Universidad de la República
Facultad de Ingeniería

gr-tempest 2.0: Mejoras a la Implementación de Tempest en GNU Radio

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Pablo Bertrand, Felipe Carrau, Victoria Severi

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Federico La Rocca ..... Universidad de la República

Tribunal

Pablo Belzarena ..... Universidad de la República
Ignacio Ramírez ..... Universidad de la República
Marcelo Rodríguez ..... Universidad de la República

Montevideo
martes 16 noviembre, 2021

gr-tempest 2.0: Mejoras a la Implementación de Tempest en GNU Radio, Pablo Bertrand, Felipe Carrau, Victoria Severi.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 115 páginas.
Compilada el martes 16 noviembre, 2021.
http://iie.fing.edu.uy/

La desconfianza es la madre de la seguridad.

Aristófanes

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Agradecemos a Federico por su disposición y orientación a lo largo del proyecto, además de la confianza en nuestro trabajo sobre gr-tempest. A nuestras familias y amigos por el apoyo constante. A Gonzalo Gutiérrez por su asesoramiento en radiofrecuencia. Al Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería por proveer los medios necesarios para el desarrollo de este trabajo.

Esta página ha sido intencionalmente dejada en blanco.

A todos los que nos bancaron en este tiempo.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Las interfaces más utilizadas en la actualidad para la conexión de monitores y despliegue de imágenes en sistemas digitales son VGA y HDMI. Las señales transmitidas a través de ellos refieren a los valores de los píxeles en claro y las especificaciones de sus estándares son de público conocimiento, lo cual implica una brecha de seguridad en su transmisión: las emisiones electromagnéticas de los cables.

Trabajos anteriores han ahondado en las consecuencias de este fenómeno y varias implementaciones demuestran la posibilidad de obtener información mediante el espionaje de esas emanaciones involuntarias. En 2014, el proyecto TempestSDR innova al incorporar técnicas de Radio Definida por Software (SDR) para su implementación, permitiendo rebajar significativamente los costos del hardware dada su capacidad de trasladar parte del procesamiento a software.

Más recientemente, el trabajo original de gr-tempest implementa a mediados del 2020 un sistema de espionaje similar a TempestSDR sobre el framework GNU Radio, un software libre que provee mayor facilidad para mantener y extender el código. En la búsqueda de emular la operativa de TempestSDR, se alcanzó una versión operativa que logra interceptar las imágenes de monitores en la práctica, pudiendo identificar oportunidades de mejora en su funcionamiento.

En este proyecto se diseñan e implementan los cambios requeridos para esas mejoras, modificando funcionalidades existentes y agregando nuevas. Se sustituye el módulo de sincronización horizontal por uno capaz de detectar los bordes tanto horizontales como verticales para centrar la imagen. Se reemplaza también la funcionalidad de cálculo del índice de interpolación para la corrección de muestreo, logrando solucionar algunos problemas de su operativa.

Dos herramientas nuevas fueron añadidas. Por una lado, una función de inferencia de la resolución del monitor espiado, a realizar en una etapa previa para conocer los parámetros necesarios para el propio espionaje. Por el otro, ecualización de la imagen por dos métodos, afinando distintos detalles requeridos para una mejor percepción de la imagen.

Finalmente, técnicas por software y por realimentación de hardware son implementadas para la optimización de la corrección de muestreo, buscando reducir los recursos computacionales dedicados a ese proceso.

El resultado, junto con una implementación de hardware de radiofrecuencia diseñada para la resolución 1080p, consiste en un sistema capaz de espiar la imagen de monitores de esa resolución en tiempo real y a una tasa de 50 MSps, permitiendo incluso la aplicación de ecualización online durante ese espionaje.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Agradecimientos III

Resumen VII

1. Introducción **1**
1.1. Contexto y Motivación 1
1.2. Objetivo del Proyecto 3

2. SDR: Radio Definida por Software **7**
2.1. Descripción 7
2.2. Hardware 8
2.3. Software 9
2.4. GNU Radio 9

3. Caracterización de una Señal de Video **13**
3.1. Trama de Video 13
3.1.1. Parámetros 14
3.1.2. Señal VGA 15
3.1.3. Señal HDMI 17
3.2. Representación Temporal y Espectral 17
3.3. Señal Detectada 19

4. Antecedentes Técnicos **21**
4.1. TempestSDR 21
4.1.1. Funcionamiento 21
4.1.2. Detección de Sincronización 23
4.1.3. Detección de Resolución 25
4.2. gr-tempest 26
4.2.1. Funcionamiento 26
4.2.2. Puntos de Mejora 29

5. Implementación de Hardware **31**
5.1. Requerimientos del Sistema 31
5.2. Diseño de Componentes 33

Tabla de contenidos

6. Implementación de Software **39**
6.1. Inferencia de Resolución 39
6.2. Detección de sincronización 42
6.2.1. Funcionamiento 42
6.3. Corrección de Muestreo 45
6.4. Aumento de la Frecuencia de Muestreo 49
6.4.1. Realimentación SDR 51
6.4.2. Descarte de Cuadros Completos 52
6.5. Ecualización 55
6.5.1. Ecualización Lineal 55
6.5.2. Corrimiento en Frecuencia 56

7. Medidas de Desempeño y Análisis de Resultados **61**
7.1. Pruebas de Hardware 61
7.1.1. Medidas VNA 61
7.1.2. Medidas GNU Radio 63
7.2. Resoluciones Inferidas 65
7.3. Sincronización 66
7.4. Cálculo del Índice de Interpolación 71
7.5. Rendimiento de CPU 73
7.5.1. Corrección en Software 73
7.5.2. Corrección en Hardware 76
7.6. Imágenes Ecualizadas 79
7.6.1. Corrección del error de frecuencia 80
7.6.2. Ecualización Lineal 82
7.7. Sistema Completo 84

8. Conclusiones y Trabajo a Futuro **89**
8.1. Conclusiones 89
8.2. Trabajo a Futuro 90

Referencias **93**

Índice de tablas **96**

Índice de figuras **98**

Capítulo 1

Introducción

1.1. Contexto y Motivación

Desde hace ya décadas, las computadoras personales se han asentado como componentes fundamentales en dimensiones como el trabajo, la educación y el ocio. Estos ámbitos se ven profundamente atravesados por el uso de los dispositivos digitales, cuyas funciones son en algunos casos convenientes y en otros necesarios para desenvolverse en distintas actividades y tareas.

En Uruguay, el uso de computadoras alcanzó el 58 % en los hogares en 2019 [1], lo cual no llega a contemplar los cambios generados por la pandemia de SARS-CoV-2. Así como varias de las aplicaciones utilizadas no implican riesgos para el usuario (como pueden ser el video on-demand, páginas web públicas o videojuegos), otros usos de estos sistemas pueden contener información sensible. Como ejemplo de esta exposición a nivel personal, se tiene que el 52 % de la ciudadanía realiza pagos en línea y el 45 % de ellos lo hace a través de la PC [2]. Otros usos sensibles incluyen el correo electrónico, imágenes de uso personal, registros de compras y, en general, sitios que requieren usuarios y almacenen datos. En el caso de empresas, a estos riesgos se les puede agregar la información privada de la propia compañía, como planes estratégicos o datos salariales, y la información confidencial de clientes, como historias clínicas o cuentas bancarias.

En este contexto, la seguridad de la información que se debe implementar ya no es un concepto ajeno a la mayoría de los usuarios de los sistemas. De hecho, el 80 % de las empresas tienen conciencia sobre los riesgos que implica la ciberseguridad [3], independientemente de las medidas que tomen al respecto. Este criterio, sin embargo, está a veces basado en un concepto de la seguridad que se limita a información transmitida a través de internet que, justamente, no se trata del único medio para interceptar comunicaciones.

TEMPEST es el nombre utilizado por la Agencia Nacional de Seguridad (o NSA) de Estados Unidos en 1972, para referirse a los estudios vinculados a las emanaciones electromagnéticas comprometedoras en cables por los que circula información [4]. Estas investigaciones se mantuvieron como información clasificada hasta 2007, dado el marco bélico en el que se las ubicó: durante la segunda guerra

mundial, investigadores de *Bell Lab* descubrieron que el texto que debía ser transmitido por cable a un dispositivo de encripción para luego ser enviado de forma segura, podía ser interceptado a unos 24 metros y reconstruyendo un 75 % de su contenido, a causa de sus emanaciones electromagnéticas. Esto hizo priorizar las problemáticas de seguridad vinculadas al fenómeno, por sobre otras implicancias de las emanaciones como podía ser su efecto en la salud.

La primera publicación técnica no clasificada vinculada al riesgo de las emanaciones electromagnéticas fue realizada por Wim van Eck en 1985, tratando en particular las provenientes de monitores de PC [5]. En ella da evidencia de que un ataque podría, efectivamente, representar una brecha de seguridad en las emanaciones, aunque sus pruebas se vieron limitadas por la inaccesibilidad de los equipos (restricciones de exportación y elevados costos).

No es sino hasta 2003 que se hace otro importante avance en el tema cuando Markus Kuhn publica un reporte técnico [6], implementando un sistema en tiempo real con una placa FPGA y un receptor AM, aunque seguía siendo una solución cara para el usuario. Elibol, Sarac y Erer [7] mejoran esta solución, agregando distancias de hasta 50 metros al espionaje e implementando el uso de la autocorrelación para obtener información de la señal, sin solucionar el problema del elevado costo de los componentes.

En 2014, sin embargo, se publica el proyecto *TempestSDR* como parte de la tesis de maestría de Martin Marinov en la Universidad de Cambridge [8]. En él se presentan herramientas para el espionaje de emisiones electromagnéticas de cables VGA con el uso de radio definida por software (o SDR, del cual se habla en el Capítulo 2). Esto implicó un cambio hacia requerimientos de hardware mucho menos exigentes, en términos tanto de costo como de movilidad. Los detalles de esta implementación se analizan en la Sección 4.1.

Este contexto da lugar al primer trabajo realizado en la Universidad de la República relativo a TEMPEST [9]. La tesis de Pablo Menoni contribuye a la vigencia de la línea de investigación, extendiendo el estudio a las señales de video HDMI. Adicionalmente, propone métodos de ecualización para una señal con las características de la obtenida por espionaje, orientado a mejorar la distinción de detalles (caracteres, texto) propios de la imagen de monitor. Algunos de sus resultados son procesados y demostrados mediante el *framework* GNU Radio (que se verá en detalle en la Sección 2.4) y se utilizan en la implementación de este proyecto.

