctamente para calcular los resultados de matriz de covarianza a partir de las expresiones adecuadas. Además, y teniendo en cuenta que es necesario llevar a cabo una optimización del modelo, se añadió en el sitio correspondiente, la expresión de la derivada de cada uno de los *kernels* introducidos *a mano*, la cual es usada a lo largo del proceso de optimización.

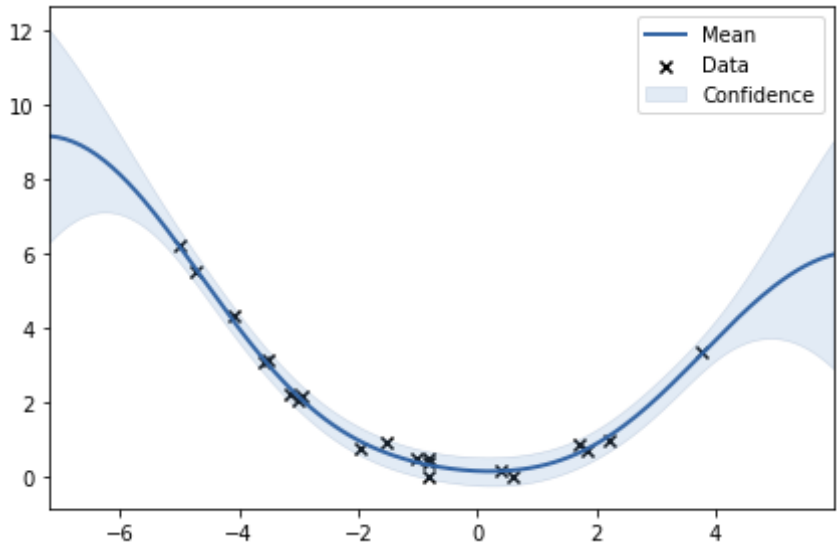

Figura 5.3 Simulación para factor de escala óptimo.

5.2 Entorno Virtual

Una buena práctica cuando se trabaja con un lenguaje de programación basado en librerías como puede ser *Python* es la creación de entornos virtuales, en los que se centraliza la instalación de librerías. La principal ventaja de esta práctica está relacionada con el control absoluto de las versiones de las librerías, pudiendo tener distintas versiones de las mismas en distintos proyectos realizados en paralelo. Además, se puede tener acceso en todo momento a las librerías instaladas en el entorno así como conocer su función de manera muy simple usando un *software* especializado en entornos virtuales [48].

Para este proyecto se ha usado el programa **ANACONDA** [49] en el que la instalación de librerías y activación del entorno se lleva a cabo de manera muy simple. Basta con acceder a la consola proporcionada por el programada e instalar las librerías usando el comando *conda install library* en dicha consola. Una vez instaladas las librerías necesarias para el proyecto, basta con acceder a la pestaña *environment* y seleccionar el entorno virtual deseado para activarlo como se aprecia en la Figura 5.4. Para este proyecto, el entorno virtual creado recibe el nombre de *myenv*. Como se aprecia en dicha Figura, existen otros entornos virtuales creados de manera totalmente aislada al de este trabajo, los cuales fueron creados para proyectos personales.

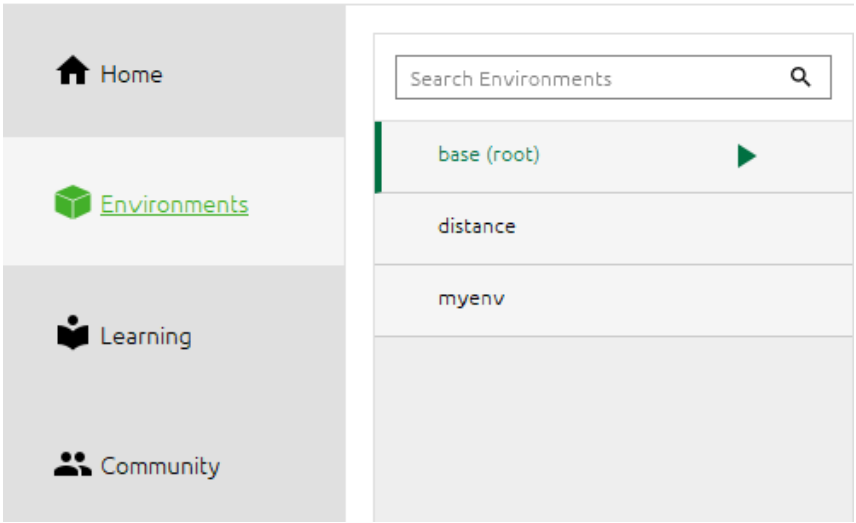

Figura 5.4 Selección del entorno virtual deseado.

Una de las principales ventajas de usar entornos virtuales está relacionada con la creación de nuevos en

tornos a partir de las versiones de las librerías instaladas en otro entorno. Para ello, se hace uso de un fichero llamado *requirements.txt* el cual puede ser usado para crear un nuevo entorno virtual a partir del siguiente comando: *conda create –name <env> –file <this file>*. En este fichero se recogen todas las librerías usadas en un determinado entorno virtual, así como la versión instalada de dicha librería. De esta forma, se puede reproducir perfectamente un entorno virtual nuevo a partir de uno previamente creado.

En este proyecto, el fichero *requirements.txt* donde se indican las librerías usadas así como sus versiones se desglosa en el Código 5.2. Se puede apreciar como el número de librerías que aparecen es considerablemente superior a las tratadas en este capítulo. Esto es debido a que, una librería puede estar soportada por otro librería que está a su vez soportada por otra librería, y así sucesivamente. Por tanto, cuando se instala una librería de cero, se instala a su vez todas las librerías que la soporta. Este es el motivo por el que aparece esta enorme cantidad de librerías.

Por último, y para conseguir reproducir con fidelidad el entorno virtual usado para este trabajo, bastará con crear un fichero *requirements.txt* usando la información que aparece en el Código 5.2, y posteriormente hacer uso del comando presentado con anterioridad.

Código 5.2 requirements.txt.

```
absl-py=0.9.0
alabaster=0.7.12
argh=0.26.2
astor=0.8.1
astroid=2.4.0
atomicwrites=1.3.0
attrs=19.3.0
autopep8=1.4.4
babel=2.8.0
backcall=0.1.0
bcrypt=3.1.7
bleach=3.1.4
ca-certificates=2020.1.1
cachetools=4.1.0
certifi=2020.4.5.1
cffi=1.14.0
chardet=3.0.4
cloudpickle=1.3.0
colorama=0.4.3
cryptography=2.9.2
cycler=0.10.0
dataclasses=0.7
decorator=4.4.2
defusedxml=0.6.0
docutils=0.16
entrypoints=0.3
flake8=3.7.9
future=0.18.2
gast=0.2.2
google-auth=1.14.1
google-auth-oauthlib=0.4.1
google-pasta=0.2.0
gpflow=1.5.1
gpy=1.9.9
grpcio=1.28.1
h5py=2.10.0
icu=58.2
idna=2.9
imagesize=1.2.0
```

```
importlib-metadata=1.6.0
importlib_metadata=1.5.0
intervaltree=3.0.2
ipykernel=5.1.4
ipython=7.13.0
ipython_genutils=0.2.0
isort=4.3.21
jedi=0.15.2
jinja2=2.11.2
jpeg=9b=hb83a4c4_2
jsonschema=3.2.0
jupyter_client=6.1.3
jupyter_core=4.6.3
keras-applications=1.0.8
keras-preprocessing=1.1.0
keyring=21.1.1
kiwisolver=1.2.0
lazy-object-proxy=1.4.3
libpng=1.6.37
libsodium=1.0.16
libspatialindex=1.9.3
markdown=3.2.1
markupsafe=1.1.1
matlabengineforpython=R2017b
matplotlib=3.2.1
mccabe=0.6.1
mistune=0.8.4
more-itertools=8.2.0
multipledispatch=0.6.0
nbconvert=5.6.1
nbformat=5.0.6
numpy=1.18.3
numpydoc=0.9.2
oauthlib=3.1.0
openssl=1.1.1g
opt-einsum=3.2.1
packaging=20.3
pandas=1.0.3
pandoc=2.2.3.2=0
pandocfilters=1.4.2
paramiko=2.7.1
paramz=0.9.5
parso=0.5.2
pathtools=0.1.2
pexpect=4.8.0
pickleshare=0.7.5
pip=20.0.2
pluggy=0.13.1
prompt-toolkit=3.0.4
prompt_toolkit=3.0.4
protobuf=3.11.3
psutil=5.7.0
py=1.8.1
pyasn1=0.4.8
pyasn1-modules=0.2.8
pycodestyle=2.5.0
pycparser=2.20
```

```
pydocstyle=4.0.1
pyflakes=2.1.1
pygments=2.6.1
pylint=2.5.0
pynacl=1.3.0
pyopenssl=19.1.0
pyparsing=2.4.7
pyqt=5.9.2
pyrsistent=0.16.0
pysocks=1.7.1
pytest=5.4.1
python=3.6.10
python-dateutil=2.8.1
python-jsonrpc-server=0.3.4
python-language-server=0.31.10
pytz=2019.3
pywin32=227
pywin32-ctypes=0.2.0
pyyaml=5.3.1
pyzmq=18.1.1
qdarkstyle=2.8.1
qt=5.9.7
qtawesome=0.7.0
qtconsole=4.7.3
qtpy=1.9.0
requests=2.23.0
requests-oauthlib=1.3.0
rope=0.17.0
rsa=4.0
rtree=0.9.4
scipy=1.4.1
setuptools=46.1.3
sip=4.19.8
six=1.14.0
snowballstemmer=2.0.0
sortedcontainers=2.1.0
sphinx=3.0.3
sphinxcontrib-applehelp=1.0.2
sphinxcontrib-devhelp=1.0.2
sphinxcontrib-htmlhelp=1.0.3
sphinxcontrib-jsmath=1.0.1
sphinxcontrib-qthelp=1.0.3
sphinxcontrib-serializinghtml=1.1.4
spyder=4.1.3
spyder-kernels=1.9.1
sqlite=3.31.1
tabulate=0.8.7
tensorboard=1.14.0
tensorflow=1.14.0
tensorflow-estimator=1.14.0
tensorflow-gpu=2.1.0
tensorflow-gpu-estimator=2.1.0
tensorflow-probability=0.9.0
termcolor=1.1.0
testpath=0.4.4
toml=0.10.0
tornado=6.0.4
```

```
154 traitlets=4.3.3
155 typed-ast=1.4.1
156 ujson=1.35
157 urllib3=1.25.9
158 vc=14.1
159 vs2015_runtime=14.16.27012
160 watchdog=0.10.2
161 wcwidth=0.1.9
162 webencodings=0.5.1
163 werkzeug=1.0.1
164 wheel=0.34.2
165 win_inet_pton=1.1.0
166 wincertstore=0.2
167 wrapt=1.12.1
168 yaml=0.1.7
169 yapf=0.28.0
170 zeromq=4.3.1
171 zipp=3.1.0
172 zlib=1.2.11
```

6 Código desarrollado

El aspecto más triste de la vida en este preciso momento es que la ciencia reúne el conocimiento más rápido de lo que la sociedad reúne la sabiduría.

Isaac Asimov

Una vez finalizado todo el análisis matemático en torno a este proyecto, tanto en este capítulo como en los siguientes se va a dejar de lado las demostraciones teóricas correspondientes al escenario, modelo o algoritmo, para estudiar el código desarrollado en este proyecto. Este código representa el principal esfuerzo invertido a lo largo de la realización de este trabajo, por lo que será necesario mostrar las consideraciones principales y los detalles más importantes con el fin de justificar el tiempo invertido en este apartado. Es importante mencionar que uno de los objetivos, a parte de conseguir entender toda la matemática detrás de este proyecto, consistía en trasladar el código realizado originalmente en *MATLAB* a otro tipo de lenguaje más propio del ML y que, además fuera de código abierto como *Python*. Por tanto, se trata igualmente de un capítulo con una alta relevancia dentro de esta memoria.

Es necesario realizar una serie de comentarios antes de entrar en detalle en el código ya que ayudan a entender la estructuración del código:

- El código está dividido en clases en las cuales se agrupan un porcentaje bastante alto del código completo. Se irá estudiando la estructuración y contenido de cada clase de manera individual.
- Un fichero principal orquestará las llamadas a cada una de las clases. Es importante en este punto tener en cuenta los argumentos de entrada y salida de cada uno de los métodos de cada clase. Se reservará también un espacio dentro de este punto para estudiar el algoritmo de programación desarrollado para orquestar el uso de las clases.
- Los resultados y las gráficas obtenidas gracias a este código serán comentadas en un capitulo posterior.

6.1 Presentación de las clases

Aunque estos apartados podrían ser considerados menos interesante, el autor de esta memoria considera muy importante su introducción ya que forman parte del esfuerzo realizado a la hora de llevar a cabo este trabajo. Es por ello que no se va a detallar en gran manera su contenido y únicamente se van a mencionar las variables y métodos principales dentro de cada clase.

6.1.1 Class *Flags*

Se trata de una clase llamada al comienzo del código correspondiente al *script* que orquesta las llamadas a las clases. En la llamada al constructor de esta clase se va a especificar el valor de las banderas que se quiere para cada uno de los supuestos planteados en este escenario. Esto es, el campo recibido en los sensores varía con el tiempo o es constante, o los sensores se encuentran fijos o desplazando su posición, entre otros.

En la Tabla 6.1, se muestran cada uno de los *flags* que forman parte del código.

En relación a la bandera correspondiente al escenario, se puede apreciar que tiene tres valores distintos, los cuales se describen a continuación:

Tabla 6.1 Banderas usadas en el código.

Bandera	Definición
scenario	Escenario usado (1, 2 ó 3).
flag_tx_est	Estimación de la posición del transmisor (1 ó 0).
flagGPstatic	Uso de algorimto GPR para escenario estático (1 ó 0).
flagGPrecursive	Uso de algoritmo GPR para escenario recursivo (1 ó 0).
flagOKD	Uso del algoritmo OKD. Algoritmo detallado en [39].
flagGholami	Uso de algoritmo "Gholami". Algoritmo detallado en [40].
flagWatcharapan	Uso de algoritmo "Watcharapan". Algoritmo detallado en [41].
flagfigs	Representación de las figuras (1 ó 0).
flagdisp	Representación de los hiperparámetros estimados del kernel (1 ó 0).
flagGridFix	Posición fija de los nodos (1 ó 0).
optimiseHyperparameters	Los hiperparámetros se obtienen del Kernel (True) o de los datos (False).
GPRegression	El modelo usado es GP para regresión (1 ó 0).
GPRegression_inducing	El modelo usado es GP para regresión con *inducing inputs* (1 ó 0).

