UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

GRADO EN ING. SIST. DE TELECOM., SONIDO E IMAGEN

“Detección de movimiento utilizando la modulación OFDM de las redes WIFI”

TRABAJO FINAL DE GRADO

Autor/a:
Jerónimo Tocado Villegas

Tutor/a:
Vicenç Almenar Terré

GANDIA, 2016

Resumen

Este trabajo final de Grado consiste en la implementación de un algoritmo de detección de movimiento utilizando la modulación OFDM usada por las redes Wi-Fi. Esto permite la utilización de tecnologías económicas existentes. Se parte de una expresión matemática relacionada con la modulación OFDM y se desarrolla para su aplicación a redes 802.11a/g, utilizando MATLAB para la síntesis de la señal Wi-Fi y las perturbaciones del canal. Para la simulación del movimiento se usa primero MATLAB y luego se prueba en equipamiento SDR, comparando las gráficas de resultados y extrayendo conclusiones. Se proponen también posibles mejoras y alternativas para implementación de bajo coste mediante equipamiento WiFi estándar.

Palabras clave: OFDM, RADAR, WIFI, detección de movimientos, SDR

Abstract

This final project consists in the implementation of a movement detection algorithm leveraging on the OFDM modulation used by WiFi networks. This makes possible the use of existing, low-priced technologies. It starts from a mathematical expression related to OFDM modulation and develops it ..., using MATLAB for WiFi signal and channel effect synthesis. MATLAB is also used for movement simulation first, and then it is tested with SDR equipment, comparing result plots and extracting conclusions. Enhancements to the algorithm and lower cost alternatives are also proposed.

Keywords: OFDM, RADAR, WIFI, movement detection, SDR

Índice

1. Introducción

La idea de este proyecto se gesta entorno a dos hechos:

El primero, después de cursar la asignatura de Tratamiento de Señal en Comunicaciones, se crea un deseo de profundizar en la materia, puesto que es prácticamente exclusiva de los Ingenieros de Telecomunicación, y como también ocurre en otras asignaturas, no da tiempo a profundizar tanto como me hubiese gustado.

El segundo, un interés por entender cómo funciona el proyecto Wisee[1], en cuya prueba de concepto se basa este proyecto. Mediante su estudio se aplican los conocimientos de la asignatura previamente mencionada, como pueden ser modulaciones digitales y procesado multitasa. A parte, se requiere profundizar en la modulación OFDM, en concreto en su implementación en redes Wi-Fi, por lo que se adquieren estos nuevos conocimientos.

1.1. Presentación

Estamos en pleno *boom* del concepto de las *Smart Cities*. Mediante la monitorización y el control de elementos de la ciudad como pueden ser el flujo de tráfico y de personas, se pretende optimizar al máximo los recursos de ésta. Para ello, se necesita una red de sensores que midan estas métricas.

El caso que concierne a este proyecto, la detección de movimiento, podría ser aplicado al ejemplo antes descrito, y aunque ya existen varios tipos de sensores que miden esto mismo, la utilización de Wi-Fi para hacerlo conlleva varias ventajas:

- **Conservación del espectro radioeléctrico** Mediante el uso de señales ya existentes se previenen las interferencias con otros servicios y se conservan las bandas de frecuencia libres para su uso en servicios adicionales
- **Utilización de hardware existente** El hardware Wi-Fi es extremadamente barato. No sólo se evita el desarrollo de un hardware *ad hoc* para este propósito, sino que incluso se puede utilizar infraestructura ya instalada, como por ejemplo, los puntos de acceso de una red Wi-Fi pública.

Otra tecnología reciente que permite llevar a cabo este desarrollo es la tecnología **SDR** (Software Defined Radio). Gracias al aumento de la potencia de cálculo de los ordenadores

[1] wisee.cs.washington.edu

personales, ahora se puede implementar en software la mayor parte de un sistema de radiocomunicaciones, eliminando el coste económico y de tiempo de la elaboración de prototipos.

1.2. Objetivos

1.2.1. Objetivo principal

- Desarrollo de algoritmo para la detección de movimiento mediante señales OFDM, concretamente Wi-Fi

1.2.2. Objetivos secundarios

- Implementación de prototipo mediante Software Defined Radio
- Propuesta para implementación en dispositivo de bajo coste

1.3. Metodología

La metodología utilizada es similar a la de las prácticas de laboratorio realizadas durante el Grado, y se compone de tres grandes bloques:

En primer lugar, se realiza un estudio teórico, partiendo de unos conceptos teóricos previamente proporcionados, en este caso, en el paper de Wisee [1]. Mediante este estudio se consigue una familiaridad con estos conceptos que hace que su futura implementación práctica se produzca de manera más rápida y con menos errores.

En segundo lugar se realiza una simulación, en este caso mediante MATLAB. Con ello se verifican los resultados teóricos previamente obtenidos y se corrigen cómodamente los posibles errores cometidos.

En tercer lugar se prueba en equipamiento SDR, midiendo ya movimiento real.