La dirección de dicha tesis, la experiencia en GNU Radio y el repentino aumento de tiempo en casa a causa de la emergencia sanitaria, llevó al tutor de Menoni, Federico La Rocca, a gestar la idea que sería la base para el presente trabajo: la implementación autocontenida de TempestSDR, realizada de forma completa por Marinov, está limitada por su propia naturaleza, careciendo de las ventajas de una plataforma como GNU Radio para el mantenimiento y la extensión del código.

A partir de esa idea, generó un sistema independiente que, si bien se inspira en TempestSDR, no depende de este: gr-tempest [10] está basado en GNU Radio, utilizando bloques de procesamiento propios y otras dependencias, así como drivers de SDR que fueron diseñados para GNU Radio (sin requerir que se desarrollen

específicamente para el aplicativo). El sistema llega a una versión funcional para el espionaje de monitores VGA y HDMI basado en SDR y se analiza en profundidad en la Sección 4.2.

En el afán por continuar esa implementación es que nace este proyecto, tomando la versión que era funcional pero que le faltaban funcionalidades y tenía bloques con lugar a mejoras, para analizar su operativa y contrastarla tanto con TempestSDR como con la bibliografía pertinente.

1.2. Objetivo del Proyecto

El objetivo de este proyecto es la implementación de funciones nuevas y cambios en funciones existentes para mejorar la operativa de gr-tempest, acercándola al comportamiento de TempestSDR y poniendo en juego consideraciones de la bibliografía. Esto se traduce en especificaciones a modificar en el sistema, que serán detalladas en la Sección 4.2.2 en base al análisis del sistema inicial.

Un objetivo adicional es el diseño e implementación de una configuración de hardware de radiofrecuencia para las pruebas y ejecución en tiempo real de gr-tempest. Por su relevancia en términos de actualidad y extensión, se selecciona la resolución 1080p o FHD como referencia para el diseño del hardware, determinando los parámetros de la señal a considerar.

Las partes mencionadas de software y hardware, junto con el monitor a espiar (conectado a un ordenador por VGA o HDMI) conforman el escenario general de espionaje mostrado en la Figura 1.1: el ordenador envía al monitor las señales correspondientes al video a desplegar; el cable, por tanto, tendrá voltajes y corrientes variables (vistas en mayor detalle en el Capítulo 3) que generarán campos electromagnéticos proporcionales a estas variaciones [11].

Los campos generados se pueden propagar por distintos medios, incluido el aire. De esta forma, una antena ubicada a una cierta distancia será capaz de captar la señal y, luego de que un circuito de radiofrecuencia acondicione la señal, el sistema basado en SDR podrá sintonizar la frecuencia de la portadora (o un armónico) y demodular la señal para entregar las muestras al detector en software (GNU Radio).

Sobre GNU Radio opera gr-tempest que, por un lado, interactúa con el SDR para proveerle información de la frecuencia de muestreo, la frecuencia portadora y la ganancia a utilizar, además de recibir las muestras de la señal de video; por el otro, utiliza estas muestras para corregir el muestreo, centrar la imagen y desplegarla en la pantalla del espía.

Para verificar los objetivos en el sistema mencionado, se diseñó en primer lugar un algoritmo de sincronización tanto horizontal como vertical. El mismo es basado en el implementado en TempestSDR, pero con las modificaciones pertinentes para adaptarlo a GNU Radio y mejorar su comportamiento.

Luego se generó una funcionalidad de inferencia de la resolución del monitor para cualquier señal espiada, haciendo uso del cálculo de autocorrelación de la misma para ubicar los picos que den información de la frecuencia en la que se repiten los cuadros y las líneas.

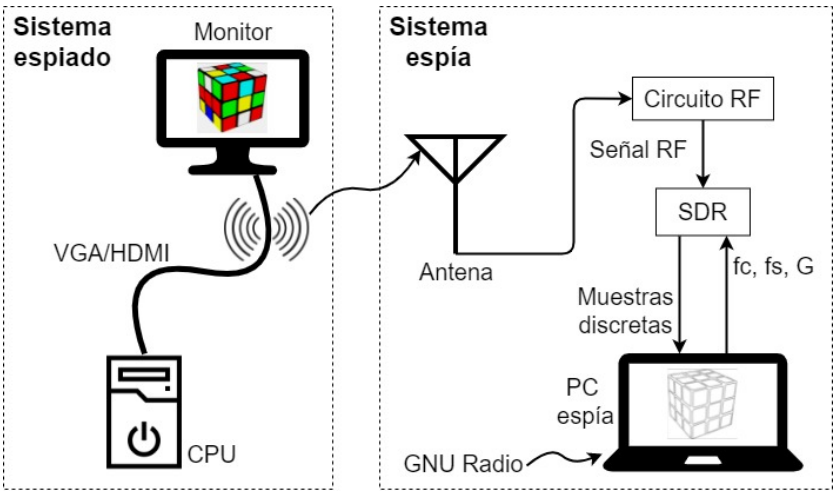


Figura 1.1: Diagrama esquemático del escenario de espionaje, incluyendo tanto al sistema espiado como al espía.

Un razonamiento análogo dio lugar a un bloque que sustituye la función de corrección de muestreo, calculando el índice de interpolación también a partir de la autocorrelación.

La interpolación a partir de ese índice implica una carga significativa sobre los filtros digitales de la unidad central de procesamiento (CPU) de la PC espía, por lo que se exploraron dos métodos para alivianar su carga: descartar cuadros completos para que la interpolación se efectúe en menos muestras, o realimentar directamente la frecuencia corregida al SDR de forma de no requerir interpolación. Tras evaluar los resultados en cada caso, se opta por la segunda como método principal y se deja el descarte como alternativa en software.

Finalmente, y haciendo uso de los resultados teóricos de Pablo Menoni, se implementa un sistema de ecualización que permite afinar los detalles de la imagen, permitiendo extraer más información de la imagen que se visualiza.

El resultado del trabajo comprende el hardware apropiado para la resolución mencionada, el módulo de GNU Radio correspondiente al software desarrollado, y tres escenarios de operación: el de inferencia de resolución, para recibir una señal de cualquier resolución de video y devolver la información de la misma; el de espionaje de monitores, para recibir señal del hardware o de una grabación y desplegar su imagen en pantalla; y el de ecualización, para recibir la señal y procesarla de forma que su despliegue haga más notables los detalles en los caracteres.

Para la documentación del trabajo, se comienza por introducir las propiedades y beneficios del paradigma de SDR en el Capítulo 2 y la caracterización de la señal de video emitida y detectada en el espionaje en el Capítulo 3. Luego, en el Capítulo 4, se analizan los antecedentes técnicos, pasando tanto por TempestSDR como gr-tempest. Se introduce en el Capítulo 5 la implementación del hardware para la resolución deseada y, como centro del proyecto, la implementación del software desarrollado en el Capítulo 6. Por último, en el Capítulo 7 se establecen los criterios para analizar y evaluar los resultados obtenidos, para luego plantear

las conclusiones y las posibles líneas de trabajo a continuar en el futuro en el Capítulo 8.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

SDR: Radio Definida por Software

Una de las claves del valor aportado por las nuevas implementaciones de espionaje por emisiones electromagnéticas es el uso de SDR en los sistemas de adquisición y procesamiento de las señales. En este capítulo se describe su funcionamiento y las ventajas de su implementación. Adicionalmente, se introduce GNU Radio como *framework* para su uso.

2.1. Descripción

SDR (*Software Defined Radio*) engloba técnicas de recepción de señales de radiofrecuencia en base al procesamiento por software. Esto consiste en trasladar funcionalidades de los equipos de recepción de radio a soluciones basadas en software, lo cual da una mayor versatilidad a la hora de hacer modificaciones o adaptaciones.

Los equipos tradicionales, en oposición a este tipo de soluciones, utilizan hardware dedicado que, dependiendo de su aplicación, se ven limitados en lo relativo a la codificación con la que pueden trabajar y su espectro de frecuencias. Una limitante del hardware dedicado es que cualquier modificación a estos aspectos posiblemente solo sea posible mediante el cambio de componentes.

El funcionamiento de SDR le permite utilizar el mismo hardware en aplicaciones variadas, quedando las tareas de modulación o detección para resolver en software. En hardware se realiza la sintonización de la frecuencia portadora, así como la frecuencia de muestreo - dos aspectos claves en la recepción de las señales y de especial importancia para este trabajo.

El diagrama de la Figura 2.1 representa la recepción por SDR: luego de la amplificación, se demodula en fase y cuadratura para una cierta portadora y se realiza el muestreo a una determinada frecuencia. Desde software, se muestra cómo ciertos parámetros pueden ser modificados en el proceso mencionado.

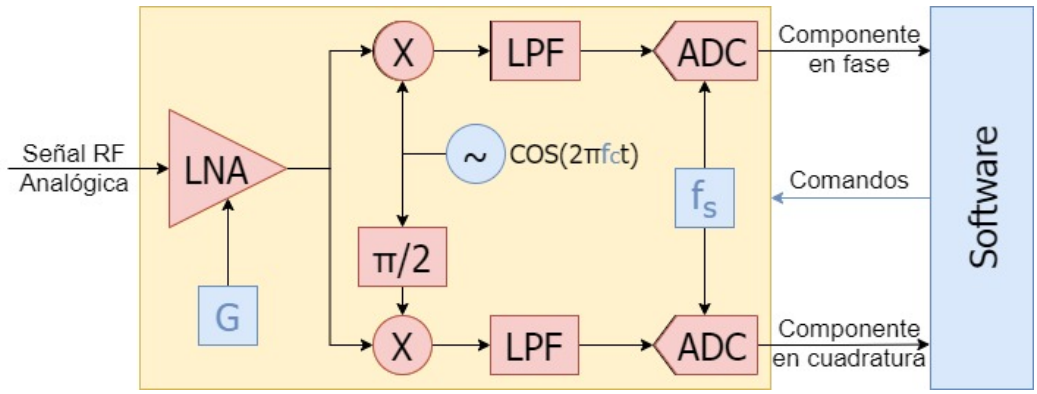


Figura 2.1: Diagrama de bloques del SDR: en amarillo lo perteneciente al hardware; en azul el software y los parámetros que por él se modifican.