1. Todos los sensores están estáticos.
2. Todos los sensores se están moviendo.
3. Todos los sensores están estáticos pero no están siempre transmitiendo.

Además, se observa que aparece un término nuevo relacionado con el algoritmo de GPR usado el cual recibe el nombre de *inducing inputs*. Se trata de una mejora del algoritmo de GPR que reduce el tiempo de computación y cuyas prestaciones y base teórica se comentarán en un capítulo posterior.

En función del valor de cada bandera, el escenario o el algoritmo usado puede variar obteniendo unos resultados diferentes en cada caso. De esta manera, se pueden simular distintos supuesto únicamente modificando el valor de la bandera y sin la necesidad de realizar grandes cambios en el código.

6.1.2 Class *Variables*

Al igual que ocurría para las banderas, existen una serie de variables o parámetros cuyos valores son constantes a lo largo de la ejecución completa del código. Por tanto, bastará con inicializarlos al comienzo del código. Estos parámetros corresponden a términos dentro del modelo cuyo valor no se estima sino que permanece fijo.

En la Tabla 6.2 se detallan cada una de estos parámetros.

El valor de la mayoría de estas constantes fueron especificados en los capítulos anteriores en los que se explicaba la procedencia e importancia de cada parámetros así como su valor. Se considera de gran importancia especificar correctamente el valor de cada variable a la hora de llamar al constructor de la clase ya que podrían inicializarse de manera errónea estos parámetros dando lugar a resultados no esperados.

6.1.3 Class *setup*

Esta clase da un giro respecto a la dinámica de las anteriores clases ya que no es utilizada para inicializar valores constantes o banderas que serán usadas a lo largo de la ejecución del código, sino que se trata de una clase preparada para inicializar el *setup* necesario para programa.

Cuando se habla de término *setup* se refiere a una serie de vectores en los cuales se van a almacenar los resultados obtenidos. Además, en esta clase se especifica el escenario de aplicación del algoritmo de GPR, esto es, la posición de los nodos donde se quiere calcular el campo recibido, o la posición inicial del transmisor en caso de no requerir de su estimación.

Para ello, cuenta con una serie de funciones que se van a ir llamando progresivamente a lo largo de la ejecución del programa principal, las cuales son descritas a continuación:

- **node_position**: Hace uso de la bandera *flagGridFix* para inicializar la posición de los nodos de manera fija o aleatoria.
- **tx_position**: Inicializa la posición del transmisor. Siempre se establece un valor inicial y en función de la bandera *flag_tx_est* se decide si se estima nuevamente la posición o se mantiene el valor establecido inicialmente. La posición del transmisor se elimina de entre las posibles posiciones de los nodos.

Tabla 6.2 Parámetros constantes a lo largo de la ejecución del código.

Variable	Descripción	Valor por defecto
shadowvarSim [dB]	Varianza del efecto shadowing en el exterior.	0:2:10
Nexp	Número total de experimentos.	10
Ttotal	Instantes de tiempo simulados.	10
P [dBm]	Potencia transmitida.	-10
U	Sensores disponibles entre el total de nodos.	10
alphai	Pérdidas por multitrayecto.	3.5
sigma_u	rho_u en [34].	200
Dcorr	Distancia de correlación.	50
varnoise	Varianza de ruido.	7
forgetting_factor	Factor de olvido. Un valor igual a cero olvida todo lo anterior mientras que con uno igual a uno todas las muestras aportan lo mismo.	0.5
rateoff	Porcentaje de los nodos totales que estarán apagados en cada instante de tiempo.	20
var_movement	Varianza entre instantes de tiempo consecutivos en el que se mueve los nodos.	1
noise_sigmau	Varianza correspondiente a la imprecisión en la posición de los sensores.	0

- **node_distribution**: Emplea la bandera *flag_moving* para determinar la ocupación de las posiciones de los nodos (especificadas en la función *node_position*) a lo largo de los instantes de tiempo.
- **initialize_vector**: Como se comentó previamente, es necesario generar una serie de vectores en los que se va a almacenar los resultados obtenidos a lo largo de la ejecución del código. Con el fin de optimizar al máximo el uso de memoria se hará uso de las banderas *flagGholami*, *flagWatcharapan*, *flagOKD*, *flagGPstatic* y *flagGPrecursive* para incializar los vectores únicamente cuando sea necesario.

Además, en la llamada al constructor de la clase será necesario especificar las características del *grid* donde se van a establecer la posición de los nodos. Por tanto, se especifican los siguientes términos: (1) Las dimensiones del *grid* las cuales son 500×500m en este proyecto, (2) la densidad de nodos dentro del *grid* (4 nodos por cada 100×100m) y (3) la distancia entre nodos, fijada en 100 metros.

6.1.4 Class *aditional_functions*

Tanto esta clase como la que se explica a continuación son las dos más importantes ya que incluyen las funciones más complejas así como la aplicación de la teoría a lenguaje de programación. Específicamente, entre este conjunto de funciones se encuentran aquellas que son utilizadas para la estimación de los parámetros requeridos en el modelo. Algunas de ellas serán tratadas sin entrar excesivamente en detalle en su contenido pero otras en cambio tendrán dedicadas una sección específica debido a su importancia en el proyecto.

Entre las funciones menos importantes se encuentran las listadas a continuación:

- **calculate_real_distance**: Se trata de una función diseñada para calcular la distancia entre las posiciones de los nodos o sensores. Como se puede comprobar en la Ecuacion (4.6) el parámetros distancia es uno de los especificados en el modelo de propagación.
- **calculate_noise_distance**: Es una función desarrollada de manera similar a la anterior pero dedicada exclusivamente a la posición de los sensores que varían su posición inicial de manera aleatoria con el fin de añadir incertidumbre a dicha posición.
- **readjust_node_emition**: El programa está preparado para simular un supuesto en el que los sensores están encendidos o apagados en función del instante de tiempo. Por tanto, será necesario especificar qué sensores están recibiendo señal y qué sensores permanecen apagados en cada instante de tiempo. Está función se encarga de determinar, en base al estado de la bandera *flag_intermitent*, los sensores que están activos en el instante de tiempo actual así como su posición.

En los siguientes apartados se pasará a estudiar de manera más detalla aquellas funciones con mayor relevancia y que además requieren de un mayor esfuerzo para su explicación.

Function *calculate_sample_shadow*

Aunque no se trate de una función encargada de estimar algún parámetro, es conveniente estudiar esta función de manera exhaustiva debido a su relación con la teoría. Es la función encargada de generar muestras que modelan la atenuación debido al fenómeno del *shadowing*, la cual sigue una distribución normal de media cero y varianza *shadowvar*. Estas muestras tienen que ser correladas de acuerdo al parámetro *Dcorr* por lo que son generadas a partir de una variable gaussiana multidimensional de media cero y matriz de covarianza obtenida a partir de la Ecuación (4.3). Para la generación de estas muestras se ha seguido el Código 6.1.

Código 6.1 Gneración de muestras para modelar atenuación debido al efecto de shadowing.

```
1  def calculate_sample_shadow(variables, Nall, Dall, kk):
2
3    # Media 0
4    m=np.zeros((Nall))
5
6    shadowvar=variables.shadowvarSim[kk]
7    shadowvar_nu=np.power(10,shadowvar/10)
8    shadowstd_nu=math.sqrt(shadowvar_nu)
9    shadowstd=10*np.log10(shadowstd_nu)
10   # En cada iteración se tiene que re-calcular este valor
11   # Parámetro: \sigma_v^{2}
12   var_v=shadowstd**2
13
14   # Cálculo de \rho_u
15   rho=np.exp(-Dall/variables.Dcorr)
16
17   # Ecuación (4) en paper
18   C=var_v*rho
19
20   # Generación de muestras por atenuación debido al efecto de shadowing
21   deltaall=np.random.multivariate_normal(m,C)
22
23   # Las muestras se distribuyen entre las salidas para las muestras de
24   # entrenamiento y las muestras de test
25   deltaM=deltaall[0:variables.M*variables.Ttotal]
26   deltaU=deltaall[variables.M*variables.Ttotal:Nall]
27
28   deltaM=np.reshape(deltaM,(variables.M,variables.Ttotal),order='F')
29
30   return deltaM, deltaU, shadowstd, var_v
```

Se puede observar como en la línea 18, la variable C es obtenida de manera equivalente a la matriz de covarianza de la Ecuación (4.3). La media es cero como se aprecia en la línea 4 y, además cabe destacar el cálculo de la variable *var_v* que corresponde con el parámetro σ_v^2 en la Ecuación (4.3). Este valor es calculado en cada iteración ya que los valores de varianza que modelan el efecto *shadowing* varían a lo largo del tiempo con el fin de llevar a cabo diferentes simulaciones para distintos valores de varianza. Esto último se detallará de manera más específica en el apartado de simulaciones realizadas. Un último aspecto a destacar corresponde a las muestras generadas ya que, como se aprecia en las líneas 25 y 26, son distribuidas entre los M nodos (*deltaM*) y los U sensores (*deltaU*). [1]

Function *estimate_tx_pos*

Para el código desarrollado en esta función se ha usado las fórmulas presentadas en el estudio realizado en la Sección 4.2.1. Aunque en esa sección se comentan también aspectos relacionados con el refinamiento llevado a cabo para mejorar aún más la estimación de la posición, no es hasta una función posterior cuando es calculado este refinamiento. Por tanto, en esta función únicamente se estima la posición del transmisor,

[1] La variable U corresponde a los sensores disponibles entre los N sensores totales.

dejando para más adelante la mejora de la estimación y quedando de esta forma un código más simple el cual se muestra en el Código 6.2.

Código 6.2 Estimación de la posición del transmisor.

```
1  def estimate_pos_tx(ZMon, posMxNoise, posMyNoise, posNxNoise, posNyNoise, t):
2    global w_told
3    global postx_est
4    # Pesos. Ecuación (12)
5    wi=10**(ZMon/10)
6    # Algoritmo recursivo para estimar la posición del transmisor
7    if t>0:
8      # Re-calculo de los pesos en cada instante de tiempo
9      # Ecuación (13)
10     w_t=w_told+np.sum(wi)
11     # Uso de los pesos para estimar la posición del transmisor
12     # Ecuación (11)
13     postx_est=np.append((np.sum(wi*posMxNoise)+w_told*postx_est[0])/w_t,
14              (np.sum(wi*posMyNoise)+w_told*postx_est[1])/w_t)
15     w_told=w_t
16   else:
17     postx_est=np.append(np.sum(wi*posMxNoise)/np.sum(wi),
18              np.sum(wi*posMyNoise)/np.sum(wi))
19     w_told=np.sum(wi)
20
21   # Se incluye el error en la localización del transmisor debido
22   # al ruido de la distancia de los senores y los "grid points"
23   dtxNoise=np.sqrt((postx_est[0]-posNxNoise)**2+
24   (postx_est[1]-posNyNoise)**2)
25
26   return dtxNoise, postx_est
```

Tras un análisis del código, se puede apreciar el algoritmo iterativo usado para estimar la posición del transmisor en el que se usa el peso calculado en el instante anterior como ya se explicó en la Sección 4.2.1 y como se observa en la línea 15. Otro aspecto a destacar se encuentra en la relación entre las líneas 5 y 10 y las Ecuaciones (4.13) y (4.14), respectivamente. Por último, la Ecuación (4.12) se desarrolla en las líneas 17 (para el primer instante de tiempo) y 13, dejando de esta forma constancia total entre el código desarrollado y la teoría estudiada. Como se ha comentado anteriormente, las ecuaciones correspondientes al refinamiento en la estimación serán estudiadas en la sección siguiente.

Function *alpha_beta_estimation*

Se trata de la función más importante dentro de este bloque ya que incluye tanto la estimación de los parámetros α y P del modelo de la Ecuación (4.6), como el refinamiento en el cálculo de la estimación del transmisor, cuyo nuevo valor es usado para mejorar de igual forma la estimación inicial de la potencia y del exponente de pérdidas.

Para esta función se ha necesitado de *técnicas* de programación un poco más complejas que hacen que la relación ecuación-código sea más difícil de asimilar, por lo que a diferencia de las funciones anteriores, no se va a mostrar la función completa para, posteriormente realizar un análisis de las líneas más trascendentales, sino que se va a ir más despacio.

Primeramente, se va a proceder al cálculo de la media de las distribuciones correspondientes a las estimaciones del exponente de pérdidas y potencia transmitida. A modo de referencia teórica se va a hacer uso de la Ecuación (4.23) en la que se se lleva a cabo una minimización de una expresión respecto a dos parámetros (μ_α y μ_P). Esta ecuación queda resumida en las líneas presentadas en el Código 6.3.

Código 6.3 Media de las muestras correspondientes a la estimación de α y P.

```
1  # Variable \hat{q} de la Ecuación (25)
2  q2use=10*np.log10(dtxNoise[0:variables.Mon])
3  # Variable \sqrt{\hat{D}}^{-1} en Ecuación (25)
4  errloc=1./dtxNoise[0:variables.Mon]
5
6  # Matriz correspondiente al primer sumando de la Ecuación (25)
7  Prev=np.append(np.ones(variables.Mon)/errloc, -q2use/errloc)
8  Prev2=np.reshape(Prev,(variables.Mon,2),order='F')
9  # Minimización correspondiente a la Ecuación (25)
10 params=lsq_linear(np.reshape(Prev,(variables.Mon,2),order='F'),
11 ZMon[0:variables.Mon]/errloc)
12
13 meanP=params.x[0]
14 meanalpha=params.x[1]
```

Las variables que se encuentran en las líneas 2 (*q2use*) y 4 (*errloc*) corresponden respectivamente a los parámetros $\hat{\mathbf{q}}$ y $\sqrt{\hat{\mathbf{D}}}^{-1}$ de la Ecuación (4.23). Por otro lado, y con el fin de calcular los valores que minimizan la función se hace uso del método *lsq_linear* correspondiente a la librería *scipy.optimize*. De acuerdo con la documentación de la librería [42], este método es usado para calcular el mínimo en la siguiente expresión: $\min_x ||Ax-b||^2$. Donde A es el primer argumento del método y b el segundo. Es por ello que el primer argumento corresponde con la variable *Prev*, la cual se trata de una matriz $Nx2$ donde la primera columna está formada por unos y la segunda por los valores calculados en *q2use*, todo ello dividido por la variable *errloc*. Además, el segundo argumento lo forman los valores de campo recibido en los sensores, los cuales están almacenados en la variable *ZMon* y corresponden al parámetro $\mathbf{z}$, dividido igualmente por la variable *errloc*. De esta forma, los valores mínimos son almacenados en la variable *params* correspondiendo el primer valor a la estimación de la potencia transmitida y el segundo al exponente de pérdidas.