1.4. Etapas

Las etapas mencionadas en el apartado anterior se componen por las siguientes tareas:

ESTUDIO TEÓRICO

- Estudio del efecto Doppler
- Profundización del conocimiento de la modulación OFDM

SIMULACIÓN

- Generación de la señal Wi-Fi (estudio y modificación del simulador de 802.11 de MATLAB)
- Modelado del canal de transmisión
- Desarrollo del algoritmo de análisis

IMPLEMENTACIÓN

- Desarrollo de interfaz USRP-Matlab
- Experimentación con canal real

1.5. Problemas

Durante el desarrollo del proyecto han surgido problemas o hechos a tener en cuenta:

1.5.1. Hardware

El requisito indispensable para la conexión entre el PC y los USRP es que la tarjeta de red del PC ha de ser Gigabit Ethernet, no sólo por la alta velocidad de transferencia de datos requerida sino que con tarjetas de red inferiores no son compatibles con el USRP (ni siquiera se encienden las luces ni responde a PING). También ha de ser una tarjeta de red de calidad (no sirven tarjetas de red USB), por lo que el portátil con el que se iba a trabajar en un principio resultó no ser compatible.

1.5.2. Software

- Aunque en principio tanto MATLAB como el software de USRP son compatibles con todos los sistemas operativos, se ha determinado que en el que mejor funciona es en Linux, tanto por puro rendimiento como por facilidad de instalación, ya que prácticamente funciona a la primera.
- Aunque facilita mucho el desarrollo de este tipo de proyectos, hay aspectos para los cuales el lenguaje de programación de MATLAB no proporciona suficiente rendimiento. Para ello en MATLAB se pueden convertir las funciones a código C y compilarlas nativamente.

  Concretamente, estas funciones han tenido que ser optimizadas y compiladas para que el algoritmo se ejecute en un tiempo razonable:

- El receptor OFDM
- Funciones de interacción con el USRP
- `symbol_classify`

Pero tampoco es una opción compilarlo todo, ya que para que una función sea convertible a lenguaje C hay algunas restricciones y hay que tenerlas en cuenta al escribir el código MATLAB.

Una optimización importante es el generar los vectores con un tamaño y luego rellenarlos respecto a aumentar el tamaño de los vectores cuando vaya siendo necesario.

También cabe destacar el caso de la función `join_carriers` cuyo rendimiento es inferior en la versión compilada.

2. Cuerpo del trabajo

2.1. Fundamentos teóricos

El fenómeno físico a explotar para la detección de movimiento en este proyecto es el **efecto Doppler**, que es el cambio de frecuencia de una onda emitida por una fuente en movimiento. Este cambio de frecuencia viene descrito por la siguiente expresión:

$$\Delta f \propto \frac{2v\cos(\theta)}{c} f \tag{1}$$

Donde v y θ son la velocidad y el ángulo de la fuente respectivamente, c es la velocidad de la luz en el medio y f es la frecuencia central de la onda.

De esta manera, midiendo esta variación de frecuencia en una onda podemos inferir el movimiento del emisor.

Concretando en el caso de este proyecto:

- La onda a estudiar es Wi-Fi
- El "emisor" es el cuerpo humano: las ondas que se emiten son las reflexiones del auténtico emisor, y es el movimiento del cuerpo el que causa el efecto Doppler sobre las ondas reflejadas.

Teniendo en cuenta estos datos, para detectar un gesto de $0{,}5m/s$, es necesario detectar la siguiente desviación de frecuencia (en el mejor de los casos, con movimiento perpendicular al desplazamiento):

$$\Delta f \propto \frac{2v\cos(\theta)}{c} f = \frac{2 \cdot 0{,}5 \cdot \cos(0)}{3 \cdot 10^8} \cdot 2{,}4 \cdot 10^9 = 8\,\text{Hz} \tag{2}$$

Es decir, una desviación de 8 Hz para redes 802.11b/g cuya frecuencia central es de 2.4 GHz, y algo más del doble (unos 17 Hz) en redes 802.11a cuya frecuencia central es de 5 GHz.

Una vez identificado aproximadamente el valor de la desviación de frecuencia a medir, se da el hecho que éste es varias órdenes de magnitud menor que el de la propia señal Wi-Fi, y por ello apenas apreciable. Por ello, se va a explotar una característica de la modulación **OFDM** utilizada por los estándares 802.11a/g que permitirá detectar esta variación.

La modulación OFDM consiste en dividir el ancho de banda del canal en varias subportadoras y modular una parte de la información total en cada una de ellas. La expresión del modulador es la siguiente:

$$x_k = \sum_{k=1}^{N} \mathbf{X}_n e^{i2\pi kn/N} \tag{3}$$

Es decir, una suma de varios datos ($\mathbf{X}_n$) modulados a una frecuencia ($e^{i2\pi kn/N}$). Como se puede observar, esta expresión coincide con la de la IDFT (transformada inversa de Fourier discreta), por lo que para deshacer la operación y obtener los datos originales, es decir, demodular, se aplica la transformada de Fourier discreta:

$$\mathbf{X}_n = \sum_{k=1}^{N} x_k e^{-i2\pi kn/N} \tag{4}$$

Teniendo esto en cuenta, si se transmiten dos símbolos ODFM iguales consecutivos, se puede realizar el siguiente desarrollo:

- Se transmiten dos símbolos ($x_k^{(1)}$ y $x_k^{(2)}$) que son iguales y se aplica una DFT de tamaño doble (2N):

$$\mathbf{X}_n = \sum_{k=1}^{N} x_k^{(1)} e^{-i2\pi kn/N} + \sum_{k=N+1}^{2N} x_k^{(2)} e^{-i2\pi kn/N} \tag{5}$$

- Como se ha determinado que son iguales, las N primeras muestras transmitidas son las

mismas que las N siguientes, por lo que se puede reescribir la expresión anterior de la siguiente manera:

$$\mathbf{X}_n = \sum_{k=1}^{N} x_k e^{-i2\pi kn/N} + \sum_{k=1}^{N} x_k e^{-i2\pi(k+N)n/N} \tag{6}$$

- Simplificando, obtenemos la siguiente expresión:

$$\mathbf{X}_n = \sum_{k=1}^{N} x_k e^{-i2\pi kn/2N}(1 + e^{-i\pi n}) \tag{7}$$

- El factor $(1 + e^{-i\pi n})$ de la expresión anterior sólo puede tomar 2 valores: 2 cuando n es par o 0 cuando n es impar, es decir, se ha reducido el ancho de banda de cada subcanal analizado por la DFT del receptor a la mitad.