2.2. Hardware

El objetivo del dispositivo de hardware es generar en su salida las muestras en tiempo discreto que logren reconstruir en bandabase una sección del espectro del espacio frecuencial, conteniendo casi toda la información necesaria para ser procesada por un algoritmo de DSP (*digital signal processing*, corriendo en capa de aplicación de un sistema operativo). Estos dispositivos, como veremos a continuación, deben cumplir con fuertes requerimientos en términos de precisión y tiempos de latencia ya que están pensados para ser utilizados en tiempo real.

Para esto se utiliza una interfaz serial con la PC de alta velocidad, generalmente USB o Ethernet. La misma debe brindar al usuario un flujo serial de datos (utilizando un protocolo de comunicación pre-definido en sus especificaciones o drivers) para que el usuario sea capaz de procesar digitalmente una señal digital con programas de radio definida por software. Existen diferentes APIs (*application programming interfaces*) o *frameworks* (estructuras de software donde se pueden llevar a cabo distintas funcionalidades personalizables) establecidos en el mercado para ser utilizados en ambientes SDR, muchos siendo de código libre. La sección siguiente describe algunos de estos programas.

Para lograr mayores frecuencias de muestreo de la señal y, por tanto, una mejor reconstrucción de las señales de alta frecuencias muestreadas, los dispositivos SDR utilizan tecnología del estado del arte, caracterizándose por su velocidad de cómputo para levantar datos y llevarlos a la memoria del puerto de comunicaciones que luego los transmite a través de la capa física USB 3.0 (de ancho de banda de hasta 5 Gbps) o USB 3.1 (de hasta 10 Gbps).

El mencionado estado del arte incluye el uso de dos canales (fase y cuadratura) ADC de alto ancho de banda, FPGAs (*field programmable gate arrays*) con grandes poderes de cálculo (capaces de procesar varias entradas digitales en paralelo y en cientos de GPIOs), y microprocesadores que, a diferencia de los procesadores de propósito general, son especialmente fabricados para chequear millones de muestras por segundo y se denominan DSPs (*digital signal processors*). Los componentes nombrados operan en conjunto con otros periféricos integrados, o

ASICs (*application specific integrated circuits*), que son diseñados específicamente para el acondicionamiento de señales y comunicaciones de datos seriales de forma asíncrona y autónoma (es decir, sin necesidad de la atención del microprocesador principal).

2.3. Software

Para que el dispositivo de hardware funcione, debe existir una contra-parte en software capaz de procesar y emitir comandos y respuestas a partir de la interfaz de comunicaciones seriales entre el sistema operativo del PC y el firmware del dispositivo SDR. Generalmente dichas comunicaciones de control son implementadas utilizando el mismo puerto por donde se transfieren los streams de datos.

Los drivers son programas que, desde el PC, interactúan con los periféricos de entrada y salida (I/O) para traducir instrucciones generales de alto nivel a instrucciones específicas de bajo nivel y viceversa. Además de esa traducción, muchos drivers manejan funciones variadas, como puede ser parte del procesamiento de los datos [12]. El *kernel* del sistema operativo posee librerías específicas para dispositivos, por lo que su intercambio con los drivers hace uso de rutinas I/O de esas librerías para tareas como el manejo de memoria y la sincronización a lo largo de múltiples *threads* independientes.

La interacción con SDR no es la excepción, requiriendo el uso de drivers para ejecutar las funciones descriptas y levantar las muestras correctamente del lado del PC. Para esto se han desarrollado diversos productos, tanto de carácter público como privado, de distintas magnitudes. Tienen en común el objetivo de habilitar ambientes basados en radio definida por software.

Un ejemplo de los más utilizados en software libre es SDR++ [13], una plataforma *open source*, con GUI basado en C++, capaz de interactuar con varios tipos de SDR. La misma liberó en agosto de 2021 su primera versión estable non-beta, y cuenta con amplia difusión en la plataforma GitHub. Otro ejemplo es SDRsharp, software también gratuito que se puede integrar al funcionamiento de diversos SDR comerciales por medio de plug-ins [14].

Finalmente, el software GNU Radio también es un medio de uso de SDR, que incluye funcionalidades mencionadas y tiene la ventaja de incorporar, en el mismo ambiente que se levantan las muestras y se efectúa el *threading*, distintas posibilidades de procesamiento de la señal. Este software es el seleccionado para el desarrollo del proyecto, por lo que la sección siguiente se dedica a profundizar en su descripción.

2.4. GNU Radio

GNU Radio contiene al ejecutable GNU Radio Companion, una aplicación de software con una interfaz gráfica que permite la interconexión de distintos bloques de procesamiento de señales de forma accesible. Se conforman archivos de formato .grc que integran una lista de dispositivos de software de DSP (instancias de clase

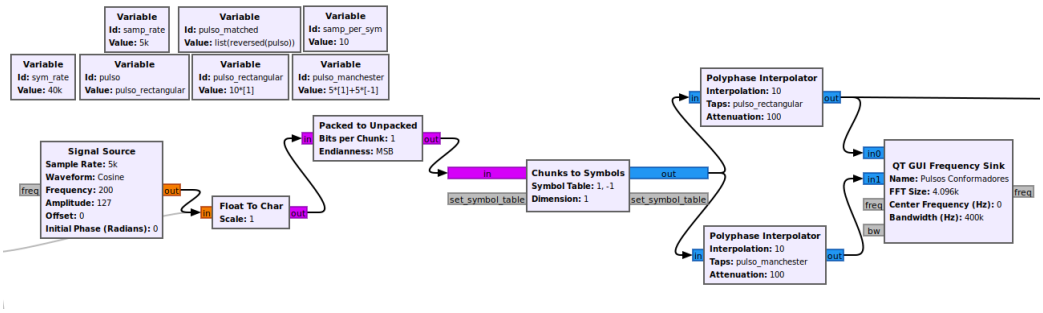


Figura 2.2: Ejemplo de diagrama de GNU Radio Companion a partir del cual se describe un archivo .grc para la comparación de pulsos conformadores.

gr::block) que denominaremos bloques. Estos bloques se deberán adherir a una regla u orden de interconexión con otros pares entrada-salida.

Como en cualquier aplicación de propósito general, los usuarios de GNU Radio se sitúan en un nivel que es abstraído del funcionamiento que sucede debajo de los diagramas de su interfaz gráfica, y solamente se deben encargar de definir e instanciar bloques de procesamiento de señales, interconectándolos visualmente y así generando archivos .grc como el de la Figura 2.2. En un nivel más bajo, los usuarios avanzados pueden hacer uso de la GNU Radio API como *framework* para programar sus propios bloques, *blocks*, de procesamiento de señales en lenguaje C++.

Dentro de los archivos .grc, GNU Radio ejecuta una arquitectura de software que hace uso de un *scheduler* [15], una entidad encargada de controlar el CPU a nivel de aplicación, para ordenar, priorizar, bloquear y agendar las diferentes tareas asignadas por el usuario. Para esto, puede hacer uso de *multitasking* (realizando un multiplexado en el tiempo de las instrucciones del CPU) o *multithreading* (si maneja múltiples núcleos o hilos de ejecución que pueden correr en paralelo).

Para la sincronización de datos en memoria cabe destacar la utilización de una estructura de datos conocida como *circular buffers*, en conjunto con el *scheduler* y herramientas de *multitasking* como *mutexes* y *condition variables*. Los *circular buffers* son diseñados como un FIFO compartido donde, uno a uno, los elementos extremos se van produciendo y consumiendo por dos entidades, productor y consumidor. Cada uno de ellos ve el *buffer* a través de un puntero que va siendo asignado a direcciones que avanzan dentro un espacio de memoria. Estos *buffers* se establecen en secciones de memoria circulares, tales que si el productor o el consumidor debe exceder el extremo superior, se lo redirige hacia el extremo inferior, creando una memoria pseudo-infinita. Así, se evita el *overhead* de transferir datos copiándolos desde la zona del productor a la zona del consumidor. Los accesos de escritura a dicho *buffer* son protegidos por el *scheduler* para realizar accesos exclusivos al recurso compartido y no generar corrupción de datos en el *flowgraph*. Esto se hace utilizando *mutexes*, el productor utiliza *lock* y el consumidor puede ser bloqueado a la espera del *unlock* con las mencionadas *condition variables*.

Para usuarios desarrolladores, GNU Radio propone una API basada en personalizar el código de un método llamado *general_work*, el cual presenta dos impor-

tantes virtudes: la primera es que es declarado con prefijo puramente virtual, y la segunda es que es miembro a una clase llamada *gr::block*. Por tanto, dicha función puede ser heredada desde *gr::block* por cualquier objeto de C++ y el GNU Radio *runtime* se encarga de registrar todos los métodos *work* en la lista de tareas del *scheduler*. El *scheduler* procede introduciéndoles muestras como entrada, lanzándolos a ejecución y esperando su retorno con cantidades de muestras como salida.

Esto implica que el usuario debería, idealmente, hacer todos los procedimientos que necesite hacerle a las muestras de una entrada de datos dentro de la función *general_work*. El objetivo de esta función es ser invocada dentro de un thread registrado a la identidad del bloque heredero de *gr::block* en el *scheduler*. Por lo tanto, tiene que ser vinculada o asignada al mismo. Esto es llevado a cabo por GNU Radio *runtime* para todos los *gr::block* instanciados en un archivo .grc. Para la vinculación entre bloques y el .grc, GNU Radio utiliza (hasta su versión 3.8) la herramienta SWIG [16] que relaciona código fuente C++ con código de Python, estrategia conocida como *bindings* o *wrappers*. Los detalles de su funcionamiento escapan del alcance de este trabajo.Las instancias del bloque en cuestión en un .grc, en conjunto con el código del archivo SWIG y un .xml o .yaml, se fusionan al momento de instalar el bloque en un código *top_block.py* de Python.

Más en detalle, los bloques pueden construirse con diferentes enfoques u orientaciones, por ejemplo, se diferencian los objetos *gr::block* según cómo procesan las muestras y las reglas que restringen la relación entrada-salida que presentan:

- Bloques que procesan iguales cantidades de muestras a su entrada y a su salida, son bloques denominados síncronos.

- Bloques que solamente consumen de su entrada y no devuelven muestras como salidas, son bloques denominados vertederos o *sinks*.

- Bloques que entregan más muestras como salida que la cantidad de muestras que consumen, son bloques denominados interpoladores.