Para el cálculo de la varianza de las distribuciones correspondientes a las estimaciones de α y P se va a proceder de la misma forma pero siguiendo la Ecuación (4.22). En este caso, se ha hecho uso del Código 6.4, el cual se explica a continuación.

Código 6.4 Varianza de la muestras correspondientes a la estimación de α y P.

```
1  # Variable \hat{\mu}_{z} en PAG 4
2  meanz=meanP-q2use*meanalpha
3  # Variable \Sigma_{z}
4  Sigmaz=(ZMon[0:variables.Mon]-meanz)[:,np.newaxis]@(ZMon[0:variables.Mon]-meanz
       )[np.newaxis,:]
5  Sigmazalpha=CNoise[0:variables.Mon,0:variables.Mon]+(variables.varnoise+
       variables.noise_varnoise)*np.eye(variables.Mon)+np.diag((variables.sigma_u+
       variables.noise_sigmau)**2/dtxNoise[0:variables.Mon]**2)
6
7  # Variable A en PAG 4
8  AAon=Sigmaz-Sigmazalpha
9  # Variable \hat{Q} en PAG 4
10 BBon=q2use[:,np.newaxis]@q2use[np.newaxis,:]
11
12 # Se obtienen las diagonales de las dos matrices anteriores
13 aaon=np.diag(AAon).flatten('F')
14 bbon=np.diag(BBon).flatten('F')
15
16 # Matriz correspondiente al primero sumando de la Ecuación (26)
17 Prev=np.append(np.ones(np.size(aaon)), bbon)
18 # Minimización correspondinete a la Ecuación (26)
19 params=lsq_linear(np.reshape(Prev,(np.size(aaon),2), order='F'),
```

```
20                  aaon, bounds=(0,np.inf))
21
22  varP=params.x[0]
23  varalpha=params.x[1]
```

Primeramente, los resultados almacenados en las líneas 2 y 4 corresponden a los parámetros μ_z y $\Sigma_{z|\alpha,P}$ de las Ecuaciones (4.19) y (4.18) respectivamente. Además, el valor de la variable *Sigmaz* corresponde al resultado de la operación: $\mathbf{\Sigma}_z = (\mathbf{z}-\hat{\boldsymbol{\mu}})(\mathbf{z}-\hat{\boldsymbol{\mu}})^T$. Por tanto, estos valores son usados para calcular las matrices de la Ecuación (4.22), a las cuales se les extrae posteriormente la diagonal almacenando los resultados en las variables *aaon* y *bbon* (líneas 13 y 14). Al igual que se hizo anteriormente, estas dos variables forman parte del primer y segundo argumento del método *lsq_linear*, mostrando especial interés en el primer argumento tratándose este de otra matriz de dimensiones $Nx2$ donde la primera columna está formada por unos y la segunda por el vector *bbon* obtenido a partir de la diagonal de *BBon* (variable que almacena la operación $\hat{\mathbf{q}}\hat{\mathbf{q}}^T$). El segundo argumento está compuesto por la diagonal de la matriz *AAon*, la cual está formada por los resultados de la operación $\mathbf{\Sigma}_z - \mathbf{\Sigma}_{z|\alpha,P}$. Como se demostró anteriormente, los resultados finales se extraen de la variable *params*.

Por último esta función tan completa incluye los aspectos relacionados con el refinamiento de la posición estimada del transmisor, presentado principalmente a partir de la Ecuación (4.15). Para este caso se ha hecho uso de las *funciones lambda* de *Python* en las cuales se definen funciones anónimas que dependen de una o varias variables y se va a proceder a minimizar esa función ya que únicamente depende de un parámetro (la posición del transmisor). El valor mínimo obtenido será la posición re-estimada del transmisor, la cual mejora a la primera estimación. Para eso se usan las líneas del Código 6.5 las cuales se explican más adelante.

Código 6.5 Refinamiento de la posición del transmisor.

```
1  rmean=np.sum(ZMon)/variables.Mon
2  betai=lambda ss, tt: np.log10(np.sqrt((ss-posMxNoise)**2+(tt-posMyNoise)**2))
3  betamean=lambda ss, tt: np.sum(betai(ss,tt))/variables.Mon
4
5  fun=lambda sstt: np.sum((betai(sstt[0],sstt[1])-betamean(sstt[0],sstt[1]))*
6  (ZMon-rmean))/np.sqrt(np.sum((betai(sstt[0],sstt[1])-
7  betamean(sstt[0], sstt[1]))**2)*np.sum((ZMon-rmean)**2))
8
9  x0=postx_est
10 # Minimización correspondiente a la Ecuación (27)
11 postx_est_prev=minimize(fun, x0)
12 postx_est=postx_est_prev.x
```

No merece la pena detenerse en este código ya que se aprecia claramente como en la línea 5 se define la función anónima la cual es minimizada en la línea 11. Los resultados finales de posición estimada del transmisor son extraídos en la línea 12.

6.1.5 Class *algoritmos*

En esta clase se estudian las dos funciones que engloban lo más relevante de este trabajo, ya que en ambas se implementan los algoritmos de GPR para estimación de campo recibido tanto para los casos en los que la información proporcionada por sensores corresponden a campo estático como a campo variante en el tiempo. Para la inicialización del *kernel* y para la optimización de sus hiperparámetros se han usado librerías existentes las cuales han sido descritos en un capítulo anterior. En esta sección únicamente se va a presentar la integración de los conceptos teóricos de GPR en un lenguaje de programación por lo que la base teórica para esta sección se encuentra en las Secciones 4.3.1 y 4.3.2.

La importancia detrás de esta sección está relacionada con el cumplimiento de uno de los objetivos planteados ya que en el capítulo correspondiente a los objetivos se mencionaba que uno de estos consistía en la asimilación y aplicación de un algoritmo de ML en un escenario *real*. Por tanto, y teniendo en cuenta que es necesario comentar también los aspectos restantes del escenario de aplicación, en esta sección confluyen los aspectos más interesantes desde el punto de vista de programación de un algoritmo de ML.

La dinámica seguida en estas funciones es siempre la misma, y se trata de uno de los procedimientos más usados a la hora de aplicar algoritmos de ML en escenarios reales. Normalmente, se parte de librerías las cuales reúnen los aspectos técnicos más complejos con el fin de liberar al programador de la gran carga que supone desarrollar los algoritmos. Por lo que gracias al uso de los métodos y clases implementadas en estas librerías, el esfuerzo final dedicado se reduce exponencialmente.

Debido a la importancia de estas dos funciones y a la cantidad de código implementado en ellas, se estudiarán por separado como se hizo en la sección anterior.

Function *algoritmo_GPStatic*

Primeramente, para esta sección se ha partido de la base teórica expuesta en la Sección 4.3.1 por lo que las ecuaciones que ahí aparecen serán usadas a lo largo de esta función. Se considera de importancia destacar, antes de entrar a valorar los aspectos más interesantes de esta función, la diferencia entre las muestras de entrenamiento y las de *test*. La variable *Mon* (que corresponde con el número M de nodos del *grid*), la cual es usada varias veces a lo largo de esta función, se utiliza para marcar el límite de las muestras que son usadas para entrenamiento dejando las restantes hasta llegar al total de las proporcionadas por los N sensores para *test*.

La preparación de algunas variables, las cuales son necesarias calcular antes de aplicar el algoritmo, se muestra en las líneas del Código 6.6.

Código 6.6 Preparación de variables aplicadas en GPR.

```
1  Prev=np.append(posNxNoise[0:variables.Mon], posNyNoise[0:variables.Mon])
2  # Entradas correspondientes a las muestras de entrenamiento
3  Xtrain=np.reshape(Prev, (variables.Mon, 2), order='F')
4  N_training=np.size(Xtrain,0)
5  # Salidas correspondientes a las muestras de entrenamiento
6  ytrain=ZMon
7  Prev=np.append(posNxNoise[variables.Mon:variables.N], posNyNoise[variables.Mon:
       variables.N])
8  # Entradas correspondientes a las muestras de test
9  Xtest=np.reshape(Prev, (variables.N-variables.Mon,2), order='F')
10
11 q=10*np.log10(dtxNoise[0:variables.Mon])
12 qd=10*np.log10(dtxNoise[variables.Mon:variables.N])
13 # Variable que trata de emular el kernel LOG y el término bias (\sigma_{P})
14 qq=varalpha*np.append(q,qd)[:,np.newaxis]@np.append(q,qd)[np.newaxis,:]+varP*np
       .ones([variables.N,variables.N])
15 Sigman=(variables.varnoise+variables.noise_varnoise)*np.eye(variables.Mon)+ np.
       diag((variables.sigma_u+variables.noise_sigmau)**2/dtxNoise[0:variables.Mon
       ]**2)
16
17 # Media de GP
18 mtrain=meanP-10*meanalpha*np.log10(dtxNoise[0:variables.Mon])
19 mtest=meanP-meanalpha*qd
```

Como entrada a la *función objetivo* se usa las posiciones imprecisas de los sensores las cuales son definidas a partir de su valor en el eje de abcisas y de ordenada. De ahí que la variable *Xtrain* de la línea 3 sea una matriz de dos columnas y *Mon* filas. Esta variable se usa para almacenar las entradas correspondientes a las muestras de entrenamiento y, de igual forma, modela el parámetro $\hat{\mathbf{X}}$ de las ecuaciones de la Sección 4.3.1. Además, la variable *ytrain* corresponde a las salidas de dicha función objetivo para las muestras de entrenamiento las cuales han sido calculadas anteriormente en otra parte del código y corresponde con el campo recibido en los sensores, almacenado en la variable *ZMon*[2]. Esta variable modela el parámetro $\mathbf{z}(t)$ en las ecuaciones de la ya mencionada sección. Por último, la variable *Xtest* de la línea 9 almacena las entradas pero en este caso correspondientes a las muestras de *test*, esto es, corresponden al parámetro $\mathbf{X}_g$ en las ecuaciones. El subíndice g indica que son posiciones que pertenecen al *grid*.

[2] El cálculo del campo recibido se lleva a cabo en el *script* principal y será comentado más adelante.

Por otro lado, las variables de las líneas 11 y 12 están relacionadas con los parámetros $\mathbf{q}$ y $\mathbf{q}_g$ respectivamente. Para el caso en el que no se optimizan los parámetros del *kernel*, y como se verá un poco más adelante, se hace uso de un *kernel* **RBF** más una serie de términos que tratan de modelar el *kernel* propuesto en la Ecuación (4.27). Estos términos que se suman a los resultados obtenidos a partir del kernel **RBF** son almacenados en la variable *qq*. Basta con hacer una breve observación para asociar estos términos de la Ecuación (4.27) con la variable *qq*.

La variable *Sigman* de la línea 15 modela las muestras de ruido correspondientes al modelo GP de la Ecuación (4.24) y, como ya explicó, estas muestras siguen una distribución cuya media es cero y cuya matriz de covarianza tiene la siguiente expresión: $\Sigma_n = \sigma_w \mathbf{I}_N + \rho_u^2 \hat{\mathbf{D}}(t)$.

Por último dentro de este primer bloque de líneas, se comenta las variables *mtrain* y *mtest* de las líneas 18 y 19. Estas variables corresponden con los parámetros $\mathbf{m}_{\mathbf{X}_g(t)}$ y $\mathbf{m}_{\hat{\mathbf{X}}(t)}$ de las Ecuaciones (4.30) y (4.25) respectivamente, esto es, con las funciones media de los procesos de GP tanto para las muestras de entrenamiento como para las muestras de *test*.

Por tanto, se dispone en este momento de una serie de variables las cuales han sido inicializadas a partir de los datos de campo recibido en las posiciones de las muestras de entrenamiento, y se quiere estimar la media y la matriz de covarianza de las muestras que modelan el campo estimado en ciertas posiciones, determinadas por las muestras de *test*. Para ello se hará uso de las líneas presentadas en el Código 6.7.

Código 6.7 Estimación de campo recibido en posiciones de *test* I.

```
1  if flags.optimiseHyperparameters==True:
2    # Creación del Kernel
3    bias=varP*np.ones([variables.N,variables.N])
4    NSEKern=GPy.kern.NSE(Xtrain.shape[1])
5    LOGKern=GPy.kern.LOG(Xtrain.shape[1])
6
7    # Suma de los dos kernels. Ecuación (42)
8    kern=NSEKern+LOGKern
9    beta=np.diag(1/((variables.sigma_u+variables.noise_sigmau)**2/dtxNoise**2+ (
         variables.varnoise+variables.noise_varnoise)*np.ones(variables.N)))
10
11   Prev=(ytrain-mtrain)[:,np.newaxis]@np.ones([1,2])
12
13   # Selección del modelo Gaussian Processes for Regression (GPR)
14   mR=GPy.models.GPRegression(Xtrain, Prev, kern)
15   # Optimización de los hiperparámetros
16   mR.optimize()
17
18   # Obtención del kernel una vez optimizado los hiperparámetros
19   kern=mR.kern
20
21   # K_{\hat{X} en Ecuación (37)
22   Ktrain=kern.K(Xtrain,Xtrain)+bias[0:N_training,0:N_training]+beta[0:
         N_training,0:N_training]
23   # K_{X_{g}, \hat{X}} en Ecuación (37)
24   Kx=kern.K(Xtest,Xtrain)+bias[N_training:,0:N_training]
25   # K_{X_{g}} en Ecuación (37)
26   Ktest=kern.K(Xtest,Xtest)+bias[N_training:,N_training:]
27   # Inversa de la Ecuación (41)
28   C_inv=np.linalg.inv(Ktrain+Sigman)
29
30   # La media de la predicción
31   mGP=mtest+Kx@C_inv@(ytrain-mtrain)
32   # La covarianza de la predicción
33   vGP=Ktest-Kx@C_inv@Kx.T
34
```

```
35      setup.RSSgpstatic=mGP
36
37      # Error cuadrático medio entre señal estimada y valor teórico
38      setup.mseGPRstatic[kk,t]=setup.mseGPRstatic[kk,t]+np.sum((setup.RSSgpstatic-
            ZUon)**2)
39    else:
40      # Kernel RBF para el caso en el que no se optimizan los hiperparámetros
41      kern=GPy.kern.RBF(input_dim=2,variance=shadowstd**2,
42      lengthscale=variables.Dcorr, inv_l=True)
43
44      # En este caso se añade la variable qq para tratar de simular lo
45      # máximo posible el kernel de la Ecuación (42)
46      Ktrain=kern.K(Xtrain,Xtrain)+qq[0:N_training,0:N_training]
47      Kx=kern.K(Xtest,Xtrain)+qq[N_training:,0:N_training]
48      Ktest=kern.K(Xtest,Xtest)+qq[N_training:,N_training:]
49      C_inv=np.linalg.inv(Ktrain+Sigman)
50
51      # La media de la predicción
52      mGP=mtest+Kx@C_inv@(ytrain-mtrain)
53      # La covarianza de la predicción
54      vGP=Ktest-Kx@C_inv@Kx.T
55
56      setup.RSSgpstatic=mGP
57
58      # Error cuadrático medio entre señal estimada y valor teórico
59      setup.mseGPstatic[kk,t]=setup.mseGPstatic[kk,t]+np.sum((setup.RSSgpstatic-
            ZUon)**2)
```

Se puede apreciar como el código se divide en dos partes bien diferenciadas, la primera de ella correspondiente a las líneas 1-38 y la segunda a las líneas 40-59. En el primer bloque se lleva a cabo la estimación del campo recibido realizando previamente una **optimización de los hiperparámetros del *kernel***.