Este desarrollo se generaliza de manera que realizando la FFT a M símbolos consecutivos se reduce el ancho de banda de cada subcanal por un factor de M.

Se puede observar en las siguientes gráficas:

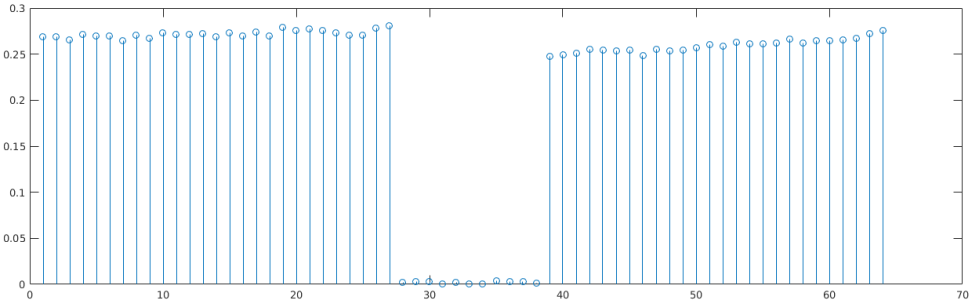

Figura 1: FFT de 1 símbolo

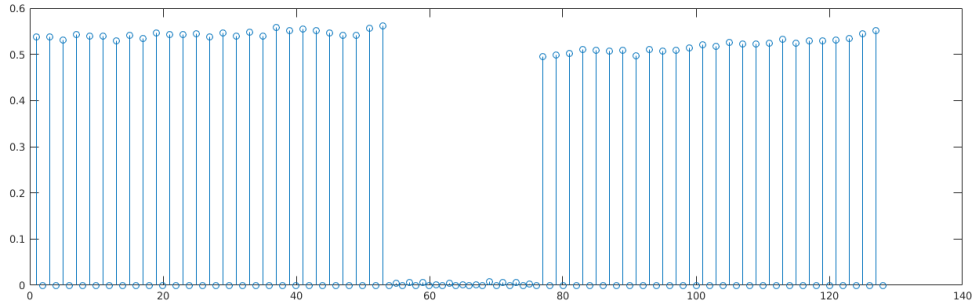

Figura 2: FFT de 2 símbolos consecutivos

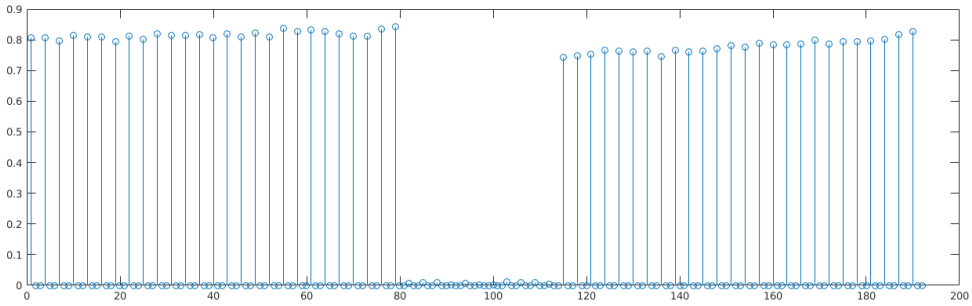

Figura 3: FFT de 3 símbolos consecutivos

Como se puede ver, en la segunda gráfica aparecen muestras con valor a 0 entre las que sí que tienen datos, y en la tercera aparecen 2 por cada muestra con datos.

Para el cálculo de valores concretos hay que tener en cuenta los parámetros usados por 802.11a/g: [2]

- Ancho de banda del canal: $20\,\text{MHz}$
- Número de subportadoras: $N = 64$

El valor propuesto por *WiSee* para conseguir detectar el efecto Doppler producido por los gestos es “una FFT de medio segundo”. Se comprueba que es suficiente para este propósito:

- Símbolos OFDM en $0.5\,\text{s}$:

$$20 \cdot 10^6 \frac{\text{muestras}}{\text{s}} \cdot 0.5\,\text{s} \cdot \frac{1\ \text{símbolo}}{64\ \text{muestras}} = 156250\ \text{símbolos} \tag{8}$$

- Ancho de banda de un subcanal (FFT estándar, 1 símbolo):

$$\frac{20\,\text{MHz}}{64\ \text{subcanales}} = 312.5\,\text{kHz por subcanal} \tag{9}$$

- Ancho de banda de un subcanal (FFT de $0.5\,\text{s}$, 156250 símbolos):

$$\frac{312.5\,\text{kHz}}{156250} = 2\,\text{Hz por subcanal} \tag{10}$$

De esta manera se ha verificado que el valor propuesto por *WiSee* es correcto y suficiente para el objetivo de este proyecto.