- Bloques con relación entrada-salida editable en tiempo de ejecución que se denominan bloques generales y presentan una característica muy útil: ser capaces de ser invocados con acceso a más cantidad de muestras de lo ordinario (de *noutput_items*) mediante el método *forecast*. Permiten mayores posibilidades en la generación de la salida pero pagan el precio de agregar latencia a todo el *flowgraph*.

En tiempo de ejecución, al comenzar el .grc, se inicializa el *flowgraph* asignando los *circular buffers* por única vez en el espacio de memoria asignado a la aplicación GNU Radio. El *scheduler* informa a los *blocks* individuales, cada uno con su respectivo *thread*, cuánto espacio de memoria puede escribir a su salida como productor y cuántos elementos han sido escritos para consumir a su entrada por *blocks* aguas-arriba. Se les entrega referencias *in* y *out* como punteros a esos espacios en sus respectivos métodos *work* que retornan escribiendo datos en sus

espacios de salida. Se recomienda utilizar al máximo dicho espacio en cada iteración de su *work*, ya que fue probado que hacer esto maximiza el *throughput* del *flowgraph*.

El resultado es un flujo de datos donde parece que todo el sistema funciona en simultáneo y todos los bloques reciben datos y comunican sus resultados continuamente a sus pares, con la debida atención del CPU, ordenada gracias al *scheduler*. Cabe destacar la flexible y escalable arquitectura del *framework* para las distintas variedades de bloques *gr::blocks* que se logran al seguir las órdenes de interconexión generadas por los usuarios en el GNU Radio Companion.

Por último, GNU Radio API y GNU Radio *runtime* permiten la creación de entes asíncronos por fuera del método *general_work* del bloque. Estos son:

- Funciones o handlers de tipo callback que se adhieren a una instancia de bloque *Variable* de GNU Radio Core y pueden ser invocadas al percibir que cambia el valor de la variable,
- Handlers de mensajes PMT asíncronos, que son funciones que pueden ser invocadas en cualquier momento que un bloque suscrito a una entrada de mensajes recibe un mensaje por dicha entrada.

Estos dos entes pueden usarse conceptualmente como comunicaciones interprocesos para manejar la recepción y transmisión de datos entre dos *threads* del *scheduler*. Son funciones ejecutadas de manera asíncrona, independientemente del momento que surgen los eventos que las disparan, son invocadas sin considerar la instrucción que esté realizando el thread transmisor o el thread receptor. Por esto en este trabajo se induce la utilización de un *lock* a un *mutex* en la recepción de cada mensaje PMT y ejecución del *work*, para que se hagan de forma exclusiva y, así, prevenir la corrupción de datos al realizar comunicaciones inter-bloques.

Además de la API, *framework*, clases de C++ y GNU Radio Companion, GNU Radio se instala junto con una interfaz basada en línea de comandos llamada *gr_modtool* que permite al usuario generar, modificar, instalar, registrar, bloques de GNU Radio OOT personalizables y totalmente integrables al GNU Radio Companion. Esos bloques están compuestos por tres archivos fuentes: un archivo .cpp y dos encabezados .h. Uno de los encabezados es correspondiente a la declaración de la clase del bloque en sí y otro que integra al bloque e invoca a una clase padre del mismo, en la cual se hacen las correctas declaraciones del bloque dentro del espacio de nombres de API GNU Radio. De esta manera los usuarios obtienen un esqueleto de software que pueden personalizar como se explicó respecto a la clase *gr::block*.

Capítulo 3

Caracterización de una Señal de Video

La recepción y procesamiento de la señal espiada requieren comprender las emisiones electromagnéticas de los cables. Con este fin, se hace esencial la previa caracterización de la señal transmitida por los cables de video, prestando especial atención a los análisis temporal y espectral de la misma.

El objetivo de este capítulo es llevar a cabo dicho análisis, introduciendo conceptos de la trama de video y describiendo los formatos VGA y HDMI, para luego ver su representación temporal espectral y así poder traducirlo en lo que se espera de la señal detectada.

3.1. Trama de Video

Comprender la trama de video es necesario para caracterizar la señal eléctrica que se transmite para generarla. La imagen se forma a partir de una unidad básica de área: el píxel. Los píxeles se extienden con un cierto alto y ancho, que determinará la resolución del cuadro (conjunto de píxeles desplegados a la vez).

Este cuadro, a su vez, debe actualizarse con una frecuencia suficiente para dar al sistema visual humano la sensación de movimiento. Es decir, simular continuidad para conformar lo que conocemos como video. La frecuencia de refresco de los cuadros (en inglés *refresh rate* o *frame rate*) puede variar según la aplicación particular del video desplegado: desde 24 cuadros por segundo en producciones cinematográficas, hasta frecuencias de 144 Hz (o incluso 300 Hz) en el refresco de monitores comerciales para *esports* o deportes electrónicos. En un punto intermedio reside la frecuencia más común utilizada en monitores y televisores en la actualidad: 60 cuadros por segundo.

Ese funcionamiento implica que el procesador encargado de generar la imagen de video deba transmitir los valores correspondientes a cada píxel en cada cuadro desplegado, proceso que se realiza secuencialmente. Se comienza por el borde superior izquierdo de la pantalla y se recorre de izquierda a derecha hasta completar la línea, para luego seguir por la siguiente línea hacia abajo, también desde la izquierda.

Adicionalmente, y como característica heredada de la televisión analógica, se

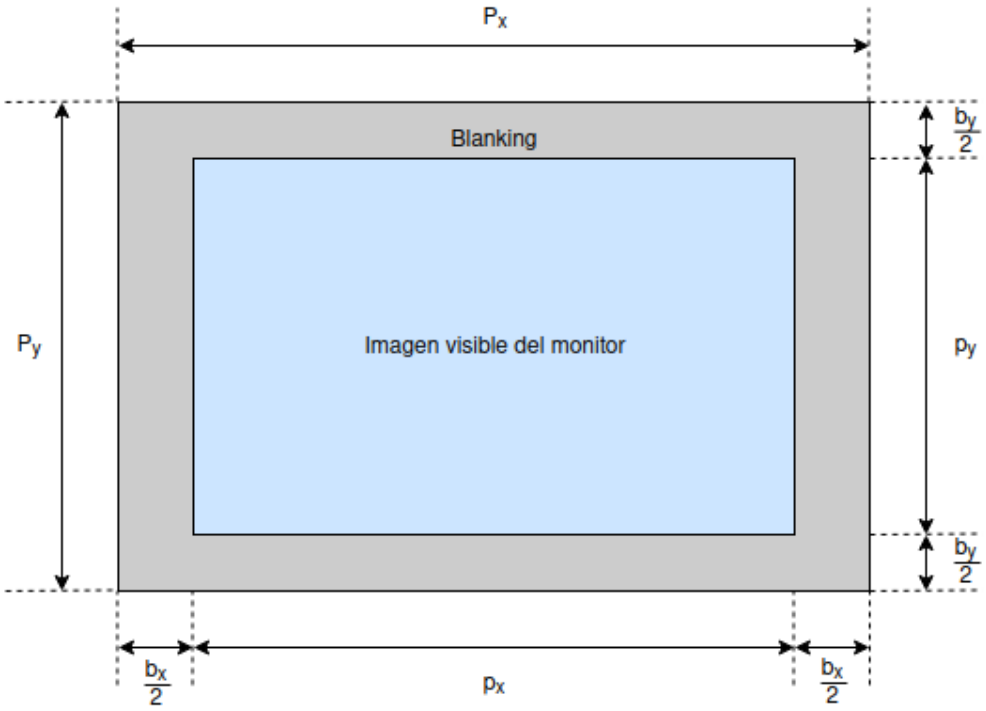


Figura 3.1: Definición de parámetros de trama de video

tienen espacios en blanco utilizados para la sincronización de la imagen. A estos se les llama *blankings* y envuelven todo el contorno del cuadro (ilustrado en Figura 3.1). Esto es, *blanking* horizontal entre que termina una línea y comienza la siguiente, y *blanking* vertical entre que termina un cuadro y comienza el siguiente. Los valores de *blanking* (con energía nula) también son parte de la señal secuencial transmitida y serán de gran utilidad a la hora de procesar la información del espionaje.

3.1.1. Parámetros

El siguiente paso para el estudio del escenario introducido es la parametrización de las magnitudes involucradas. En primer lugar, se tiene la cantidad de píxeles horizontales y verticales de la imagen visualizada p_x y p_y, respectivamente. A partir de la cantidad de píxeles adicionales de sincronismo horizontal y vertical, b_x y b_y, se obtiene la cantidad total de píxeles horizontales y verticales trasmitidos, P_x y P_y, dados por:

$$P_x = p_x + b_x$$

$$P_y = p_y + b_y$$

Cada cuadro está conformado por $P_x \times P_y$ píxeles y se trasmiten a una frecuencia de refresco f_v, por lo que el tiempo de duración en la transmisión de cada píxel es:

$$T_p = \frac{1}{P_x P_y f_v} \tag{3.1}$$

Los parámetros de la trama están estandarizados por la norma VESA [17]. El estándar es generado por *Video Electronics Standards Association* para la compatibilidad entre monitores define reglas para la creación de nuevas resoluciones y los valores de todos los parámetros involucrados en las resoluciones ya existentes.

La notación de la resolución es de la forma $p_x \times p_y@f_v$, quedando definida completamente por sus tamaños vertical y horizontal de imagen, junto con su frecuencia de refresco. El ejemplo de la resolución de interés seleccionada para este trabajo se denota $1920 \times 1080@60$.

Igualmente, existen otros parámetros asociados a cada resolución definida por esos tres valores. Muchos pueden ser calculados a partir de ellos, mientras que otros vienen dados directamente por la norma. Los más notorios son los tamaños de los *blankings*.

La Figura 3.2 fue extraída de la norma y permite visualizar la totalidad de los parámetros asociados a la resolución de interés. Algunos de estos valores se utilizan más adelante, en cálculos relativos a las características temporales de la señal.