Pero primeramente, se realiza la definición del *kernel* en las líneas 3,4 y 5 en las que se inicializa el término *bias* y los *kernel* **NSE** y **LOG** respectivamente. Cada término corresponde con cada sumando de la Ecuación (4.27) y, gracias al uso de la librería *GPy*, la cual se presentó en un capítulo anterior, basta con usar el operando $+$ para definir la función *kernel* por completo.

Posteriormente, destaca las líneas 14 y 16 donde nuevamente la librería *GPy* es usada para definir el modelo, el cual como se ha comentado a lo largo de esta memoria no es otro que un proceso gaussiano para regresión (GPR), y se inicializa con las entradas correspondientes a las muestras de entrenamiento y con las salidas una vez restada la media. Una de las líneas más importantes es la 16, en la que se aprecia como el modelo se optimiza con el fin de obtener aquellos hiperparámetros que proporcionen mejores resultados en la función *kernel*.

Nuevamente, las líneas 21-28 corresponden a la inicialización de variables las cuales son sumamente importante en este proyecto. Con el fin de conseguir un mejor entendimiento de esta parte tan importante, se ha rellenado la Tabla 6.3 en la que se realiza una asociación entre las variables definidas en estas líneas y los parámetros de la Ecuación (4.32).

Tabla 6.3 Asociación parámetro-variable en Ecuación (4.32).

Variable	Parámetro	Argumentos de entrada
Ktrain	$\mathbf{K}_{\hat{\mathbf{X}}}$	Muestras de entrenamiento
Kx	$\mathbf{K}_{\mathbf{X}_g,\hat{\mathbf{X}}}$	Muestras de entrenamiento y test
Ktest	$\mathbf{K}_{\mathbf{X}_g}$	Muestras de test

Para inicializar estas variables se hace uso de la función *kernel* una vez optimizado los hiperparámetros. Los argumentos de entrada de esa función son las posiciones correspondientes a las muestras de entrenamiento o *test* según corresponda. A los resultados proporcionados como salidas de la función se les suma el término *bias* el cual no fue introducido en la optimización del *kernel* y por tanto se añade en este punto y además,

a la variable *Ktrain* se le suma la variable *beta* que incluye el ruido de las muestras como se aprecia en la Ecuación (4.32). Adicionalmente, se encuentra la variable *C_inv* que está asociada con el parámetro $\mathbf{C}_{\hat{\mathbf{x}}}^{-1}$ (Ecuación (4.36)).

Todas estas variables definen perfectamente la función densidad de probabilidad conjunta necesaria en todo proceso gaussiano para, una vez condicionada dicha función densidad de probabilidad a las salidas (*ytrain* o $\mathbf{z}(t)$) se pueda obtener la media y la matriz de covarianza de las muestras que modelan el campo estimado. En este punto se aprecia claramente como las líneas 31 y 33 corresponden perfectamente con las Ecuaciones (4.34) y (4.35), calculando de esta forma la media y la matriz de covarianza de las muestras respectivamente.

Por último, en la línea 38 se calcula el error cuadrático medio cuya finalidad será estudiada en el capítulo de simulaciones realizadas.

Para el caso en el que no se optimizan los hiperparámetros el código es mucho más simple ya que únicamente se define un una función *kernel* **RBF** la cual no es optimizada y que además es usada para calcular directamente las variable *Ktrain*, *Kx* y *Ktest* en las líneas 46-49. Adicionalmente, se le suma la variable *qq* con el fin de simular de la manera más fiel posible el *kernek* de la Ecuación (4.27). Igualmente se estima la media y la matriz de covarianza de las muestras en las líneas 52 y 54 a partir de las variables anteriores y de las ecuaciones ya mencionadas.

Function *algoritmo_GPRecursive*

Una vez estudiado el código para el caso en el que campo es estático se va a proceder a presentar el código utilizado para estimación en el supuesto en el que el campo recibido en los sensores sea variante en el tiempo. Las diferencias desde el punto de vista teórico respecto al caso en el que el campo recibido en los sensores es estático han sido estudiadas en un capítulo anterior por lo que aquí únicamente se presentará la aplicación de las fórmulas mostrando especial interés en la introducción del factor de olvido.

Las líneas correspondientes a la estimación en este supuesto se encuentran en el Código 6.8 en el que únicamente se incluyen las líneas correspondientes a la estimación sin optimización de hiperparámetros con el fin de evitar cargar en exceso el código introducido en esta memoria.

Código 6.8 Estimación de campo recibido en posiciones de *test* II.

```
1  global mprior
2  global Cprior
3  Prev=np.append(posNxNoise[0:variables.Mon], posNyNoise[0:variables.Mon])
4  # Entradas correspondientes a las muestras de entrenamiento
5  Xtrain=np.reshape(Prev, (variables.Mon, 2), order='F')
6  N_training=np.size(Xtrain,0)
7  # Salidas correspondientes a las muestras de entrenamiento
8  ytrain=ZMon
9  Prev=np.append(posNxNoise[variables.Mon:variables.N], posNyNoise[variables.Mon:
       variables.N])
10 # Entradas correspondientes a las muestras de entrenamiento
11 Xtest=np.reshape(Prev, (variables.N-variables.Mon,2), order='F')
12
13 q=10*np.log10(dtxNoise[0:variables.Mon])
14 qd=10*np.log10(dtxNoise[variables.Mon:variables.N])
15 # Variable que trata de emular el kernel LOG y el término bias (\sigma_{P})
16 qq=varalpha*np.append(q,qd)[:,np.newaxis]@np.append(q,qd)[np.newaxis,:]+varP*np
       .ones([variables.N,variables.N])
17 Sigman=(variables.varnoise+variables.noise_varnoise)*np.eye(variables.Mon)+ np.
       diag((variables.sigma_u+variables.noise_sigmau)**2/dtxNoise[0:variables.Mon
       ]**2)
18
19 # Media de GP
20 mtrain=meanP-10*meanalpha*np.log10(dtxNoise[0:variables.Mon])
21 mtest=meanP-meanalpha*qd
22
23 # Kernel RBF para el caso en el que no se optimizan los hiperparámetros
```

```
24 kern=GPy.kern.RBF(input_dim=2,variance=shadowstd**2,
25 lengthscale=variables.Dcorr, inv_l=True)
26
27 # En este caso se añade la variable qq para tratar de simular lo
28 # máximo posible el kernel de la Ecuación (42)
29 # K_{\hat{X}} en Ecuación (37)
30 Ktrain=kern.K(Xtrain,Xtrain)+qq[0:N_training,0:N_training]
31 # K_{X_{g}, \hat{X}} en Ecuación (37)
32 Kx=kern.K(Xtest,Xtrain)+qq[N_training:,0:N_training]
33 # K_{X_{g}} en Ecuación (37)
34 Ktest=kern.K(Xtest,Xtest)+qq[N_training:,N_training:]
35 # Inversa de la Ecuación (41)
36 C_inv=np.linalg.inv(Ktrain+Sigman)
37
38 # \mu_{post}. Ecuación (57)
39 mpos=Kx@C_inv@(ytrain-mtrain)
40 # \Sigma_{post}. Ecuación (58)
41 Cpos=Kx@C_inv@Kx.T
42
43 # "prior" in los nodos
44 if t==0 or (flags.flag_moving==0 and flags.flag_intermitent==0):
45   mprior=mpos
46   Cprior=Cpos
47
48 # La media de la predicción
49 mGP=mtest+variables.forgetting_factor*mpos+(1-variables.forgetting_factor)*
       mprior
50 # La covarianza de la predicción
51 vGP=Ktest-variables.forgetting_factor*Cpos-(1-variables.forgetting_factor)*
       Cprior
52
53 setup.RSSgprecursive=mGP
54
55 #\mu_{prior}. Ecuación (59)
56 mprior=variables.forgetting_factor*mprior+(1-variables.forgetting_factor)*mpos
57 #\Sigma_{prior}. Ecuación (60)
58 Cprior=variables.forgetting_factor*Cprior+(1-variables.forgetting_factor)*Cpos
59
60 # Error cuadrático medio entre señal estimada y valor teórico
61 setup.mseGPrecursive[kk,t]=setup.mseGPrecursive[kk,t]+np.sum((setup.
       RSSgprecursive-ZUon)**2)
```

Observando las líneas expuestas, se aprecia como la preparación de variables del Código 6.6 son exactamente iguales que las líneas 3-21. Además, la definición de la función *kernel*, así como el calculo de las variables de la Tabla 6.3 se realiza en las líneas 30-36 de manera similar a como se hacia en el caso estático.

Las novedades en este fragmento de código empiezan a aparecer en la línea 39 y 41, donde se calcula la media y la matriz de covarianza para la función densidad de probabilidad *a posteriori* usando las expresiones de las Ecuaciones (4.39) y (4.40) respectivamente. Por otro lado, para el (1) instante cero, o (2) para el caso en el que los sensores desplacen su posición en cada instante de tiempo, o (3) para el supuesto en el que los sensores permanezcan apagados en algún instante de tiempo, la media y matriz de covarianza *a priori* coincide con los valores *a posteriori* calculados en las líneas mencionadas anteriormente. Esta consideración se muestra en las líneas 44-46.

Una vez inicializadas estas variables, la media y matriz de covarianza de las muestras que modelan el campo estimado se calcula en las líneas 49 y 51 haciendo uso de la base teórica de las Ecuaciones (4.37) y (4.38). En estas líneas, y como ocurría en las ecuaciones, aparece el factor de olvido o *forgetting factor* para dar una mayor relevancia en la estimación a la muestras recientes frente a las más antiguas.

Por último, se calcula la media y matriz de covarianza *a priori* para la siguiente iteración en las líneas 56 y 58 haciendo uso de las Ecuaciones (4.41) y (4.42).

6.1.6 Class *gráficas*

Esta última clase tiene una importancia considerablemente inferior a sus dos inmediatas predecesoras ya que únicamente cuenta con tres métodos los cuales son usados para representar los resultados obtenidos. Los detalles más importantes de estos métodos están relacionados con el uso de la librería *matplotlib*, por lo que se considera innecesario entrar demasiado en detalle.

Los tres métodos que forman esta clase se listan a continuación:

- **representacion_scenario**: Es usada para representar una gráfica muy interesante del escenario de aplicación del algoritmo ya que presenta la posición del transmisor así como de los sensores y nodos donde se pretende estimar el campo recibido.
- **representacion_variacion_potencia**: Se trata de un método usado para representar el campo recibido en función de la distancia. Para los resultados de RSS que se quieren representar se consideran las medidas de los sensores, los valores estimados y los valores teóricos.
- **representacion_final_algoritmo**: Se encarga de representar el error cuadrático medio entre la estimación realizada y el valor teórico real frente a distintos valores de varianza de las muestras que modelan la atenuación por efectos de *shadowing* (σ_v^2).

6.2 Fichero principal

Todas y cada una de las clases presentadas en la anterior sección son inicializadas meticulosamente y los métodos desarrollados en cada una de ellas son llamados de manera orquestada a partir de un fichero principal. En torno al 90 % está condensado en las clases presentadas en la sección previa mientras que el 10 % del código restante, el cual se presenta en esta sección, se encarga de gestionar las llamadas a las clases y métodos de manera ordenada.

Este 10 % correspondiente al fichero principal cuenta con la siguiente estructura: Destaca la presencia de tres bucles que hacen que el tiempo de ejecución del algoritmo sea bastante extenso. En los dos bucles siguientes al primero se itera en torno a los instante de tiempo y los distintos valores de varianza de *shadowing*. Por defecto, se lleva a cabo una simulación de 10 instantes de tiempo, con un segundo por iteración mientras que los valores de varianza que modela las pérdidas por *shadowing* están acotados entre 0.5 y 10, iterando cada dos valores. El primer bucle está implementado con el fin de realizar un promedio de los resultados obtenidos. Más adelante, en el capítulo correspondiente a las simulaciones realizadas, se profundizará más en la importancia de este promediado.

Como se vio en las funciones correspondientes a la Clase *setup*, la posición de los nodos del *grid* se configuraban haciendo uso de funciones definidas en esa clase. En cambio, las posiciones de los sensores dentro de ese *grid* es determinada en el fichero principal. Además, como se comentó en el capítulo correspondiente a la presentación del escenario, es necesario añadir una indeterminación a la posición de los sensores con el fin de dar un mayor realismo al modelo, ya que esta no puede ser perfectamente conocida. Este calculo de error se realiza igualmente en el fichero principal.

Por otro lado, la acción más importante a realizar dentro del fichero principal se recoge en las líneas mostradas en el Código 6.9.