2.2. Simulación del sistema

El diagrama de bloques del sistema completo es el siguiente:

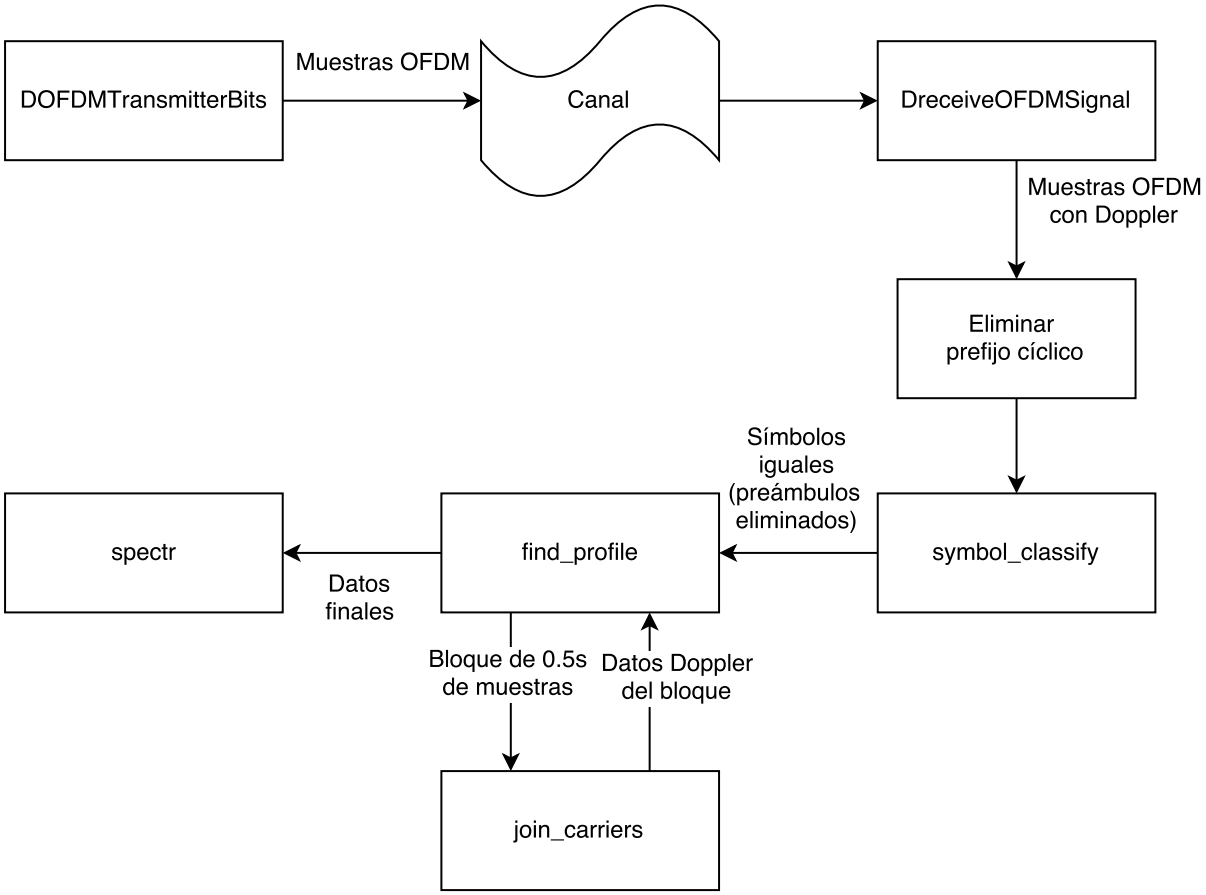

Figura 4: Diagrama de bloques del sistema

El funcionamiento es el siguiente:

- La función DOFDMTransmitterBits genera la señal WiFi completa con todos los símbolos iguales (excepto preámbulos)
- La señal pasa por un canal
- La función DreceiveOFDMSignal recibe los datos y la empieza a preparar, eliminando el prefijo cíclico
- La función symbol_classify elimina los preámbulos, dejando la señal con todos los símbolos iguales y apta para ser analizada
- La función find_profile separa la señal en bloques de 0.5s para su análisis, utiliza la

función join_carriers que es la que realiza la FFT larga de análisis y reúne todos los análisis de todos los bloques

- La función spectr representa gráficamente la información

NOTA: Las funciones cuyo nombre comienza por D son modificaciones de las funciones de MATLAB sin la D inicial. Estas modificaciones se detallan en los siguientes apartados.

Para la simulación del sistema se necesitan modelar dos bloques principales:

2.2.1. Canal con efecto Doppler

El canal escogido para modelar el sistema es el canal Rician, que se caracteriza porque la señal llega al receptor por varios caminos: un camino dominante directo y otros caminos secundarios, que se corresponderían a la señal llegando directamente al receptor más las reflexiones con el cuerpo respectivamente.

Este canal ya está modelado por MATLAB y se usa de la siguiente manera:

```
chan = ricianchan(ts,fd,k)
```

Donde `ts` es el tiempo de muestreo ($t_s = \frac{1}{f_s} = \frac{1}{20 \cdot 10^6}$), `fd` es la máxima frecuencia Doppler y `k` es la relación entre la potencia del camino directo y los caminos secundarios.