3.1.2. Señal VGA

La señal de video VGA se constituye de tres señales con modulación PAM unipolar NRZ. Se trasmite una señal por cada color primario de la luz: rojo, verde y azul. La intensidad de cada color esta dada por el valor de la tensión en el rango de 0 a 0.7 V. Esto puede escribirse con la siguiente expresión donde x_i es el valor de intensidad de ese color en el i-ésimo píxel y $p(t)$ el pulso conformador. $p(t)$ es un pulso NRZ, por lo tanto es un pulso rectangular de ancho tiempo de píxel T_p.

$$x(t) = \sum_{i=-\infty}^{\infty} x_i p(t - iT_p). \tag{3.2}$$

El abordaje de este trabajo se basa en el principio de las emanaciones electromagnéticas generadas por las corrientes eléctricas variables en el cable. La característica de un pulso NRZ implica una tensión constante para valores constantes de píxeles adyacentes con lo cual, para un tramo de píxeles de similar intensidad, el cable no emana ondas electromagnéticas y por lo tanto no podrá ser captada por un sistema de espionaje. Por tanto, la imagen captada será capaz de evidenciar los bordes de las figuras (cambios en el color y por ende en la tensión) pero no su relleno, completando las zonas monótonas con píxeles sin energía. Los resultados vistos en capítulos siguientes serán consecuentes con esta observación y se trata de un asunto a subsanar, dado que dificulta el reconocimiento del contenido de la imagen de video.

VESA MONITOR TIMING STANDARD

Adopted:	11/17/08
Resolution:	1920 x 1080 at 60 Hz (non-interlaced)
EDID ID:	DMT ID: 52h; Std. 2 Byte Code: (D1, C0)h; CVT 3 Byte Code: n/a
Method:	*** NOT CVT COMPLIANT ***
	Per CEA-861 --- 1080p (Code 16) Timing Definition

Detailed Timing Parameters

Timing Name	= **1920 x 1080 @ 60Hz;**						
Hor Pixels	= **1920;**	// Pixels					
Ver Pixels	= **1080;**	// Lines					
Hor Frequency	= 67.500;	// kHz	=	14.8 usec	/	line	
Ver Frequency	= 60.000;	// Hz	=	16.7 msec	/	frame	
Pixel Clock	= **148.500;**	// MHz	=	6.7 nsec	± 0.5%		
Character Width	= **4;**	// Pixels	=	26.9 nsec			
Scan Type	= **NONINTERLACED;**			// H Phase	=	1.4 %	
Hor Sync Polarity	= **POSITIVE**	// HBlank	=	12.7% of HTotal			
Ver Sync Polarity	= **POSITIVE**	// VBlank	=	4.0% of VTotal			
Hor Total Time	= 14.815;	// (usec)	=	550 chars	=	2200 Pixels	
Hor Addr Time	= 12.929;	// (usec)	=	480 chars	=	1920 Pixels	
Hor Blank Start	= 12.929;	// (usec)	=	480 chars	=	1920 Pixels	
Hor Blank Time	= 1.886;	// (usec)	=	70 chars	=	280 Pixels	
Hor Sync Start	= 13.522;	// (usec)	=	502 chars	=	2008 Pixels	
// H Right Border	= 0.000;	// (usec)	=	**0** chars	=	0 Pixels	
// H Front Porch	= 0.593;	// (usec)	=	**22** chars	=	88 Pixels	
Hor Sync Time	= 0.296;	// (usec)	=	**11** chars	=	44 Pixels	
// H Back Porch	= 0.997;	// (usec)	=	**37** chars	=	148 Pixels	
// H Left Border	= 0.000;	// (usec)	=	**0** chars	=	0 Pixels	
Ver Total Time	= 16.667;	// (msec)	=	1125 lines	HT – (1.06xHA)		
Ver Addr Time	= 16.000;	// (msec)	=	1080 lines	= 1.11		
Ver Blank Start	= 16.000;	// (msec)	=	1080 lines			
Ver Blank Time	= 0.667;	// (msec)	=	45 lines			
Ver Sync Start	= 16.059;	// (msec)	=	1084 lines			
// V Bottom Border	= 0.000;	// (msec)	=	**0** lines			
// V Front Porch	= 0.059;	// (msec)	=	**4** lines			
Ver Sync Time	= 0.074;	// (msec)	=	**5** lines			
// V Back Porch	= 0.533;	// (msec)	=	**36** lines			
// V Top Border	= 0.000;	// (msec)	=	**0** lines			

VESA Display Monitor Timing Standard — Version 1.0, Rev. 13
 — Page 89 of 105

Figura 3.2: Definición de los parámetros temporales para la resolución 1920x1080@60. Tomada de la norma VESA Display Monitor Timing Standard Copyright 1994-2013 Video Electronics Standards Association.

3.1.3. Señal HDMI

La estandarización HDMI establece una diferencia clara con respecto a VGA, puesto que pasa a operar con señales binarias. Manteniendo tres canales para transmitir el rojo, verde y azul, cada uno se traduce en valores de 8 bits, totalizando el largo de las palabras en 24 bits por píxel.

Los 8 bits de cada canal, a su vez, son codificados en 10 bits, con lo que la duración de pulso es ahora duración de bits y en lugar de intensidad del píxel se tiene valor de bit. El detalle de la codificación se escapa del alcance de este trabajo, pero sí es relevante destacar que la operativa de HDMI es similar a VGA en su forma de llevar la información de la intensidad de los colores de cada píxel y de *blankings* por los diferentes canales, realizando el barrido de la pantalla en el mismo sentido para las líneas consecutivas.

De la mano de la implementación de la codificación, se destaca que señales que son constantes en la imagen no lo sean en su valor codificado, con lo que el relleno sí podrá ser visto en el espionaje de señales HDMI.

3.2. Representación Temporal y Espectral

En el capítulo anterior se describieron las técnicas de recepción por SDR, donde los parámetros de la frecuencia del oscilador f_c y la frecuencia de corte del filtro pasabajos f_l se definen por software. Tras analizar la lógica detrás de la transmisión de las señales en los cables, se hace necesario describir la señal de video en el dominio de la frecuencia para comprender cómo la selección de los parámetros involucrados, f_c, f_l y $f_s = \frac{1}{T_s}$, permiten la reeconstrucción de la señal. A continuación se realiza el desarrollo del análisis en el dominio de frecuencia basado en el desarrollo de Pablo Menoni [9] para que el trabajo sea autocontenido.

Se comienza con el cálculo de la transformada de Fourier de la señal representada en (3.2), donde $X(f) = \mathcal{F}\{x(t)\}(f)$ y $P(f) = \mathcal{F}\{p(t)\}(f)$.

$$X(f) = P(f) \sum_{i=-\infty}^{\infty} x_i e^{-j2\pi i f T_p} \tag{3.3}$$

Por otro lado, se tiene la DTFT (Discrete Time Fourier Transform) de los píxeles x_i, que se puede expresar como:

$$X_s(e^{j\Omega}) = \sum_{i=-\infty}^{\infty} x_i e^{ji\Omega}. \tag{3.4}$$

$X_s(e^{j\Omega})$ es por definición una función peridodica de periodo 2π. Evaluando la DTFT de x_i en $\Omega = 2\pi f T_p$ se puede reescribir la expresión (3.3) como:

$$X(f) = P(f) X_s(e^{j2\pi f T_p}).$$

Notar que la evaluación de la DTFT realizada es un reescalamiento en frecuencia de la misma (en particular $\Omega = \pi$ corresponde al punto $f = \frac{1}{2T_p}$, $\Omega = 2\pi$ a

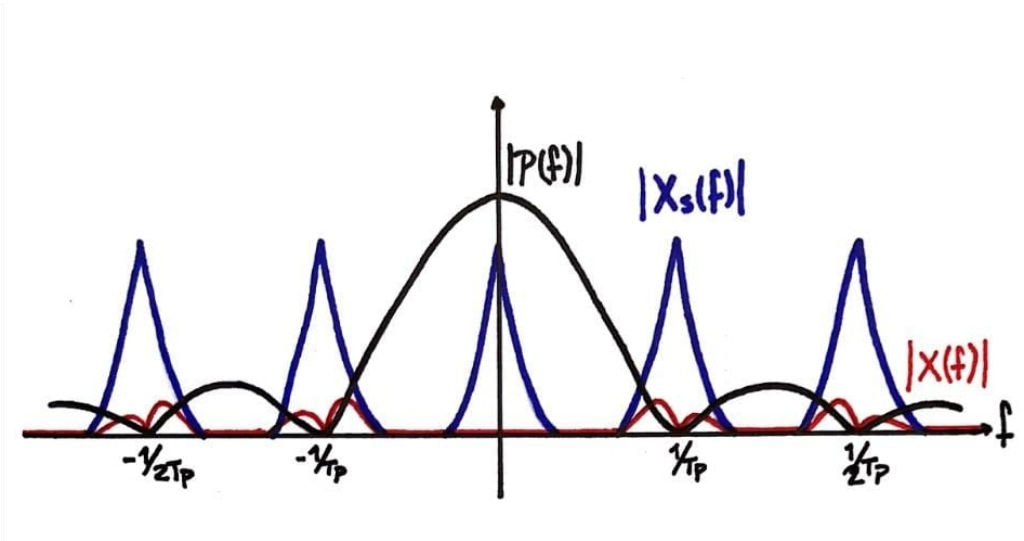


Figura 3.3: Bosquejo del espectro de la señal de video $X(f)$ junto con $X_s(f)$ y el pulso conformador. La componente banda base de $X(f)$ no se representa porque no se propaga.

$f = \frac{1}{T_p}$, etc.). Con este cambio de variable, $X_s(e^{j2\pi fT_p})$ es periódica de periodo $\frac{1}{T_p}$. Para facilitar la lectura, de ahora en más se denominará $X_s(f)$ en lugar de $X_s(e^{j2\pi fT_p})$:

$$X(f) = P(f)X_s(f). \tag{3.5}$$

Considerando al pulso conformador como un pulso rectangular, $p(t) = 1$ cuando $-\frac{T_p}{2} < t < \frac{T_p}{2}$ y 0 en otro caso, se tiene que $P(f)$ es el seno cardinal de argumento fT_p:

$$P(f) = sinc(fT_p).$$

En la Figura 3.3 se representa un bosquejo se las señales en el dominio de la frecuencia. Se puede ver en azul el espectro $X_s(f)$ en conjunto con el espectro del seno cardinal en negro. En rojo, se representa el producto de estos dos, $X(f)$. El espectro $X_s(f)$ es periódico de período $\frac{1}{T_p}$, con concentración de información en las frecuencias bajas. Visualmente, el espectro de $X_s(f)$ se asemeja a triángulos con vértice en las bajas frecuencias. La componente en bandabase de $X(f)$ no se representa ya que no se propaga.