Código 6.9 Calculo de campo recibido en sensores.

```
1 # Path-loss exponent
2 alpha=np.ones(variables.N)*variables.alphai
3
4 # Shadowing
5 delta=np.append(deltaM[indexon,t],deltaU)
6
7 # Ruido
8 noise=np.append(np.sqrt(variables.varnoise)*np.random.randn(variables.Mon),np.
      zeros(variables.U))
```

```
9
10  # RSSI
11  Z=variables.P-np.diag(10*alpha*np.log10(dtx))@np.ones(variables.N)+delta+noise
12  ZMon=Z[0:variables.Mon]
13  ZUon=Z[variables.Mon:variables.N]
```

En la línea 8 se observa claramente la implementación en fórmula de la Ecuación (4.6) en la que se aprecia las muestras que modelan la atenuación por *shadowing* así como el ruido gaussiano que modela el ruido propio de las muestras así como la imprecisión en la posición de los sensores. Estas muestras quedan almacenadas en las variables *delta* y *noise* respectivamente. Los resultados obtenidos a partir de la línea 8 tienen principalmente dos funciones:

1. Las primeras *Mon* muestras son tomadas como si fueran información proporcionada por sensores y se almacena en la variable *ZMon*. Esta variable es usada principalmente como salida de la función objetivo en las posiciones de entrenamiento por lo que se asocia al parámetro $\mathbf{z}(t)$ en las Ecuaciones.
2. Las muestras restantes son tomadas como campo recibido de manera teórica en las posiciones de los nodos. Por tanto el valor estimado a partir del algoritmo de GPR será comparado con los valores almacenados en esta variable con el fin de calcular el error cuadrático medio y de esta forma obtener un parámetro que mide la precisión de la estimación realizada.

6.3 Código desarrollado para *sparse* GP

Como ya fue comentado en un capítulo anterior, en este trabajo se hace uso de una técnica que pretende reducir el tiempo de ejecución total de este código. La base teórica detrás de esta técnica se presenta en la Sección 3.4. Por tanto, y siguiendo con la idea de este capítulo, únicamente se mostrarán las líneas de código necesarias para hacer uso de esta técnica.

La librería *GPy*, ya explicada con anterioridad en este documento, permite de manera muy simple determinar el modelo de GP requerido para cada escenario. Como se veía en el Código 6.7 en la línea 12, la elección del modelo usado se lleva a cabo de manera trivial a partir del método *GPRegression*.

Para usar el modelo *sparse* GP con el fin de reducir el tiempo de ejecución total, basta con hacer el cambio mostrado en el Código 6.10.

Código 6.10 Selección del modelo *sparse* GP.

```
1  num_inputs=150
2  mC=GPy.models.SparseGPRegression(Xtrain, Prev, kern, num_inducing=num_inputs)
3  mC.optimize
```

En dicho código se aprecia como únicamente es necesario hacer uso del método *SparseGPRegression* para seleccionar el modelo con reducción de muestras en lugar del modelo de regresión convencional. Además, se observa como en la línea 1 se indica el número de muestras que van a "codificar" la información correspondiente al conjunto total de muestras. Ni que decir tiene que una reducción de este valor implica un decremento en el tiempo total de ejecución pero en cambio, la estimación realizada es menos precisa.

7 Simulaciones y resultados obtenidos

Equipado con sus cinco sentidos, el hombre explora el universo que lo rodea y a sus aventuras las llama Ciencia.

EDWIN POWELL HUBBLE

La principal finalidad de todo código desarrollado en un proyecto reside en la obtención de resultados, usados con el fin de llevar a cabo una serie de conclusiones. Por tanto, definir perfectamente las simulaciones que se van a realizar es uno de los pasos más importantes dentro de cualquier trabajo ya que determinarán conclusiones finales del estudio realizado.

A lo largo de este capítulo se van a presentar las simulaciones que se han realizado a partir del código desarrollado, así como los resultados obtenidos. Además, se mencionarán algunas de las conclusiones técnicas principales obtenidas a partir de un análisis de los resultados obtenidos gracias a las simulaciones realizadas. Las conclusiones finales del proyecto se dejarán para un capítulo posterior.

7.1 Robustez de valores estimados

Uno de los aspectos más interesantes tratados a lo largo de la memoria está relacionado con la estimación de aquellos parámetros del modelo de los cuales no se conocían de manera exacta su valor. Tras una explicación extensa acerca del contexto teórico en el que se mueven estas estimaciones, así como una presentación detallada del código implementado para poder llevar a cabo dicha estimación, en esta sección se van a mostrar algunos resultados correspondientes a la exactitud de estos parámetros.

Primeramente se van a mostrar los resultados de una serie de simulaciones correspondientes a distintas estimaciones de la posición del transmisor. Como se ha comentado en la sección anterior, el *grid* en el que se mueve el escenario de este trabajo está determinado por una superficie de 500x500m. Además, y como se aprecia en la Figura 4.1, los ejes están definidos de forma que se establece la posición cero en el medio y por tanto, las posibles posiciones van desde el -250 hasta el 250 tanto para el eje de abscisas como para el de ordenadas. Por tanto, se podrá determinar la posición del transmisor en cualquier lugar dentro de este *grid*, siempre y cuando se tenga en cuenta las consideraciones acerca de los ejes mencionadas anteriormente.

Tabla 7.1 Estimación de posición del transmisor para distintas casos.

	$x_0^{[1]}$	$\hat{x}_0^{[1]}$	$x_0^{[2]}$	$\hat{x}_0^{[2]}$	$x_0^{[3]}$	$\hat{x}_0^{[3]}$	$x_0^{[4]}$	$\hat{x}_0^{[4]}$
Pos x	0	-5.19	-30	-43.56	130.6	104.36	-180.4	-224.67
Posy	0	7.34	60	64.07	130.6	107.38	-180.4	-203.43

Como se aprecia en la Tabla 7.1, se han realizado una serie de cuatro simulaciones (indicadas en el superíndice de los elementos de la tabla) en las que se ha variado drásticamente la posición del transmisor a lo largo de la totalidad de la superficie del escenario. Se puede observar como los resultados obtenidos no son especialmente exactos, existiendo en algunos casos un error de en torno al 10 %, pero teniendo en cuenta todas las fuentes de error añadidas al modelo, se puede concluir que estos resultados son aceptables ya que no estima de manera exacta la posición del transmisor, pero ayuda a dar una idea aproximada de su posición.

Por otro lado, se realizaron otras cuatro simulaciones pero para distintos valores de *potencia transmitida* y de **exponente de pérdidas**. La principal finalidad de estas simulaciones es similar a la anterior, ya que se pretende mostrar una idea de la exactitud y robustez de las estimaciones de estos dos parámetros. Las estimación correspondiente a estas dos variables se muestra en la Tabla 7.2 para cada una de las cuatro simulaciones realizadas.

Tabla 7.2 Estimación de potencia transmitida y exponente de pérdidas para distintas simulaciones.

	$P_{tx}^{[1]}(dBm)$	$\alpha^{[1]}$	$P_{tx}^{[2]}(dBm)$	$\alpha^{[2]}$	$P_{tx}^{[3]}(dBm)$	$\alpha^{[3]}$	$P_{tx}^{[4]}(dBm)$	$\alpha^{[4]}$
Valor teórico	-15	3.6	-10	3.5	-5	3.4	0	3.3
Valor estimado	-18.19	3.50	-12.19	3.41	-6.89	3.38	0.33	2.95

A la vista de los resultados se puede concluir que las estimaciones realizadas para la potencia transmitida y para el exponente de pérdidas son considerablemente mejores que para la posición del transmisor. Cabe destacar que las estimaciones del parámetro α son especialmente exactas, exceptuando para la última simulación en la que el error cometido es considerable. En un gráfica correspondiente a un apartado posterior se mostrará desde otro punto de vista la fiabilidad de estas estimaciones.

7.2 Estimación de campo recibido

En esta sección se muestran los resultados más importantes de este trabajo, ya que se van a presentar las simulaciones realizadas, así como los resultados obtenidos a partir de estas, en las que se muestra la precisión del modelo a la hora de estimar el campo recibido en los nodos del *grid*. Al igual que para la estimación de parámetros, la explicación de teórica así como el código desarrollado para llevar a cabo estas simulaciones fueron presentadas en los capítulos anteriores, por lo que está sección va a estar enfocada en la introducción de las simulaciones realizadas y los resultados obtenidos. También se apartará un espacio en esta sección para presentar la manera de obtener cada una de las figuras, esto es, se comentarán las banderas que son necesarias de inicializar para cada una de las simulaciones realizadas. El resto de parámetros permanecen constantes en todas las simulaciones.

7.2.1 Campo estático

En este primer apartado se va a partir del supuesto en el que el campo es constante a lo largo del tiempo, y por tanto únicamente será necesario tener en cuenta la muestra actual para poder realizar las estimaciones del campo recibido.

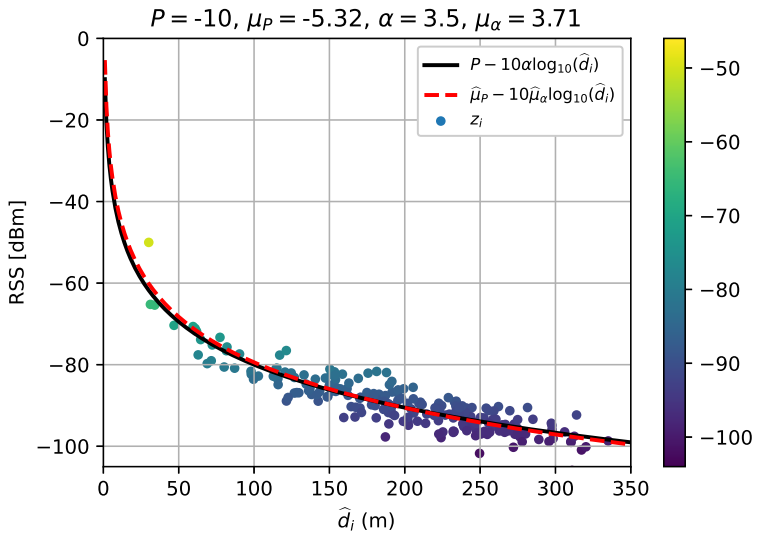

Figura 7.1 Variación de campo recibido respecto a la distancia.

Una de las simulaciones más interesantes realizadas en este apartado se muestra en la Figura 7.1. A partir del análisis de esta figura se pueden desarrollar algunas conclusiones importantes:

- Los puntos corresponden a la información, proporcionada por los sensores, de campo recibido en ciertas posiciones del espacio. Si se analiza con detenimiento se puede concluir que se tratan de los mismos puntos de la Figura 4.1 pero distribuidos de otra forma.
- La línea negra corresponde con el campo recibido en función de la distancia para el caso "teórico". Esto es, para el caso en el que la potencia del transmisor y el exponente de pérdidas no son estimados sino que se parte de sus valores originales.
- La línea roja a rayas corresponde a la variación del campo recibido en función de la distancia para el caso en el que los parámetros son estimados. Como ya se apreció en la sección anterior, las estimaciones son considerablemente aceptables, hasta el punto de obtener resultados muy próximos a los teóricos.
- Un aspecto interesante en esta simulación corresponde a la estimación de campo recibido en posiciones próximas al transmisor. Como ya se ha comentado, la información proporcionada por los sensores en estos puntos es muy imprecisa como ya se demostró a raíz de la Figura 4.2. Gracias a las mejoras introducidas, las cuales ya fueron explicadas con anterioridad en esta memoria, la información proporcionada por estos sensores tienen un peso menor que la del resto, por lo que la estimación en esas posiciones cercanas siguen siendo buenas a pesar de ese error.

En esta situación de campo estático, se llevaron a cabo además algunas simulaciones adicionales. La primera de ellas, da lugar a una gráfica en la que se muestra la media de la distribución *a posteriori* estimada a partir del Proceso gaussiano. La idea detrás de esta gráfica reside en el estudio del campo recibido a lo largo de todas las posiciones del escenario. Se podría haber representado de manera discreta, únicamente a partir de datos aislados, pero tiene un mayor sentido dibujar una malla en la que se muestra como el campo recibido varía a lo largo de todas las posiciones del escenario. La representación de esta importante simulación se muestra en la Figura 7.2.

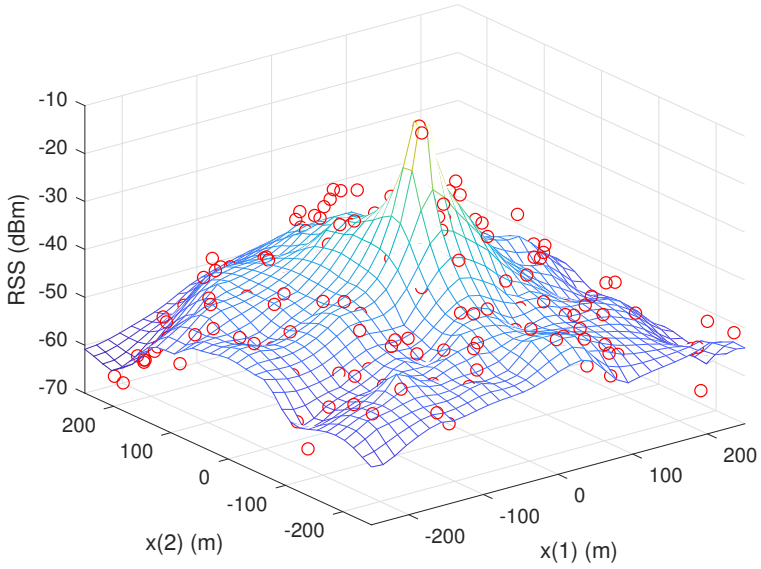

Figura 7.2 Campo recibido a lo largo de todo el escenario ($x_0 = [0\ 0]$).

En esta figura se representa además con puntos rojos el campo recibido en los sensores, y se aprecia de manera clara como esta información afecta determinantemente al campo estimado. Por ejemplo, en la esquina superior izquierda (valores pequeños del eje x, y grandes del eje y), se aprecia una hendidura en la superficie debido a una acumulación de medidas en torno a esa zona. Se puede observar también, como la superficie representada trata de "unir" y "juntar" todos los puntos rojos, obteniendo de esta forma los resultados esperados. Por tanto, se ve claramente la **regresión** llevada a cabo entre los puntos con la información de campo recibido de los sensores.