2.2.2. Síntesis de señal WiFi

El **Communications System Toolbox** de MATLAB contiene un sistema completo de modulación y demodulación (incluyendo sincronización y ecualización) de OFDM que sigue el estándar 802.11a, por lo que en principio es idóneo para el propósito de este proyecto. Sin embargo, requiere de unas modificaciones y código adicional para que sirva completamente:

OFDMTransmitter La función de esta clase es generar la señal OFDM, incluyendo preámbulos, símbolos piloto y todo lo necesario para el funcionamiento.

El principal inconveniente de esta clase es es que acepta como parámetro el mensaje en formato `string`, que internamente es convertido a bits usando código ASCII (7 bits por carácter). Como la cantidad de bits que se transmite en un símbolo OFDM (usando modulación BPSK para las subportadoras) es 48 bits (1 bit por portadora, 48 portadoras útiles al excluir portadoras piloto y banda de guarda), no se puede completar exactamente 1 símbolo OFDM

mediante caracteres ($\frac{48}{7} = 6{,}86$) ya que los bits del séptimo caracter que no caben en el primer símbolo pasan al segundo, haciendo que símbolos consecutivos no puedan ser iguales.

Se ha optado por modificar la clase eliminando la parte de conversión del `string` y haciendo que transmita siempre la misma secuencia de bits, sustituyendo:

```
% Convert message to bits
msgInBits = coder.const(double(dec2bin(obj.PayloadMessage,
 obj.NumBitsPerCharacter).'));
obj.pPayloadBits = msgInBits(:) - 48;
```

por:

```
obj.pPayloadBits = repmat([0 1 0 ... 0 1 0],1, 500);
```

De esta manera, los datos codificados por cada símbolo son los mismos, por lo que los símbolos serán prácticamente iguales (a excepción de las portadoras piloto, hecho que se tendrá en cuenta más adelante).

OFDMReceiver La función de esta clase es la inversa a la anterior, es decir, recuperar el mensaje desde el símbolo modulado, cosa que en principio no es lo que se busca en este proyecto ya que no se busca decodificar el mensaje sino trabajar con la señal a nivel de muestra. Sin embargo, también contiene la detección de paquete que sí es necesaria, por lo que se ha modificado eliminando las partes de corrección de offset de frecuencia, ecualización, demodulación y conversión de bits a `string` que es la salida original de la clase. Sustituyendo:

```
% Correct frequency offset
[oneFrameToProcess, estFreqOffset] = coarseFreqCorrection(obj, oneFrameToProcess);

% Apply equalizers
[postEqData, preEqData, eqGains] = frameEqualization(obj, oneFrameToProcess);

% BPSK demodulation
msgInBits = step(obj.pBPSKDemod, postEqData(:));

% Save bits to output
y = [y, msgInBits(1:obj.pNumBitsPerDisplay)']; %#ok<AGROW>
```

por:

```
y(lastOutIdx:lastOutIdx+length(oneFrameToProcess)-1) = oneFrameToProcess;
lastOutIdx = lastOutIdx+length(oneFrameToProcess);
```

Se consiguen obtener las muestras justo después de la detección de paquete, listas para trabajar.

2.2.3. Algoritmo de detección

Una vez obtenidas las muestras, se necesitan hacer unos últimos ajustes antes de hacer la FFT larga:

- Eliminación de prefijos cíclicos: cada símbolo lleva delante 16 muestras iguales a las últimas 16 muestras del propio símbolo para ayudar a la estimación del canal y evitar interferencia entre símbolos.

  ```
  for i = 1:length(fftData2)/(64+64/4)
      fftData3(:,i) = fftData2(1+64*i+16*(i-1):64*(i+1)+16*(i-1));
  end
  ```

- Eliminación de portadoras piloto: estas portadoras cambian de valor cada cierta cantidad de símbolos, por lo que se eliminan directamente pasando cada símbolo al dominio de la frecuencia, multiplicando por 0 estas portadoras y devolviéndolo al dominio del tiempo:

  ```
  mask = ones(64,1);
  mask(8) = 0;
  mask(22) = 0;
  mask(44) = 0;
  mask(58) = 0;

  fftData4 = complex(zeros(64, length(fftData3)));

  for i=1:length(fftData3)
      newSymb = ifft(fft(fftData3(:,i)) .* mask);
  ```

- Clasificación de símbolos: se eliminan los preámbulos largos de la señal, dejando sólo los símbolos con los datos conocidos. Para ello, se descartan los símbolos cuyo coeficiente de correlación con el símbolo conocido es menor a 0.9:

```
c = corrcoef(newSymb, symb);
c = abs(c(2,1));
if c > 0.9
    fftData4(:,i) = newSymb;
else
    fftData4(:,i) = zeros(64,1);
end
```

Llegado a este punto la señal está lista para ser analizada. Según *WiSee*, la manera de realizar el análisis es calcular la FFT de 0.5 s cada 5 ms. Es decir:

- La primera FFT se calcula desde el comienzo de la señal hasta 0.5 s
- La segunda FFT se calcula desde 5 ms hasta 505 ms. Nótese que una gran parte de la señal se solapa con la primera FFT.

Para poder hacer el cálculo se necesita el valor en muestras o símbolos en vez de en tiempo, por lo que se calcula este valor:

$$5 \cdot 10^{-3} \text{ s } \cdot 20 \cdot 10^{6} \frac{\text{muestras}}{s} \cdot \frac{1 \text{ símbolo}}{64 \text{ muestras}} = 1562{,}5 \text{ símbolos} \tag{11}$$

Para no cortar un símbolo a mitad, se trunca este valor y se decide calcular la FFT cada 1562 símbolos.