Al observar el espectro de la señal a reconstruir, se concluye que la frecuencia del oscilador utilizado por el SDR de recepción debe ser un múltiplo de $\frac{1}{T_p}$ para bajar la señal a bandabase.

El seno cardinal es una función que se anula en cada múltiplo de $\frac{1}{T_p}$. Por otro lado, el espectro de $X_s(f)$ está centrado justamente en los múltiplos de $\frac{1}{T_p}$. El resultado de estas formas en frecuencia genera que la señal que se propaga y es espiada, $X(f)$, no contenga la información de continua de $X_s(f)$.

La consecuencia para con la propia práctica del espionaje será que los valores constantes no se propaguen y, por ende, sea más difícil reconstruirlos. Es el caso de píxeles consecutivos con igual o similar valor de intensidad, la señal recibida no

contendrá información de su continuidad, haciendo que áreas de un cierto valor de intensidad sean representadas solamente por sus bordes. Además, los bordes serán solo los horizontales, dada la forma secuencial en la que se transmiten las líneas de píxeles. Esto es coherente con lo expresado en la sección 3.1.2, donde se vio que valores constantes de píxeles no emanan ondas electromagnéticas y por lo tanto se dificulta su espionaje.

3.3. Señal Detectada

Como se desarrolló en el capítulo anterior, la recepción de la señal espiada se realiza mediante técnicas de radio definida por software. Esta sección se dedica a analizar la señal detectada por el sistema basado en SDR, considerando el procesamiento dado por esa modalidad.

Se vio que la recepción de la señal consiste en un demodulador centrado en un múltiplo de $\frac{1}{T_p}$, un filtro pasabajos con frecuencia de corte f_l y un conversor analógico-digital a una tasa T_s. Para evitar el aliasing cuando se muestrea la señal la frecuencia de corte del filtro pasabajo debe ser idealmente $f_l = \frac{1}{2T_s}$.

Por lo tanto, la señal recibida puede expresarse como:

$$x_R(nT_s) = \mathcal{F}^{-1}\left\{ X\left(f - m\frac{1}{T_p}\right) rect\left(\frac{f}{\frac{1}{2T_s}}\right)\right\} |_{t=nT_s} .$$

Recordando la igualdad (3.5), es posible reescribirla como:

$$x_R(nT_s) = \mathcal{F}^{-1}\left\{ P\left(f - m\frac{1}{T_p}\right) X_s\left(f - m\frac{1}{T_p}\right) rect\left(\frac{f}{\frac{1}{2T_s}}\right)\right\} |_{t=nT_s}.$$

Dado que $X_s(f)$ tiene período $\frac{1}{T_p}$:

$$x_R(nT_s) = \mathcal{F}^{-1}\left\{ P\left(f - m\frac{1}{T_p}\right) X_s(f) rect\left(\frac{f}{\frac{1}{2T_s}}\right)\right\} |_{t=nT_s}. \tag{3.6}$$

Sea $G(f)$ tal que:

$$G(f) = P\left(f - m\frac{1}{T_p}\right).rect\left(\frac{f}{\frac{1}{2T_s}}\right).$$

Obsérvese que $G(f)$ es el seno cardinal enventanado, con frecuencia de corte $\frac{1}{2T_s}$ y centrado en el armónico m. En otras palabras, es la porción de ancho $\frac{1}{2T_s}$ del espectro del seno cardinal en el entorno al m-ésimo cruce por cero. Con lo visto, la señal recibida queda:

$$x_R(nT_s) = \mathcal{F}^{-1}\{X_s(f)G(f)\}|_{t=nT_s}. \tag{3.7}$$

De esta forma, en recepción se tiene la forma de la señal (3.5) donde, en vez del pulso conformador $P(f)$, se tiene $G(f)$. Realizando el análisis inverso de la sección

3.2 se obtiene la señal de recepción con la forma de (3.2) con $g(t)$ en lugar de $p(t)$:

$$x_R(nT_s) = \sum_{i=-\infty}^{\infty} x_i g(nTs - iT_p). \tag{3.8}$$

De esta última expresión se observa que las muestras x_i recibidas son ponderadas por la función $g(t)$ que es un seno cardinal desplazado y enventanado. Cuanto mayor sea el armónico seleccionado, menor será la amplitud de los lóbulos secundarios del seno cardinal involucrados y más atenuada será la señal recibida, principalmente en las componentes de baja frecuencia.

Hasta el momento, se consideró que el demodulador se sintoniza exactamente en un múltiplo de $\frac{1}{T_p}$. Sin embargo, el SDR y el software no poseen relojes precisamente iguales. A su vez, la norma VESA tiene una tolerancia para valores como T_p. Este efecto se modela con un corrimiento en frecuencia, f_Δ, de la frecuencia $\frac{1}{T_p}$. Considerando que f_Δ varía lentamente, lo cual es esperable, su existencia causa un giro de las muestras en el plano complejo descrito por la siguiente igualdad:

$$x'_R(nT_s) = e^{j2\pi f_\Delta nT_s} x_R(nT_s). \tag{3.9}$$

Recapitulando, se tiene que la señal en recepción contiene la información deseada por el sistema de espionaje x_i, ponderada por la señal $g(t)$ y girando en el plano complejo por el corrimiento en frecuencia. Estos efectos perjudican la visualización de las imágenes, por lo que se busca atenuar sus efectos con el diseño de un filtro ecualizador y un sistema de corrección de frecuencia.

Capítulo 4

Antecedentes Técnicos

El primer capítulo de este trabajo brindó un marco histórico de los antecedentes de TEMPEST. El presente capítulo busca profundizar en los dos proyectos más significativos en el desarrollo de gr-tempest 2.0: TempestSDR, como pionero en la inclusión del paradigma SDR en TEMPEST y, naturalmente, la versión original de gr-tempest.

4.1. TempestSDR

TempestSDR, desarrollado por Martin Marinov, es la primera implementación de espionaje por emisiones electromagnéticas mediante el uso de SDR. En ella se utilizan conceptos teóricos previos como los vistos en el Capítulo 3, junto con los beneficios del uso de SDR explicados en el Capítulo 2.

El software fue diseñado en el lenguaje de programación C y se ejecuta mediante una interfaz gráfica de usuario (GUI) basada en Java que permite interactuar con sus varias funcionalidades.

Este tipo de desarrollo se hace fácil de utilizar por usuarios finales, puesto que se cuenta con el ejecutable muy intuitivo que integra todo el aplicativo. Sin embargo, tiene dos problemáticas inherentes al mismo: mantenimiento y extensión. Como se verá a continuación, el programa es desarrollado desde cero y depende de sus propias librerías y *plug-ins* de interacción con hardware, con lo que todas las actualizaciones y cambios externos deben ser agregados manualmente en lugar de depender de un *framework*.

En esta sección se hace un resumen del funcionamiento del software, pasando por cómo recibe y procesa la señal para su despliegue, para luego hacer énfasis en los temas de interés para este trabajo: la detección de la sincronización de los bordes de la imagen y la detección o inferencia de la resolución espiada.

4.1.1. Funcionamiento

El sistema consiste en dos partes: una librería (TSDRLibrary) con las funcionalidades de su proyecto, y un programa *host* que interactúa con la librería para

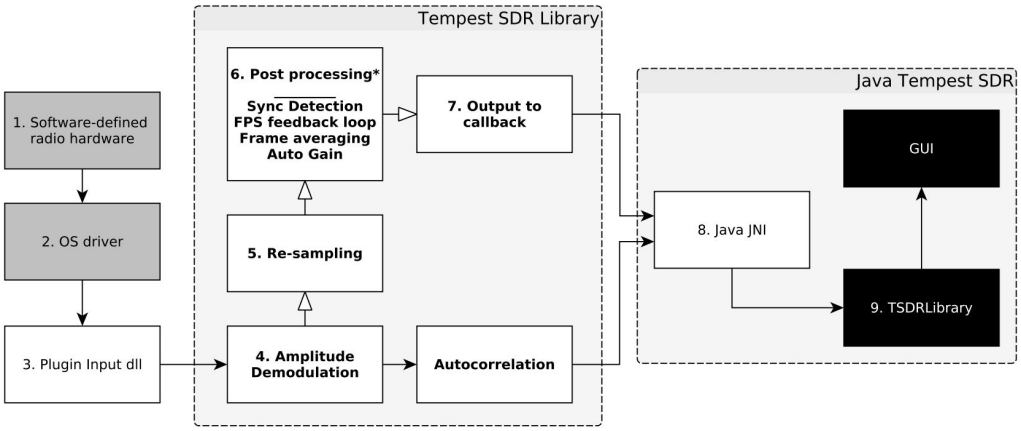


Figura 4.1: Procesamiento usando el sistema con GUI de Java. El fondo blanco representa código en C y fondo negro en Java, mientras que en gris están los procesos de hardware externo. Tomado de [8]

controlar esas funcionalidades y presentar el video espiado - en este caso, el programa en Java mencionado con la GUI. Más allá de los *drivers* correspondientes al SDR utilizado, el sistema no depende de librerías de terceros; esto la hace liviana y fácil de compilar.

La tarea del *host* es simple: armar y correr la librería. Cuando hay un cuadro de video disponible, la librería es cargada para generar una instancia de procesamiento, establecer la resolución y frecuencia de actualización (modificables en tiempo de ejecución), cargar los *plug-ins* pre-compilados y comenzar la ejecución de las funciones de la librería.

El flujo de los datos es mostrado en la Figura 4.1 y describe el procesamiento en un sistema operando en vivo. Con la numeración de la figura, se recorren los siguientes pasos:

1. Los datos de radiofrecuencia son convertidos en muestras complejas por el dispositivo SDR y enviadas al PC.

2. El driver de hardware recibe y encola las muestras.

3. Un *plug-in* haciendo *polling* del driver recibe un bloque de datos y lo envía a la TSDRLibrary.

4. A las muestras complejas se les pide la magnitud (para no tener que lidiar con el error en frecuencia) y son transmitidas tanto al corrector de muestreo como al calculador de autocorrelación. A este último se le avisa, además, si se perdieron o descartaron muestras, de manera de volver a comenzar su cálculo.

5. Se hace la corrección de muestreo para igualar la frecuencia de muestreo a la esperada para la resolución.

6. En este punto se realizan algunos procesamientos de la señal para mejorar la señal recibida. La detección de sincronización se usa primero para centrar la imagen en el display. Luego un pasa-bajos en el dominio del tiempo (o *motion blur*) para reducir el ruido, y finalmente un corrector de la ganancia para asegurar que todo el rango dinámico de la señal es utilizado para desplegar la imagen.