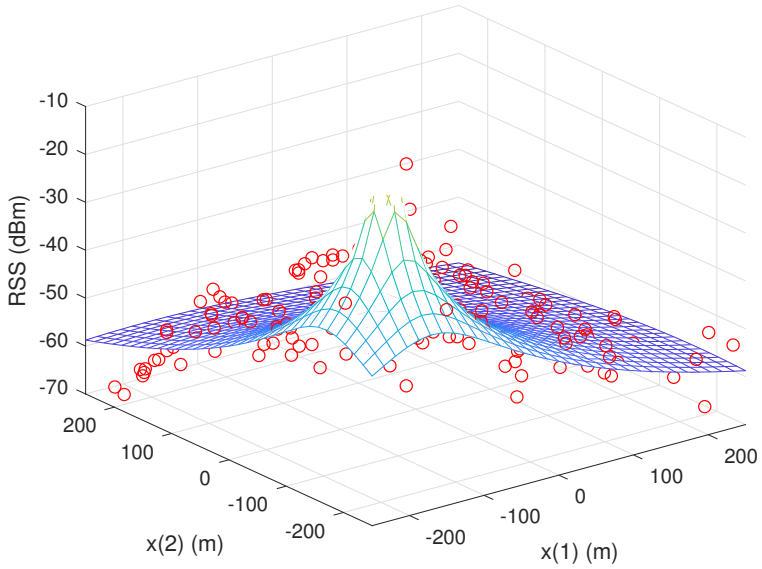

Figura 7.3 Campo recibido a lo largo de todo el escenario ($x_0 = [-150 \ -150]$).

La posición del transmisor podría cambiar como se muestra en la Figura 7.3, pero la dinámica sigue siendo la misma, ya que la superficie va decayendo desde la posición del transmisor hasta las posiciones más alejadas, como se estudió a raíz de la Figura 7.1.

Por último, la última simulación llevada a cabo en este apartado en el que el campo es estático a lo largo del tiempo, está relacionada con el cálculo del error en la estimación. Para esta simulación, se va a hacer uso del modelo de la Ecuación (4.6) con el fin de calcular el valor teórico de campo recibido en la posiciones del *grid*, usando el valor original de potencia transmitida y de exponente de pérdidas. Este valor teórico será comparado con el valor estimado a partir de la siguiente ecuación:

$$MSE = \frac{1}{M}\sum_{i}(\mu_{g_i}(t) - f_{g_i}(t))^2 \tag{7.1}$$

Para tener una gráfica interesante, se ha representado el error para distintos valores de varianza de *shadowing*. La gráfica obtenida tras un promediado de 10 experimentos se muestra en la Figura 7.4.

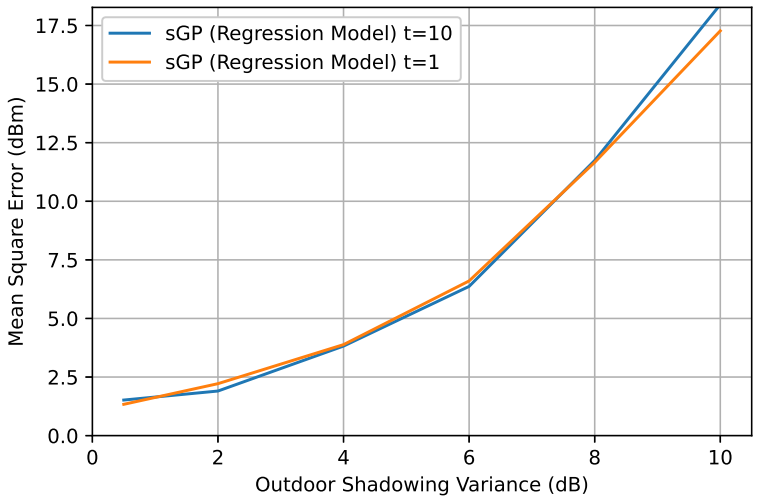

Figura 7.4 Error cuadrático medio del campo estimado para distintos valores de varianza de *shadowing*.

En dicha gráfica se aprecia el comportamiento esperado, ya que el error cuadrático medio crece a la vez que la varianza ya que el ruido introducido en el modelo es cada vez más grande. Además, se ha representado la curva para los instantes de tiempo primero y último, obteniendo unos resultados muy similares (salvo para valores muy grandes de varianza). Este comportamiento también era esperado ya que en este apartado el campo recibido en los sensores es constante a lo largo del tiempo.

Obtención de figuras

Como se comentó en la introducción de la sección, se va a apartar un espacio para indicar los valores de las banderas y las llamadas a las clases presentadas en un capítulo anterior que fueron usadas para obtener cada una de las figuras de cada sección.

- De manera general, los valores de bandera fueron los presentados en la Tabla 7.3.

Tabla 7.3 Valores de bandera para simulación en situación de campo estático.

Bandera	Valor de simulación
scenario	1
flag_tx_est	1
flagGPstatic	1
flagGPrecursive	0
flagOKD	0
flagGholami	0
flagWatcharapan	0
flagfigs	1
flagdisp	1
flagGridFix	1
optimiseHyperparameter	True
GPRegression	1
GPRegresssion_inducing	0

- La Figura 7.1 únicamente se representa para el primer valor de instante de tiempo y varianza de *shadowing*. Para su representación, se hace uso del método *representacion_variacion_potencia* correspondiente a la clase *graficas*.
- Para obtener las Figuras 7.2 y 7.3 es necesario primeramente estimar el campo recibido. Para ello, se hace uso del método *algoritmo_GPstatic* de la clase *algoritmos*. Previamente, se tuvo que estimar la posición del transmisor y de los parámetros α y P_{tx} con las funciones *estimate_tx_pos* y *alpha_P_estimation* respectivamente, y generar las muestras que modelan las pérdidas por *shadowing* con *calculate_sample_shadow*. Todas estos métodos pertenecen a la clase *aditional_functions*. Para su representación, y teniendo en cuenta que no se encontró ninguna librería en *Python* para representar mallas de forma aceptable, se importaron los datos de campo estimado a *MATLAB* y se representaron las gráficas a partir de comandos pertenecientes a este *software*.
- Por último, para la Figura 7.4 se calculó primeramente el error cuadrático medio a partir del método *algoritmo_GPstatic* y se hizo uso después del método *representacion_final_algoritmo* para su representación.
- Todas estas acciones se resumen en un *script* el cual se adjunta como anexo a esta memoria: *static_GP.py*.

7.2.2 Campo dinámico

Aunque las simulaciones más interesantes en esta sección corresponden al apartado en el que el campo recibido es estático a lo largo del tiempo, se considera igualmente necesario mostrar los resultados correspondientes a las simulaciones en las que el campo recibido varía a lo largo del tiempo. Por tanto, en este apartado se va a mostrar el error cuadrático medio que se obtiene al comparar el valor estimado con el valor teórico de campo recibido. Se va a mostrar de esta forma una gráfica parecida a la que se muestra en la Figura 7.4 pero con algunos matices que se comentarán a continuación.

Para esta simulación se va a partir de los mismos valores de varianza de *shadowing* que para el caso anterior, con el fin de obtener una gráfica muy similar. Además, los resultados obtenidos se van a comparar con las simulaciones realizadas usando la técnica OKD (*ordinary kriging with detrending*) [39], para de esta forma mostrar la mejora que supone usar la técnica propuesta en este trabajo.

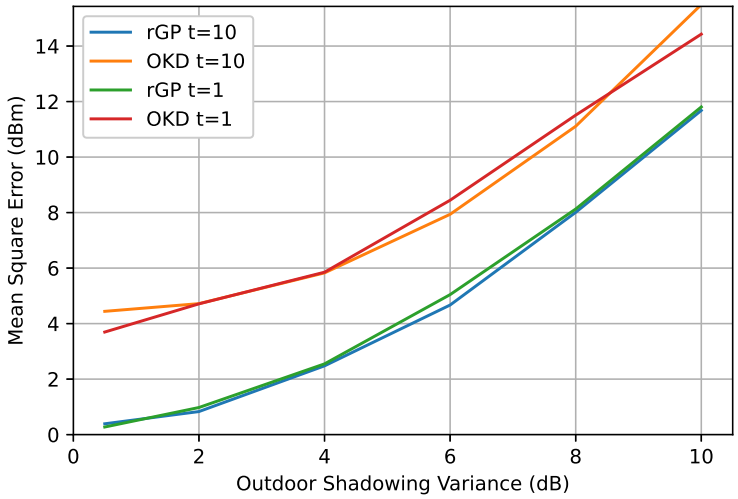

Figura 7.5 Error cuadrático medio del campo estimado para distintos valores de varianza de *shadowing* y campo variante en el tiempo.

En la gráfica mostrada en la Figura 7.5 se puede apreciar de manera clara la mejora con respecto a la técnica OKD ya que para todos los valores de varianza, el error cuadrático medio obtenido tiene un valor más pequeño. Por tanto, se puede concluir que la técnica propuesta en este trabajo proporciona resultados más precisos.

Por otro lado, y en relación a los instantes de tiempo, se puede observar como incluso para el último instante de tiempo, los valores de error cuadrático medio obtenido permanecen aproximadamente iguales, lo que incita a pensar que la técnica basada en el factor de olvido proporciona resultados muy fiables, por lo que el estudio del error cuadrático medio se puede llevar a cabo independientemente al instante de tiempo incluso cuando el campo recibido varía a lo largo del tiempo.

Obtención de figuras

Las figuras correspondientes a esta sección fueron obtenidas teniendo en cuenta las siguientes consideraciones:

- Los valores de las banderas son los mismos que los mostrados en la Tabla 7.3, pero cambiando el valor de *flagGPstatic* a cero y el de *flagGPrecursive* a uno.
- El resto de llamadas que fueron empleadas para estimar el campo recibido en una situación variante en el tiempo fueron exactamente iguales.
- Para llevar a cabo las estimaciones a partir de la técnica OKD [39] se usaron las funciones previamente creadas en *MATLAB*, las cuales fueron llamadas desde el *script* de *Python*.
- Teniendo en cuenta que sólo fue necesario cambiar el valor de algunas banderas con el fin de obtener las figuras de esta sección, el *script* empleado en este caso no se añade como adjunto en el anexo, pero recibe el nombre de *dinamic_GP.py*

7.2.3 *Sparse GP*

En este apartado se van a mostrar los resultados obtenidos a partir de las simulaciones correspondientes a la técnica de proceso gaussiano con *inducing inputs*.

En la Figura 7.6 se aprecia claramente como los resultados obtenidos con esta técnica son notablemente peores respecto a aquellos obtenidos a partir del modelo de regresión convencional. Aunque para valores muy pequeños de varianza de *shadowing* las estimaciones realizadas tienen un error inferior, conforme va

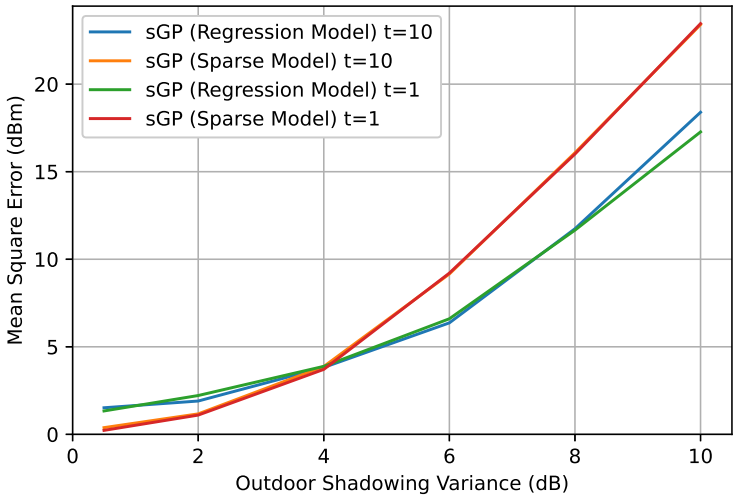

Figura 7.6 Comparación de resultados respecto a la simulación con *inducing inputs*.

aumentando dicha varianza, se incrementa igualmente el error entre el valor estimado y el valor teórico. Además, únicamente para valores de varianza menores a 4dB, el error en la estimación se encuentra por debajo respecto del modelo de regresión convencional. Este error en la estimación se podría achacar a una selección de número de muestras muy pequeña, pero no es el caso ya que el número de muestras seleccionado para este modelo fue de 150 (como se aprecia en la línea 1 del Código 6.10) frente a las 218 muestras del modelo de regresión original. Por tanto, se puede concluir que incluso para valores de número de muestras próximos al original, la estimación realizada con este nuevo modelo no es especialmente precisa.

Esto se podría deber entre otras cosas a las características del escenario en el que se aplica este algoritmo. Este tipo de técnicas son fuertemente dependientes del escenario de aplicación por lo que se podría pensar que estos errores en la estimación son debido a las características intrínsecas al escenario. Como línea futura se podría (1) buscar otro tipo de *kernel* o se podría (2) estudiar alguna forma de cambiar el escenario de forma que se adapte mejor a este nuevo modelo. Esta segunda variante no correspondería con un caso real ya que en un escenario práctico, es el algoritmo el que tiene que adaptarse al escenario y no viceversa, pero las simulaciones podrían arrojar resultados interesantes desde el punto de vista académico.

Por otro lado, las ventajas de esta técnica aparecen cuando se comparan los tiempos de ejecución de ambos modelos. Para el modelo con *sparse* GP, el tiempo total de ejecución del algoritmo fue de 5619.058 segundos (1 hora, 33 minutos y 39 segundos) para un promediado de 10 experimentos. En cambio, para el mismo número de simulaciones, la duración total para el caso en el que se usa el modelo de regresión habitual fue de 6517.57 segundos (1 horas, 48 minutos y 37 segundos), lo que supuso un ahorro de en torno a 15 minutos (13.79 %). Un ahorro de menos del 15 % puede parecer insignificante, pero cuando se habla de tiempos tan grandes como suceden en algunas simulaciones de este trabajo, el tiempo total de ahorro podría rondar las horas.

Por último, y tras un breve estudio de esta técnica se puede sacar la conclusión de que se trata de un algoritmo a partir del cual se obtienen realmente beneficios desde el punto de vista de los tiempos de ejecución de las simulaciones (y más todavía cuando se usan lenguajes de programación no orientados al cálculo matricial como *Python*), pero que, en base al escenario de aplicación del algoritmo, se podrían llegar a obtener estimaciones menos precisas como ocurre en este trabajo y como además se mostró en la Figura 7.6.