Con esto se tienen todos los datos necesarios para implementar el propio algoritmo de análisis, divido en 2 funciones:

find profile

```
function [ zm ] = find_profile( endData )

        % Definicion de valores calculados previamente
        PIECE_LENGTH = 10e6;
        SYMBOL_OFFSET = 1562;
```

```
        % Condiciones iniciales para empezar desde el principio
        offset = 1;
        dataLength = length(endData);
        counter = 0;
        from = offset;
        to = offset + PIECE_LENGTH - 1;
        peak = -1;

        % Salida
        zm = [];

        % Mientras quede que analizar...
        while to < dataLength

        % ... extraemos un trozo ...
        piece = endData(from:to);
        counter = counter + 1;

        % ... lo analizamos ...
        [joined, peak] = join_carriers(piece, peak);

        % ... guardamos el resultado ...
        zm = [zm; joined];

        % ... y pasamos al siguiente trozo ...
        offset = offset + SYMBOL_OFFSET*64;
        from = offset;
        to = offset + PIECE_LENGTH - 1;

        end

end
```

join carriers

```
function [ joined, peakOut ] = join_carriers( piece, peakIn )

        % Se calcula el valor absoluto de la FFT de 0.5s
        piece = abs(fft(piece));
        pieceLen = length(piece)/64;

        % Se promedian los valores de todas las portadoras
        joined = zeros(1,pieceLen);
        for i=0:62
        joined = joined + piece(i*pieceLen + pieceLen/2:(i+1)*
           pieceLen-1 + pieceLen/2);
        end

        % Para representar graficamente, se busca el maximo...
        if peakIn < 1
```

```
        [~,peakOut] = max(joined);
        else
        peakOut = peakIn;
        end

        % ...y se representa en el centro
        if peakOut > 10
        joined = joined(peakOut-50:peakOut+50);
        else
                joined = [];
        end
end
```

Para representar el resultado (la frecuencia Doppler en función del tiempo) se utiliza la siguiente función:

```
function spectr( zm )

        [xSize, ySize] = size(zm);

        freqBound = ((ySize-1)/2)*2;

        f = linspace(-freqBound, freqBound, ySize);
        t = linspace(0, 1562*64/20e6 * xSize, xSize);

        surf(f, t, zm);
        shading interp;
        view(90,90);
        xlabel('Frecuencia Doppler (Hz)');
        ylabel('Tiempo (s)');

end
```

2.2.4. Resultados de la simulación

Se ha ejecutado el simulador en 2 casos: sin canal (transmisor directamente conectado a receptor) y con canal Rician con frecuencia máxima Doppler de 8Hz (equivalente a gesto de de 0.5m/s) obteniendo los siguientes resultados:

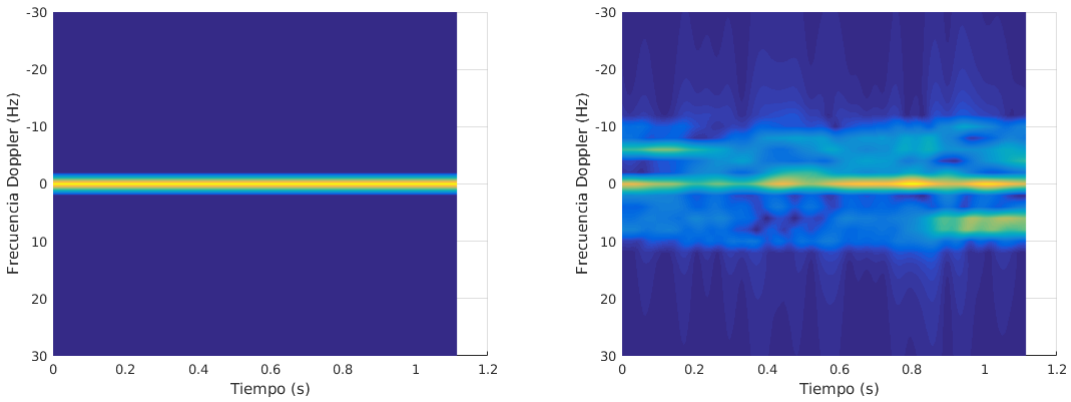

Figura 5: Resultados sin canal y con canal 8Hz

De estas gráficas se extraen las siguientes conclusiones:

- En la primera imagen se observa que toda la energía de la señal está concentrada en la frecuencia Doppler de 0Hz, es decir, sigue el camino directo de transmisor a receptor por lo que no hay desviación de frecuencia.
- En la segunda imagen también se observa una mayor concentración de energía en $f = 0\,\text{Hz}$, sin embargo también se ve que una parte de esta energía se ha dispersado a su alrededor, en unos $8\,\text{Hz}$ en cada dirección. Las frecuencias positivas significan un movimiento hacia el receptor y las negativas un alejamiento de éste.

  Como conclusión, los resultados obtenidos concuerdan con lo esperado.

2.3. Experimentos

El siguiente paso es sustituir el canal simulado por el entorno real. Para ello, se van a utilizar 2 USRP N210, uno como emisor y otro como receptor.

Este dispositivo consiste, a grandes rasgos, en un ADC (para recepción), un DAC (para transmisión) y unos mezcladores. Se conecta por Ethernet al PC y se envían las muestras a transmitir en banda base, junto a la frecuencia a la que se quiere transmitir.