7. Los cuadros se pasan al *host*.

8. En este ejemplo se pasa al Java Native Interface (JNI) que lo pasa a un formato manejable por la maquina virtual.

9. La instancia generada para TSDRLibrary recibe los datos y los pasa al GUI para desplegar la imagen.

Los detalles de este flujo tienen una complejidad significativa y, en varios puntos, de poca relevancia para este trabajo, mientras que en otros contribuyen a las soluciones requeridas para la implementación en GNU Radio. En las secciones siguientes se presentan esos desarrollos.

La Figura 4.2 muestra el resultado del funcionamiento explicado. La GUI incluye la pantalla de despliegue de la señal espiada y el panel de parámetros para la interacción con el software, que contiene la resolución, movimiento manual de la ubicación de la imagen, frecuencia de muestreo y selección del *frame rate* a partir de la autocorrelación, entre otros.

4.1.2. Detección de Sincronización

Es natural que, aún resolviendo los problemas relacionados a la corrección de muestreo, la imagen recibida no tiene por qué estar sincronizada con los márgenes provistos para el despliegue de la imagen - de hecho, probablemente no lo esté. Se desea, por tanto, encontrar los márgenes laterales o *blankings* que servirán de referencia para hacer el ajuste correspondiente.

La manera más simple de caracterizar estos *blankings*, propone Marinov, es que no implican ningún tipo de actividad en relación a la zona donde se espera que haya video. Entonces, según cómo se la analice, deberá tener mayor o menor energía que la zona de video. Por esto se considera que la intensidad media del intervalo de *blanking* será muy distinta a la de la región de video.

A partir de esta noción, y teniendo en cuenta que se tiene tanto un *blanking* vertical como uno horizontal, es posible detectar sus ubicaciones separando la búsqueda en dos dimensiones: se genera un arreglo horizontal que contiene el promedio de todas las filas de un cuadro, así como uno vertical con el promedio de todas las columnas del mismo cuadro. Estos promedios deberían variar sus valores en las zonas de video pero su energía media en la ubicación del *blanking* debería ser consistente.

Para cualquiera de estas líneas, ya sea fila o columna, consideremos que se tienen n elementos $b[k]$, donde n puede corresponder a P_x o P_y. Estos se dividen

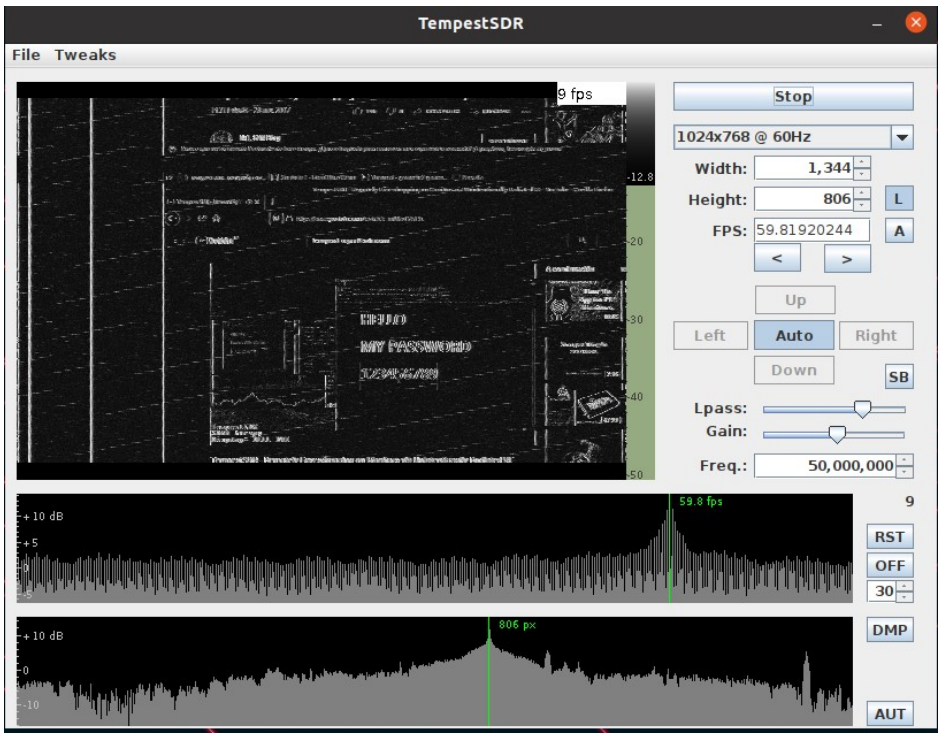


Figura 4.2: GUI y salida de TempestSDR.

en dos regiones de longitudes 2ω (correspondiente al *blanking*) y $n - 2\omega$ (correspondiente al video) centradas en c y $n - c$, respectivamente. Nuevamente, y para remitirse a la nomenclatura utilizada, el valor 2ω puede corresponder a b_x o a b_y y el valor $(n - 2\omega)$ a p_x o p_y, según si se trata del *blanking* horizontal o vertical.

Luego, la diferencia media estará dada por:

$$\beta(c) = \left(\sum_{k=c-\omega}^{c+\omega} \frac{b[k \bmod n]}{2\omega} - \sum_{k=2\omega-c}^{2(n-\omega)-c} \frac{b[k \bmod n]}{2n - 4\omega} \right)^2 \tag{4.1}$$

El problema se encuentra, entonces, en hacer corresponder la ubicación de c con la posición del *blanking*. Esto es análogo a encontrar el valor de c que maximice β en la ecuación (4.1). La Figura 4.3 muestra cómo se ven los parámetros mencionados para los datos correspondientes a un cuadro.

La metodología implementada por Marinov para hallar el escenario en el que se maximiza es independiente de si se trata del arreglo vertical u horizontal, y consiste en fijar el tamaño esperado del *blanking* (conocido dada una cierta resolución) y recorrer el arreglo calculando β para cada posible ubicación del *blanking* c. De esta forma se devuelve el valor de c con mayor β.

Luego, dadas las ubicaciones de *blankings* encontradas (valores de c para los arreglos vertical y horizontal), se reordena el cuadro para desplegarlo en tiempo real con la alineación apropiada. Es decir, sin cortar el video (ver Figura 4.4).

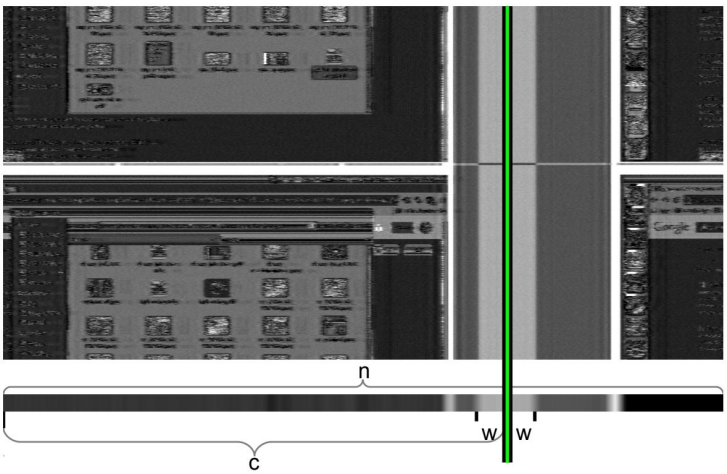


Figura 4.3: Definición de c, ω y n, en este caso con la relación que maximiza β. Tomado de [8].

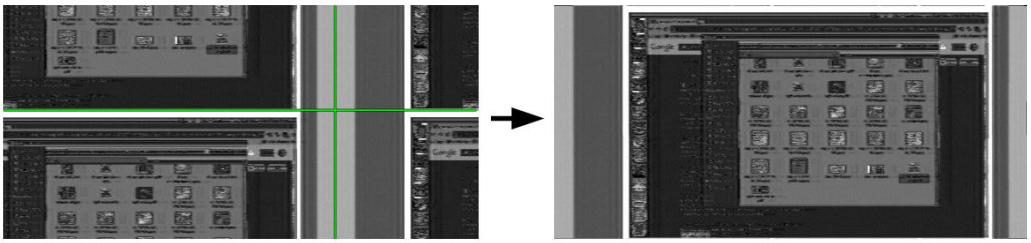


Figura 4.4: Para los c dados para los arreglos horizontal y vertical, se alinea el cuadro para centrar el video. Tomado de [8]

4.1.3. Detección de Resolución

El procesamiento en varios de los puntos desarrollados asumen que la resolución y frecuencia de refresco de la imagen utilizada por el monitor espiado son conocidos. Este no tiene por qué ser el caso, por lo que es apropiado estimar estos datos a partir de algunas características de la señal recibida.

Los cuadros sucesivos de la imagen en la señal se repite con una frecuencia de refresco que se desea estimar. Una forma de estimarla es a través del cálculo de la autocorrelación discreta entre las muestras:

$$R_{vv}[j] = \sum_{i=-\infty}^{\infty} v_i \bar{v}_{i-j}$$

en donde v representa las muestras discretas y $\overline{v}$ denota el conjugado de las mismas.

La autocorrelación es una medida de similitud de la señal consigo misma desplazada: R_{vv} será mayor en las distancias j para las que la coincidencia sea mayor.

La persistencia temporal de las imágenes de video es un supuesto válido en el despliegue de las imágenes de video [18]. Por este motivo, la similitud de la señal será máxima (tendrá un pico) cuando se correlaciona un cuadro con el siguiente. El inverso del tiempo transcurrido entre un cuadro y el siguiente es, por definición,

la frecuencia de refresco, con lo que la distancia encontrada entre los picos (la distancia en tiempo entre los cuadros) resulta en ese valor de interés.

$$f_v = \frac{1}{t_{peak}} \tag{4.2}$$

Es importante notar que la ecuación (4.2) es aplicable en tanto la medida del pico esté dada en tiempo. En el cálculo de autocorrelación para un j dado, su valor temporal depende de la frecuencia de muestreo: si se duplica la frecuencia de muestreo, el tiempo en el que se adquiere la j-ésima muestra es la mitad. Por eso se requiere la conversión apropiada para hallar los tiempos de pico a utilizar en cálculos de frecuencia.