Obtención de figuras

Las figuras correspondientes a esta sección fueron obtenidas teniendo en cuenta las siguientes consideraciones:

- Los valores de las banderas son los mismos que los mostrados en la Tabla 7.3, pero cambiando el valor de *GPRegression_inducing* a uno.
- El resto de llamadas a métodos de clases son exactamente iguales que para el caso estático sin *inducing inputs*

- Al igual que para el caso dinámico, el *script* usado en esta sección no se muestra en el anexo pero recibe el nombre de *spase_GP.py*

7.3 Variación de la potencia transmitida a lo largo del tiempo

La base teórica correspondiente a este apartado se encuentra en la Sección 4.4. Por tanto, las ecuaciones presentadas en dicha función fueron añadidas en el código con el fin de obtener los márgenes entre los que se van a mover las estimaciones. En este apartado las simulaciones van a estar centradas en torno a un único nodo, estimando el campo recibido en ese nodo de manera individual. En estas circunstancias, se va a calcular el valor teórico en ese nodo, el estimado, y los márgenes. Los resultados obtenidos se muestran en la Figura 7.7.

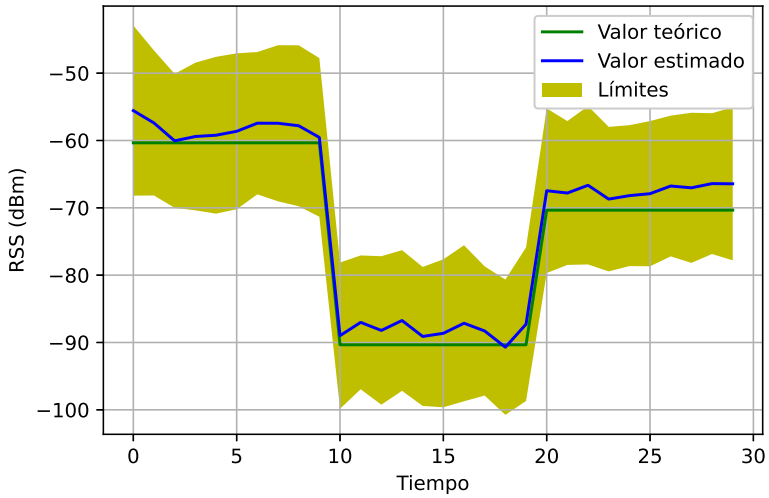

Figura 7.7 Variación de la potencia transmitida a lo largo del tiempo.

En la figura se aprecia claramente como las estimaciones siguen siendo igualmente precisas aún en un entorno en el que la potencia del transmisor varía a lo largo del tiempo. Para esta simulación se ha optado por cambiar la potencia transmitida cada 10 segundos siendo esta de 30 dBm, 0 dBm y 20 dBm respectivamente a lo largo del tiempo. Se puede observar igualmente como tanto el valor teórico como el estimado se encuentran en todo momento dentro de los márgenes establecidos por el método de HCRB, por lo que se puede concluir que las estimaciones son realizadas con éxito en estas condiciones.

8 Conclusiones y trabajos futuros

En el fondo, los científicos somos gente con suerte: podemos jugar a lo que queramos durante toda la vida.

Lee Smolin

El contenido completo desde el punto de vista teórico, así como el código usado además de los resultados obtenidos a partir del mismo han sido desarrollados a lo largo de los capítulos anteriores, por lo que en este último capítulo de la memoria se van a presentar las conclusiones obtenidas tras la realización de este proyecto así como los posibles trabajos futuros que pudieran surgir posterior a este estudio, los cuales ayudarían a seguir avanzando en este complejo campo y además proporcionaría una mayor riqueza a este estudio.

Por tanto, este breve capítulo estará dividido en dos partes bien diferenciadas: (1) la primera de ellas corresponderá con las conclusiones obtenidas y la (2) segunda con los posibles trabajos futuos. Cada una de estas partes serán desarrolladas en los apartados siguientes.

8.1 Conclusiones

Como en todo trabajo, proyecto o estudio, se trata de una buena práctica guardar un espacio al final de la memoria en el que se recogen una serie de conclusiones obtenidas tras la realización del proyecto. Esta práctica ayuda a sintetizar el trabajo realizado para, que en un futuro cuando se desee continuar con este trabajo, se disponga de un estado de partida conocido que evite duplicar el trabajo realizado, ahorrando de esta forma un tiempo considerable. Además, ayuda a comprobar si se han alcanzado los objetivos fijados al comienzo del trabajo, marcando los resultados como éxito o fracaso, y dejando como trabajos futuros aquellos objetivos que no se hayan alcanzado.

En este trabajo, y si se retorna la lectura al capítulo donde se presentaba los objetivos, fueron fijados dos tipos de objetivos bien diferenciados:

1. Convertir el código implementado en *MATLAB* en otro lenguaje de código abierto como es *Python*.
2. Comprender las posibles aplicaciones del *Machine Learning* en un escenario real.
3. Incluir mejoras de cara a reducir la carga computacional.

En cada uno de las secciones posteriores se profundizará un poco más en cada uno de estos puntos.

8.1.1 Conversión del Código

Se trata del objetivo fijado desde el punto técnico y el cual consistía en alcanzar los resultados obtenidos en [34], los cuales fueron conseguidos gracias a librerías y código desarrollado en *MATLAB*, pero a partir de librerías y código en *Python*.

No hace falta acceder a dicha referencia ya que en el capítulo correspondiente al código desarrollado se mostró perfectamente como las fórmulas presentadas en el estudio teórico fueron usadas para implementar las línea del código en *Python*. Por tanto se recomienda encarecidamente repasar el contenido de este capítulo con el fin de comprobar la implementación de las fórmulas de este proyecto a lenguaje de programación.

De esta manera, y a la vista de los resultados obtenidos, se puede concluir que este objetivo fue alcanzado satisfactoriamente.

8.1.2 Aprendizaje ML

Se trata del objetivo fijado desde el punto de vista personal. Se considera importante volver a recordar en este punto que, todo el marco teórico que rodea a este trabajo, parte de un estudio realizado anteriormente y en el cual, el autor de esta memoria no tuvo ninguna implicación. La base teórica de la que partía el realizador de este trabajo, en cuanto a aspectos relacionados con el ML era nula y, por tanto, haber realizado este trabajo desde cero hubiera supuesto un esfuerzo el cual se encuentra muy por encima de lo requerido como *Trabajo Fin de Máster*.

Por otro lado, se trata de un objetivo cuyo cumplimiento es difícilmente demostrable de manera empírica, ya que no hay ninguna forma de medir el nivel de conocimientos adquiridos de forma numérica. Por tanto, este apartado tiene un enfoque más personal que técnico.

De manera sincera, y aunque no haya forma de demostrar las competencias adquiridas, puedo decir que el aprendizaje adquirido gracias a este trabajo es muy considerable. Como se ha comentado, la base era nula y tras unos meses invertidos en la comprensión de los fundamentos de GPR aplicados en este escenario, se pudo satisfactoriamente y de manera autónoma con ayuda del tutor de este trabajo, obtener resultados fiables. Por lo que se puede concluir que este objetivo fue alcanzado ya que, a partir de este momento, los conocimientos generales sobre los algoritmos de ML, y más específicamente sobre los algoritmos de GP para regresión, han sido adquiridos de manera satisfactoria.

De esta forma se alcanza uno de los objetivo personales (ajenos a este trabajo) que andaba buscando el cual consistía en adquirir conocimientos sobre ML de manera que pudieran ser aplicados en un escenario real.

8.2 Trabajos futuros

Otra de las buenas prácticas que rodean a todo trabajo está basada en la fijación clara de los objetivos, esto es, en delimitar el alcance del proyecto de forma que no deje lugar ningún tipo de dudas. De esta forma, se pueden sacar conclusiones relacionadas con el éxito o fracaso del proyecto, al cumplir o no los objetivos como ya fue explicado en la sección anterior. También evita realizar trabajo adicional ya que, al delimitar el alcance del proyecto de manera clara, las actividades a realizar así como el trabajo total a desarrollar quedan perfectamente acotados.

En cambio, el hacer un esfuerzo por delimitar el alcance del trabajo de manera clara, no implica que no se pueda "ir más allá", y que el estudio realizado se acabe al finalizar la memoria del proyecto. Podrían darse el caso de posibles estudios o actividades futuras relacionadas con el trabajo que trasciendan los límites y alcance del proyecto, pero que pueda llegar a proporcionar resultados interesantes. Al ser necesario que el proyecto tenga un cierre bien definido, este tipo de supuestos se suelen especificar en el apartado de trabajos futuros y por tanto, en esta sección se presentarán algunos de los trabajos futuros que pudieran surgir de este trabajo.

Al no encontrarse especialmente relacionados los objetivos de este trabajo con el estudio teórico, es difícil de encontrar, por mi parte, vías interesantes de estudio que puedan surgir a partir de este proyecto. De todas formas, existen algunos supuestos trabajos futuros que pueden ser interesantes los cuales son listados a continuación, y están relacionados con el transmisor:

1. El primero de ellos está relacionado con detectar la existencia o ausencia de transmisor. En este trabajo se da el supuesto de que siempre existe un transmisor que además se encuentra en una zona fija. Por tanto, sería un avance detectar si existe o no transmisor antes de estimar el campo recibido, lo que ahorraría tiempo de computación.
2. Otro aspecto interesante estaría relacionado con la existencia de más de un transmisor lo que implicaría un cambio considerable del modelo ya que sería necesario tener en cuenta los efectos de emisores adicionales que interfieren entre ellos. Por tanto, sería necesario añadir al modelo original las interferencias entre transmisores una vez han sido detectados y localizados correctamente.

Apéndice A

Código correspondiente al *script* usado para obtener las figuras de la Sección 7.2

Código A.1 *static_GP.py*.

```
# Autor: Daniel Vela Calderón
# Trabajo Fin de Máster

# Código Python correspondiente al paper:
# "Recursive Estimation of Dynamic RSS Fields Based on
# Crowdsourcing and Gaussian Processes"

# Script principal encargado de llamar a las distintas funciones que forman
# este proyecto

# Librerías importadas
import numpy as np
import math
import time
import random
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.optimize import lsq_linear
import GPy
import matlab.engine
from flags import Flags
from variable import Variables
from setup import Setup
import aditional_functions as functions
import graficas as gr
import algoritmos as alg
#import gpflow

# Herramienta para ejecturar funciones de Matlab
eng=matlab.engine.start_matlab()
eng.addpath(r'C:\Users\Dani Vela\Desktop\TFM\Codes\Matlab\FinalCode', nargout
    =0)


#%%