2.3.1. Esquema del montaje

Los materiales utilizados han sido los siguientes:

- PC de sobremesa:

- Procesador: Intel Core i5-6600 (3.3 GHz)
- RAM: 16GB
- Sistema operativo: Ubuntu 16.04.1 LTS
- MATLAB R2015b

- Router TP-LINK TL-WDR4900 como switch Gigabit Ethernet
- 2 unidades de Ettus USRP N210
- 2 antenas Wi-Fi omnidireccionales estándar
- 2 adaptadores SMA (USRP) a RP-SMA (antenas)

2.3.2. Funcionamiento del USRP N210

MATLAB proporciona en su web[2] el **USRP Support from Communications System Toolbox** que permite la transmisión y recepción de muestras. Su funcionamiento es el siguiente:

```
%Configuracion de parametros
radio = comm.SDRuTransmitter('IPAddress', '192.168.10.2',...
          'CenterFrequency', 2.5e9,...
          'Gain', gain, ...
          'InterpolationFactor', 100/msps);

% Division de muestras por bloques
iterations = ceil(length(data)/frame_size);
data_length = length(data);
 for i=0:iterations-1
   from = 1+i*frame_size;
   to = min(1+(i+1)*frame_size, data_length);
   % Envio
   step(radio, data(from:to));
 end
```

Transmisor Los parámetros a configurar son los siguientes:

- IPAddress: La dirección IP del dispositivo. Se puede cambiar para poder conectar varios dispositivos al mismo ordenador. En este caso la que está configurada es la que viene por defecto.

[2] http://es.mathworks.com/hardware-support/usrp.html

- CenterFrequency: La frecuencia a la que se emitirá la señal. Se ha elegido 2.5 GHz porque está cerca de la banda de 802.11b/g pero no es un canal estándar, por lo que no se interferirá en otras redes WiFi

- Gain: La ganancia (en dB) del transmisor. La máxima configurable para este dispositivo es de 31.5dB

- InterpolationFactor: El DAC interno del dispositivo trabaja internamente a una tasa de 100Msps, de manera que si queremos transmitir señales a tasas menores hay que indicar al dispositivo que interpole la señal antes de enviarla.

  En este caso, en principio el factor de interpolación se debería configurar a $\frac{100Msps}{20Msps} = 5$, pero el dispositivo trabaja mejor con factores de interpolación cuyo valor sea potencia de 2, para así utilizar sólo filtros de media banda, por lo que se ha escogido un factor de interpolación de $\frac{100Msps}{25Msps} = 4$ y la interpolación de 20Msps a 25Msps se realiza por software en MATLAB.

Receptor Los parámetros a configurar son similares a los del apartado anterior:

```
%Configuracion de parametros
radio = comm.SDRuReceiver('IPAddress','192.168.10.20', ...
                'CenterFrequency', 2.5e9, ...
                'DecimationFactor', 100/msps, ...
                'Gain',32, ...
                'FrameLength', frame_size
                );
```

- IPAddress: Mismo propósito que en el transmisor, pero observar que la IP ha cambiado.

- CenterFrequency: En este caso es la frecuencia central que se bajará a banda base.

- DecimationFactor: Análogo al InterpolationFactor del apartado anterior. El ADC trabaja a 100Msps por lo que si sólo se necesitan 25Msps se puede diezmar en el propio USRP para ahorrar la transmisión de datos extra.

- Gain: La ganancia (en dB) del receptor. La máxima configurable para este dispositivo es de 38dB

- FrameLength: En vez de transmitir las muestras individualmente, se agrupan en Frames.

Particularidades

- Como se ha comentado anteriormente, los dispositivos funcionan mejor con tasas de interpolación y diezmado con valor de potencia de 2, por lo que se usa la función `resample` antes de transmitir y después de recibir los datos. La propia función se encarga de calcular y filtrar con los filtros pertinentes para evitar aliasing.

```
txSig25 = resample(txSig20, 25,20);
txSig20 = resample(txSig25, 20,25);
```

- Hay que tener en cuenta que las muestras se deben enviar en el rango de -1 a 1 por lo que al generarse se deben normalizar antes de enviar:

```
data = data / max(abs([max(real(data)) min(real(data)) max(
    imag(data)) min(imag(data)) ])) * 0.8;
```

2.3.3. Resultados

El montaje del experimento es el siguiente: Los dos dispositivos están situados a 1 metro de distancia entre ellos, conectados a un switch y éste también al ordenador por Ethernet.

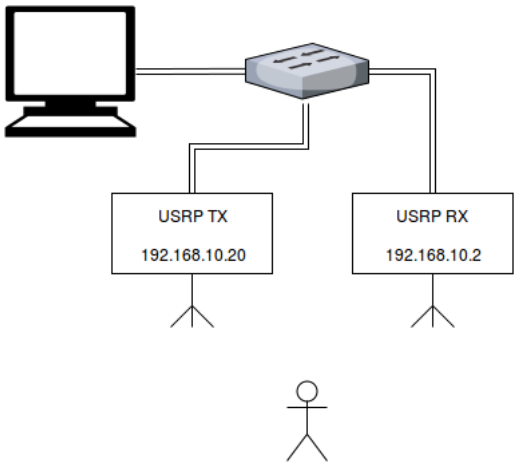

Figura 6: Montaje del experimento

Figura 7: Montaje físico

Se han representado 2 escenarios y se han realizado 2 medidas de cada uno de ellos.