```

```
# Sentencia necesario para medir el tiempo de ejecución de las simulaciones
expend=time.time()

'''
Los distintos flags son almacenados en la siguiente clase:
    1. Escenario usado. Hay tres posibles escenarios:
        1.1. Todos los sensores están estáticos (1)
        1.2. Todos los sensores se están moviendo (2)
        1.3. Todos los sensores están estáticos pero no están siempre
            transmitiendo (3)
    2. Estimación de la posición del transmisor (1 ó0)
    3. Uso de algorimto GPR para escenario estático (1 ó0)
    4. Uso de algoritmo GPR para escenario recursivo (1 ó0)
    5. Uso del algoritmo OKD. Mirar referencia del paper (1 ó0)
    6. Uso de algoritmo "Gholami". Mirar referencia del paper (1 ó0)
    7. Uso de algoritmo "Watcharapan". Mirar referencia del paper (1 ó0)
    8. Representación de las figuras (1 ó0)
    9. Representación de los hiperparámetros estimados del kernel (1 ó0)
    10. Posición fija de los nodos (1 ó0)
    11. Los hiperparámetros se obtienen del Kernel (True) o de los datos (False
        )
    12. El modelo usado es GP para regresión (1 ó0)
    13. El modelo usado es GP para regresión con inducing inputs (1 ó0)
'''
flags=Flags(1, 0, 1, 0, 0, 0, 0, 1, 0, 1, True, 1, 0)
flags.flag_mov_inter()

# Establecemos una semilla para controlar la aleatoriedad de las muestras
np.random.seed(1)

'''
Las variables que han sido obtenidas en distintos estudios son almacenados en
la siguien clase, además de las variables necesarias para la ejecución del
programa:
    1. Varianza del efecto "shadowing" en el exterior en dB
    2. Número total de experimentos
    3. Instantes de tiempo simulado
    4. Potencia transmitida (dBm)
    5. Número de nodos no disponibles
    6. Pérdidas por multitrayecto
    7. rho_u en el paper (mdB)
    8. Distancia de decorrelación
    9. Varianza de ruido
    10. "Forgetting Factor". Lambda = 0, olvida todo lo anterior. Lambda = 1
    todas las muestras anteriores aportan lo mismo
    11. Porcentaje de los nodos totales que estarán off en cada instante de
        tiempo
    12. Varianza entre instantes de tiempo consecutivos en el que se mueve los
        nodos
    13. noise_sigmau
    14. noise_varnoise
'''
variables=Variables([0.5, 2, 4, 6, 8, 10], 10, 10, -10, 10, 3.5, 200, 50,
                    7, 0.5, 20, 1, 0, 0)

'''
Clase para preparar el setup necesario para este trabajo:
```

```
    1. Grid donde se encuentran los nodos (500mx500m)
    *La densidad de nodos de observación es de 4 por cada 100mx100m
    2. Densidad de nodos dentro del grid
    3. Distancia entre nodos
'''
setup=Setup([500, 500], 0.2, 100)

# Posición de los nodos
setup.node_position(flags, variables)

# Posición del transmisor
# Situación del transmisor en el centro del grid
postx=[0, 0]
setup.tx_position(postx, variables)

# Distribución de los nodos disponibles entre el total de nodos del Grid
setup.node_distribution(flags, variables)

# A continuación se inicializa los vectores donde se almacenará el error
# cuadrático medio para cada algoritmo
setup.initialize_vectors(flags, variables)

# Bucle principal
for nexperiments in range(0,variables.Nexp): # Nexp
    print ('Nexp=' + str(nexperiments) + '/' + str(variables.Nexp-1))

    # Para cada experimento, genera los datos. Vector con la posición de los
    # sensores y de todos los grid
    posNx=np.append(setup.posMxall[:,:,nexperiments].flatten('F'), setup.posUx)
    posNy=np.append(setup.posMyall[:,:,nexperiments].flatten('F'), setup.posUy)

    # Calculo de las distancias real
    Nall=len(posNx)
    Dall=functions.calculate_real_distance(Nall, posNx, posNy)

    #%%
    for kk in range(0,len(variables.shadowvarSim)): #len(shadowvarSim)
        print ('ShadowvarSim=' + str(kk) + '/' + str(len(variables.shadowvarSim)
            -1))
        # Genera las muestras para la atenuación debido al fenómeno de
        # "shadowing" la cual sigue una distribución normal de media cero y
        # varianza shadowvar. Estas muestras tienen que ser correladas por
        # por lo que son generadas con una variable gaussiana multidimensional
        # de media cero y covarianza c
        deltaM, deltaU, shadowstd, var_v = functions.calculate_sample_shadow(
            variables, Nall, Dall, kk)

    #%%
        for t in range(0,variables.Ttotal): #Ttotal
            print ('Instante=' + str(t) + '/' + str(variables.Ttotal-1))

            # En función del instante de tiempo, algunos nodos estarán emitiendo
            # o no. Esto se refleja en los nuevos valores de posNx y posNy
            posNx, posNy, indexon = functions.readjust_node_emition(flags,
                variables, setup, t, nexperiments)

            dtx=np.sqrt((postx[0]-posNx)**2+(postx[1]-posNy)**2)
```

```
144             dtxNoise=np.append(np.absolute(dtx[0:variables.Mon]+variables.
                    sigma_d*np.random.randn(variables.Mon)),dtx[variables.Mon:
                    variables.N])
145             # Generación a partir de la función randn de MATLAB
146             #dtxNoise=np.append(np.absolute(dtx[0:Mon]+sigma_d*np.array(eng.
                    randn(1,Mon))[0]),dtx[Mon:N])
147
148             # Error en la posición
149             errd=dtxNoise[0:variables.Mon]/dtx[0:variables.Mon]
150             posMxNoise=posNx[0:variables.Mon]*errd
151             posMyNoise=posNy[0:variables.Mon]*errd
152
153             posNxNoise=np.append(posMxNoise, setup.posUx)
154             posNyNoise=np.append(posMyNoise, setup.posUy)
155
156             # Calculo de la "distancia de ruido" entre posiciones
157             DNoise=functions.calculate_noise_distance(variables, posNxNoise,
                    posNyNoise)
158
159         #%%
160             CNoise=var_v*np.exp(-DNoise/variables.Dcorr)
161
162             # Datos
163
164             # Path-loss exponent
165             alpha=np.ones(variables.N)*variables.alphai
166
167             # Shadowing
168             delta=np.append(deltaM[indexon,t],deltaU)
169
170             # Ruido
171             noise=np.append(np.sqrt(variables.varnoise)*np.random.randn(
                    variables.Mon),np.zeros(variables.U))
172
173             # RSSI teórico
174             Z=variables.P-np.diag(10*alpha*np.log10(dtx))@np.ones(variables.N)+
                    delta+noise
175             ZMon=Z[0:variables.Mon]
176             ZUon=Z[variables.Mon:variables.N]
177
178             if flags.flag_tx_est == 1:
179                 # Estimación de la posición del transmisor
180                 dtxNoise, postx_est = functions.estimate_pos_tx(ZMon, posMxNoise,
                                        posMyNoise, posNxNoise, posNyNoise, t)
181
182             else:
183                 postx_est=postx
184
185             # Representación únicamente en la primera iteración
186             if flags.flagfigs==1 and t==0 and kk==0 and nexperiments==0:
187                 # Figura 2 en el paper
188                 gr.representacion_scenario(setup, postx, ZMon)
189
190         #%%
191             # Estimación de alfa y de P
192             meanP, meanalpha, varalpha, varP = functions.alpha_beta_estimation(
```

```
193             flags, variables, dtxNoise, CNoise, ZMon, ZUon, posMxNoise,
                    posMyNoise,
194             posNyNoise, posNxNoise, postx_est)
195
196 #%%
197             if flags.flagfigs==1 and t==0 and kk==0 and nexperiments==0:
198                 # Figura 4 en el paper
199                 gr.representacion_variacion_potencia(variables, ZMon, dtxNoise,
200                                                      meanP, meanalpha)
201
202             # A continuación iría uno por cada algoritmo que se quiera hacer
203             #%%
204             # Gholami
205             if flags.flagGholami==1:
206                 # Estimación a partir de método de Gholami [Mirar referencias en
                        la memoria]
207                 alg.algoritmo_Gholami(variables, setup, posMxNoise, posMyNoise,
                        posNyNoise, posNxNoise, ZMon, kk, t, ZUon)
208             #%%
209             # Watcharapan
210             if flags.flagWatcharapan==1:
211                 # Estimación a partir de método de Watcharapan [Mirar
                        referencias en la memoria]
212                 alg.algoritmo_watcharapan(variables, setup, posNyNoise,
                        posNxNoise, ZUon, ZMon, kk, t)
213             # %%
214             # Se estima la señal recibida en los nodos disponibles
215             #OKD
216             if flags.flagOKD==1:
217                 # Estimación a partir de método de OKD [Mirar referencias en la
                        memoria]
218                 alg.algoritmo_OKD(variables, setup, meanP, meanalpha, dtxNoise,
                        ZMon, DNoise, ZUon, kk, t)
219             #%%
220             # GPrecursive
221             if flags.flagGPrecursive==1:
222                 # Estimación a partir del método de GP para campos variantes en
                        el tiempo
223                 alg.algoritmo_GPRecursive(flags, variables, setup, posNxNoise,
224                 posNyNoise, ZMon, dtxNoise, meanalpha, meanP, varalpha, varP,
225                 shadowstd, ZUon, kk, t)
226             #%%
227             # GPstatic
228             if flags.flagGPstatic==1:
229                 # Estimación a partir del método de GP para campos estáticos en
                        el tiempo
230                 alg.algoritmo_GPStatic(flags, variables, setup, posNxNoise,
231                 posNyNoise, ZMon, dtxNoise, meanalpha, meanP, varalpha, varP,
232                 shadowstd, ZUon, kk, t)
233 #%%
234
235 # Representación de los resultados de error obtenidos a partir de las
        simulaciones
236 my_tplot=1
237 gr.representacion_final_algoritmos(flags, variables, my_tplot, setup)
238
239 # Tiempo total de ejecución
```

```
240 elapsed=time.time()-expend
241 print("El tiempo es: " + str(elapsed))
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] N. R. A.A.Ayeni, N.Faruk and Y. Adediran, *Spatial Spectrum Utilization Efficiency Metric for Spectrum Sharing System*, 4th ed. The Society of Digital Information and Wireless Communications, 2015.

[2] Yu-Shuan Yeh and D. Reudink, "Efficient spectrum utilization for mobile radio systems using space diversity," *IEEE Transactions on Communications*, vol. 30, no. 3, pp. 447–455, 1982.

[3] K. Seshukumar, R. Saravanan, and M. S. Suraj., "Spectrum sensing review in cognitive radio," in *2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT)*, 2013, pp. 1–4.

[4] M. Wellens, J. Riihijarvi, M. Gordziel, and P. Mahonen, "Evaluation of cooperative spectrum sensing based on large scale measurements," in *2008 3rd IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks*, 2008, pp. 1–12.

[5] D. Pfammatter, D. Giustiniano, and V. Lenders, "A software-defined sensor architecture for large-scale wideband spectrum monitoring," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, ser. IPSN '15. Association for Computing Machinery, 2015, p. 71–82.

[6] B. Wang, Q. Chen, L. T. Yang, and H. Chao, "Indoor smartphone localization via fingerprint crowdsourcing: challenges and approaches," *IEEE Wireless Communications*, vol. 23, no. 3, pp. 82–89, 2016.

[7] F. Jiménez, O. Naranjo, J.E.and Gómez, and J. Anaya, "Vehicle tracking for an evasive manoeuvres assistant using low-cost ultrasonic sensors," *Positioning and Tracking Sensors and Technologies in Road Transport*, 2016.

[8] M. Dashti, S. Yiu, S. Yousefi, F. Perez-Cruz, and H. Claussen, "Rssi localization with gaussian processes and tracking," in *2015 IEEE Globecom Workshops (GC Wkshps)*, 2015, pp. 1–6.

[9] P. Kumar, L. Reddy, and S. Varma, "Distance measurement and error estimation scheme for rssi based localization in wireless sensor networks," in *2009 Fifth International Conference on Wireless Communication and Sensor Networks (WCSN)*, 2009, pp. 1–4.

[10] B.-G. Lee, Y.-S. Lee, and W.-Y. Chung, "3d navigation real time rssi-based indoor tracking application," in *JUCT : Journal of Ubiquitous Convergence Technology*, vol. 2, 2008.

[11] A. D. Little. (2009, August) New measure of human brain processing speed. https://www.technologyreview.com/s/415041/new-measure-of-human-brain-processing-speed/.

[12] G. Smit, M. Papadopoulos, A. Macek, and A. Solda, "Why machine learning is crucial to effective utilization of big data," 2018.

[13] A. Rayón. (2016) El machine learning en la era del big data. https://blogs.deusto.es/bigdata/el-machine-learning-en-la-era-del-big-data/.

[14] R. APD. (2019) ¿cuales son los tipos de algoritmos del machine learning? https://www.apd.es/algoritmos-del-machine-learning/.

[15] P. Priyadarshini. (2019) Algorithms for regression problems? https://geekflare.com/choosing-ml-algorithms/.

[16] G. Gahukar. (2018) Classification algorithms in machine learning.... https://medium.com/datadriveninvestor/classification-algorithms-in-machine-learning-85c0ab65ff4.

[17] Wikipedia contributors. (2020) Association rule learning. https://en.wikipedia.org/w/index.php?title=Association_rule_learning&oldid=956074157.

[18] S. big data. (2019) Aprendizaje automatico para deteccion de anomalias. https://sitiobigdata.com/2019/12/24/aprendizaje-automatico-para-deteccion-de-anomalias/#.

[19] P. Gupta. (2017) Decision trees in machine learning.

[20] M. Schott. (2019) Artificial neural networks for machine learning.

[21] S. Sharma. (2017) Artificial neural network (ann) in machine learning.

[22] J. T. Raj. (2019) A beginner's guide to dimensionality reduction in machine learning.

[23] Wikipedia contributors. (2020) Deep learning. https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=964077715.

[24] S. Ray. (2017) Understanding support vector machine(svm) algorithm from examples (along with code). https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

[25] L. Columbus. (2019) State of ai and machine learning in 2019. https://www.forbes.com/sites/louiscolumbus/2019/09/08/state-of-ai-and-machine-learning-in-2019/#6045ff541a8d.

[26] F. Perez-Cruz, S. Van Vaerenbergh, J. J. Murillo-Fuentes, M. Lazaro-Gredilla, and I. Santamaria, "Gaussian processes for nonlinear signal processing: An overview of recent advances," *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 40–50, 2013.

[27] M. C. Taberner, *Evaluacion de procesos Gaussianos en la estimacion de parametros biofisicos*. Trabajo Fin de Máster. Universitat de Valencia, 2013.

[28] V. D. la Pompa Torras, *Procesos Gaussianos para problemas de regresion y estimacion de la incertidumbre*. Trabajo Fin de Máster. Escuela Politécnica Superior de Madrid, 2018.

[29] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[30] K. Bailey. (2016) Gaussian processes for dummies. https://katbailey.github.io/post/gaussian-processes-for-dummies/.

[31] POWLERio. (2020) Sparse gps: approximate the posterior, not the model. https://www.prowler.io/blog/sparse-gps-approximate-the-posterior-not-the-model.

[32] M. Titsias, "Variational learning of inducing variables in sparse gaussian processes," *Journal of Machine Learning Research - Proceedings Track*, vol. 5, pp. 567–574, 04 2009.

[33] C.-A. Cheng and B. Boots, "Incremental variational sparse gaussian processregression," *30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spa*, vol. 5, pp. 567–574, 04 2016.

[34] I. Santos, J. J. Murillo-Fuentes, and P. M. Djurić, "Recursive estimation of dynamic rss fields based on crowdsourcing and gaussian processes," *IEEE Transactions on Signal Processing*, vol. 67, no. 5, pp. 1152–1162, 2019.

[35] Q. Dong and XuXu, "A novel weighted centroid localization algorithm based on rssi for an outdoor environment," *Journal of Communications*, vol. 9, no. 3, 2014.

[36] W. Suwansantisuk and H. Lu, "Localization in the unknown environments and the principle of anchor placement," 06 2015, pp. 2488–2494.

[37] H. Nurminen, M. Dashti, and R. Piché, "A survey on wireless transmitter localization using signal strength measurements," *Wireless Communications and Mobile Computing*, vol. 2017, pp. 1–12, 02 2017.

[38] Wikipedia contributors, "Empirical bayes method," https://en.wikipedia.org/w/index.php?title=Empirical_Bayes_method&oldid=936229395, 2020.

[39] W. Christopher, J.-M. Montero, G. Fernandez-Aviles, and J. Mateu, "Spatial and spatio-temporal geostatistical modeling and kriging," *Journal of Agricultural, Biological and Environmental Statistics*, vol. 22, 06 2017.

[40] M. Gholami, R. Monir Vaghefi, and E. Ström, "Rss-based sensor localization in the presence of unknown channel parameters," *Signal Processing, IEEE Transactions on*, vol. 61, pp. 3752–3759, 08 2013.

[41] W. Suwansantisuk and H. Lu, "Localization in the unknown environments and the principle of anchor placement," 06 2015, pp. 2488–2494.

[42] scipy.org. (2005) lsqlinear documentation. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.lsq_linear.html.

[43] numpy. (2020) numpy. https://numpy.org/.

[44] matplotlib. (2020) matplotlib. https://matplotlib.org/.

[45] scipy. (2020) scipy. https://www.scipy.org/.

[46] Gpy. (2020) Gpy. https://github.com/SheffieldML/GPy.

[47] W. Bulten. (2015) First steps with gpy. https://www.wouterbulten.nl/blog/tech/first-steps-with-gpy/.

[48] J. D. Munoz. (2017) Entornos de desarrollo virtuales con python 3. https://openwebinars.net/blog/entornos-de-desarrollo-virtuales-con-python3/.

[49] Anaconda. (2020) Anaconda. https://www.anaconda.com/.

[50] J. Wågberg, D. Zachariah, T. Schön, and P. Stoica, "Prediction performance after learning in gaussian process regression," 04 2016.

[51] H. Trees, K. Bell, and S. Dosso, "Bayesian bounds for parameter estimation and nonlinear filtering/tracking," *The Journal of the Acoustical Society of America*, vol. 123, p. 2459, 06 2008.

[52] M. Kohm. (2012, May) A bundle of versatile classes and packages. [Online]. Available: http://www.ctan.org/pkg/koma-script/