En primer lugar se han hecho las mediciones sin movimiento:

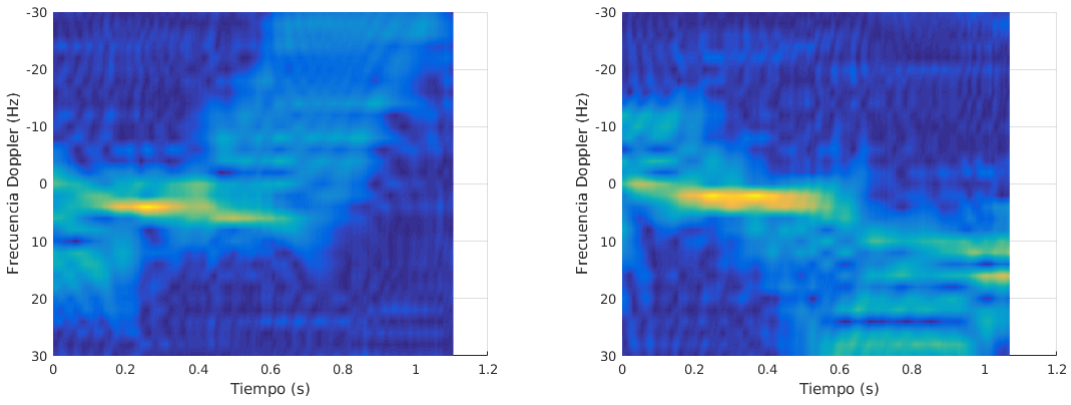

Figura 8: Resultados sin movimiento

En segundo lugar, se ha realizado la medida con movimiento, centrado entre los dos dispositivos y a un metro de distancia (ver figura 6):

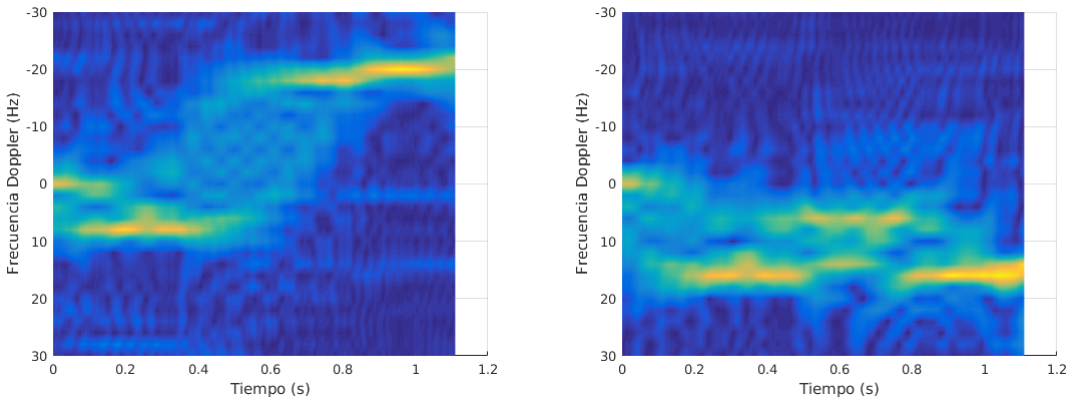

Figura 9: Resultados con movimiento movimiento

Comparando las gráficas, se puede observar que cuando hay movimiento, la energía se distribuye y se aprecian dos franjas principalmente, mientras que en las gráficas sin movimiento sólo se aprecia una franja principal con más energía.

Con esto, pese a que no se ha logrado el nivel de precisión esperado, se concluye que es posible detectar movimiento mediante el concepto expuesto en este proyecto.

3. Conclusiones

3.1. Trabajo futuro

Con este proyecto se ha demostrado que es posible detectar movimiento utilizando la modulación OFDM de las redes WiFi, sin embargo mediante su desarrollo se han abierto muchos campos de investigación para posible mejora, como por ejemplo:

- Aumento de la precisión: Con los 2Hz de resolución que se ha obtenido debería ser posible detectar gestos humanos, sin embargo se requiere mejorar en aspectos como la sincronización para reducir el ruído introducido por los offsets de frecuencia
- Optimización de algoritmos para una futura implementación en tiempo real

3.1.1. Propuesta de implementación de bajo coste

Existe una característica en los chipsets WiFi Atheros AR92xx y AR93xx que consiste en que reportan datos sobre el espectro, concretamente los datos de la FFT en banda base.

`https://wireless.wiki.kernel.org/en/users/drivers/ath9k/spectral_scan`

Con ello, se podría implementar el sistema en un routers Wi-Fi **TP-LINK TL-MR3020** cuyo precio es menor de 30 euros. Se ha seleccionado este equipo por sus características:

- Chipset AR9331 compatible con `spectral_scan`
- Compatible con **OpenWRT** (distribución Linux para dispositivos embebidos)

4. Bibliografía

Referencias

[1] Qifan Pu, Sidhant Gupta, Shyam Gollakota, Shwetak Patel, *Whole-Home Gesture Recognition Using Wireless Signals*, `http://wisee.cs.washington.edu/wisee_paper.pdf`

[2] IEEE, *802.11a-1999 High-speed Physical Layer in the 5 GHz band*, 1999, `http://standards.ieee.org/getieee802/download/802.11a-1999.pdf`

[3] Hanzo, Lajos, *OFDM and MC-CDMA for broadband multi-user communications, WLANs and broadcasting*, 2003,

[4] Chiueh, Tzi-Dar, *OFDM baseband receiver design for wireless communications*, 2007,

[5] Kalivas, Grigorios, *Digital radio system design*, 2009,