Para la estimación de la cantidad de lineas en un cuadro, se utiliza la naturaleza de la repetición de las líneas: cada linea tendrá una gran similitud con la siguiente. Por lo tanto, la autocorrelación ya descrita presenta picos (menores, pero aún notables) cuando se superponen una línea con las de la siguiente.

El tiempo transcurrido entre un pico y el siguiente (Δt) permite el cálculo del alto de la pantalla espiada con la siguiente relación:

$$P_y = \frac{1}{\Delta t f_v} \tag{4.3}$$

Finalmente, los datos obtenidos son suficientes para identificar por completo la resolución, por medio de la búsqueda en la tabla de *VESA modes*: ingresando el tamaño vertical y la frecuencia de refresco se puede describir una única resolución, con lo que el resultado es la resolución horizontal y los correspondientes tamaños de *blanking*.

4.2. gr-tempest

Se mencionó al comienzo la oportunidad detectada de volver a implementar el software TempestSDR, pero en un *framework* libre que facilite los principales aspectos vistos en la sección anterior: el mantenimiento y extensión del código.

En este contexto es que se comienza a trabajar en una implementación de GNU Radio, gr-tempest, buscando implementar las mismas funcionalidades pero sin atarse a los mismos métodos para ejecutarlas.

La publicación inicial del trabajo es explicada a continuación y cubre parcialmente las funcionalidades incluidas en el original, pero logra una versión operativa capaz de espiar en tiempo real.

4.2.1. Funcionamiento

Una particularidad del GNU Radio Companion (GRC) es que hace muy notorio el flujo de los datos procesados: la herramienta opera de interfaz gráfica para GNU Radio, mostrando en pantalla los bloques fuente del programa y los de módulos OOT como, en este caso, gr-tempest. Un diagrama simple a la vista permite

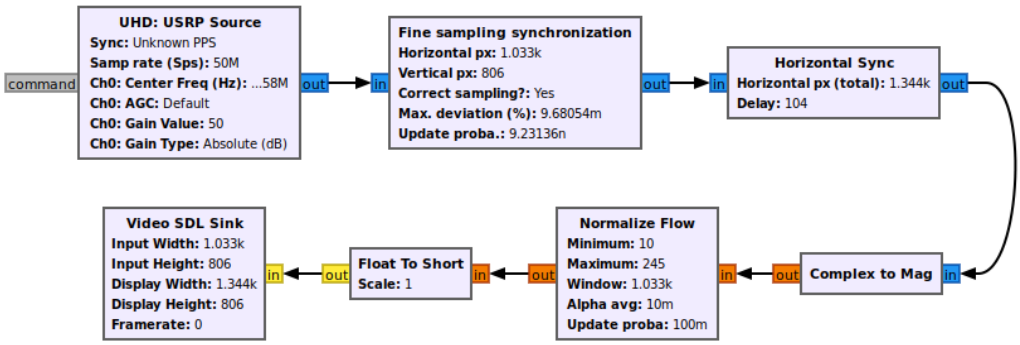


Figura 4.5: Diagrama de recepción de gr-tempest original con corrección de muestreo automático.

distinguir los procesos que se realizan sobre la señal recibida y el orden en el que se hacen.

El punto de partida de gr-tempest para este proyecto incluye varios bloques del módulo OOT funcionales. La Figura 4.5 muestra uno de los archivos GRC que describe un diagrama utilizable en el espionaje de monitores.

Como se pude ver, el proceso comienza con la recepción de la muestras de la señal por parte del bloque que opera como driver del SDR. Luego, se realiza la corrección de muestreo y la alineación horizontal de la imagen, para finalmente normalizar los módulos de las muestras y desplegarlos en pantalla.

A continuación se resume el funcionamiento de cada uno de los bloques del módulo, incluyendo su finalidad y metodología. Por detalles referirse a la documentación del código [10].

- **Fine Sampling Synchronization:** Bloque que recibe todas las entradas permitidas por el *scheduler* y los utiliza para hallar el pico de correlación con respecto al entorno de la próxima línea. También los almacena (vía *set history*) para acceder a los datos de todo un cuadro y calcular la máxima correlación con respecto al cuadro siguiente. Utilizando el valor relativo del pico encontrado, obtiene la cantidad de muestras que entran actualmente en un cuadro completo, esto puede utilizarse para calcular un estimador de la corrección que debe aplicarse al tiempo de muestreo para lograr alinear la cantidad de muestras de un cuadro con la cantidad de píxeles por muestra que se están recibiendo al momento. Al estimador se le hará referencia como índice de interpolación o *ratio* y se usa para interpolar la señal recibida y corregir el muestreo.

- **Hsync:** Bloque que recibe la señal con el muestreo corregido, ejecuta una búsqueda de la línea vertical (correspondiente al límite entre la imagen de video y el *blanking* vertical izquierdo) lo utiliza para devolver la imagen sincronizada horizontalmente.

- **Normalize flow:** Bloque que toma algunos valores máximos y mínimos de

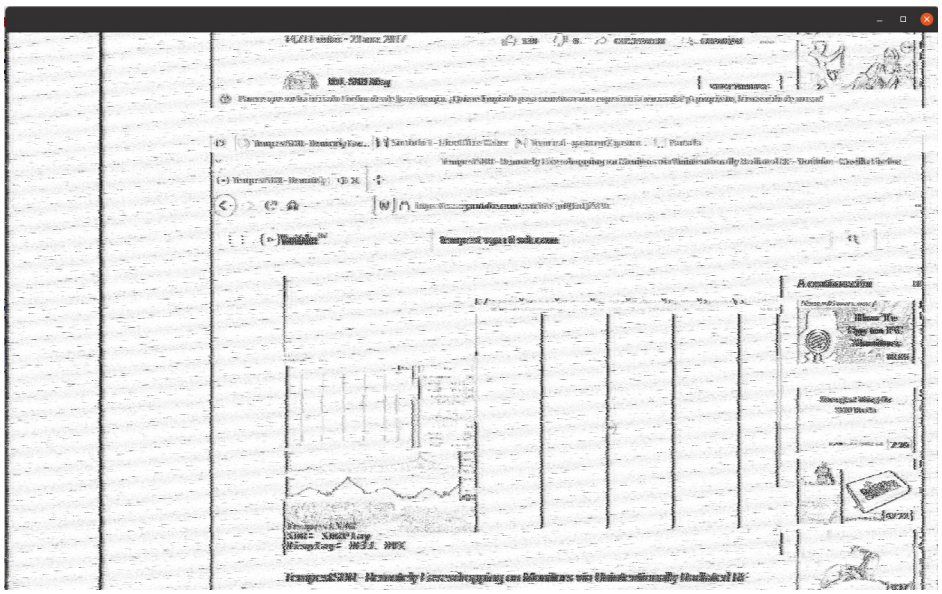


Figura 4.6: Salida de video con la versión original de gr-tempest.

los elementos que recibe y los utiliza para normalizar los módulos de todos los datos que ingresan.

- **Framing:** No incluido en el sistema mostrado, este bloque determina si cada linea de la salida corresponde a una línea de la entrada o a una línea de ceros, según la referencia del tamaño vertical. Se utiliza en caso de no conocer la resolución de la imagen espiada, para limitar el tamaño de la salida al del bloque de despliegue puesto que este último no es capaz de adaptar sus parámetros en tiempo de ejecución.

- **Sampling Synchronization:** No incluido en el sistema mostrado, este bloque es una versión manual del *Fine Sampling Synchronization*. Realiza la misma correlación de una línea con la siguiente pero el remanente se corrige manualmente por el usuario en la GUI.

El resultado de este sistema, que además invierte los valores de las muestras (con respecto a la salida de TempestSDR) para que mayor intensidad de los píxeles se traduzca en negro sobre el fondo blanco, devuelve una salida como en la Figura 4.6. La misma da una noción del estado funcional de los bloques mencionados: el muestreo es corregido correctamente para que las muestras se desplieguen acorde las líneas del cuadro, mientras que el posicionamiento horizontal de la imagen deja al borde izquierdo del *blanking* vertical sincronizado con el borde izquierdo del *display*.

El sistema incluye una GUI de GNU Radio para modificación de ciertos parámetros en tiempo de ejecución según el escenario (diagramas más automatizados requieren la modificación de menos parámetros) que se ejecuta en paralelo a la salida de video del bloque *Video SDL Sink*.

4.2.2. Puntos de Mejora

La versión de gr-tempest a partir de la cual se comienza el desarrollo de este proyecto tiene una operativa funcional que, efectivamente, procesa la señal recibida y permite ver una imagen distinguible a la salida. Presenta, sin embargo, algunos puntos de mejora sobre los que trabajar:

- El bloque *Hsync* realiza correctamente la sincronización horizontal de la imagen recibida pero no alinea verticalmente lo que se desea mostrar.
- El bloque *Fine Sampling Synchronization* realiza sus operaciones para todas las muestras recibidas y esto incluye la de interpolación, que es de especial interés puesto que incrementa el procesamiento requerido por el equipo. De esta forma se limita la frecuencia de muestreo que puede tolerar el procesador, no alcanzando tasas de muestreo altas en el funcionamiento online.
- El mismo bloque tiene una limitación en el constructor: no ajusta la corrección de muestreo a menos que se vuelva a construir manualmente mediante el cambio de algún parámetro en tiempo de ejecución.
- No se cuenta con una función de inferencia de resolución, sino que funciona únicamente mediante la selección manual de características de la señal.
- No se cuenta con una función que busque mejorar la imagen a mostrar. Como agregado al proceso de corregir el muestreo y centrar la imagen para desplegarla, se puede implementar un sistema de ecualización.

Estos puntos dan lugar a profundizar el trabajo ya realizado y sobre ellos se enfoca el proyecto.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Implementación de Hardware

El objetivo de este capítulo es definir los requerimientos del sistema utilizando la caracterización de la señal, para luego diseñar una configuración de hardware para el funcionamiento deseado y seleccionar los componentes específicos en cada caso.

5.1. Requerimientos del Sistema

El estudio bibliográfico de los trabajos previos dio lugar a las siguientes consideraciones a la hora de definir los requerimientos del sistema:

- Señal de video VGA
- Se selecciona la resolución denominada 1080p. A continuación se muestran algunas de sus particularidades (tomadas de la norma, tal como en la Figura 3.2):
  - Resolución desplegada en pantalla 1920x1080 @ 60 Hz
  - $H_{blanking} = 280px$
  - $V_{blanking} = 45px$
  - $t_{px} = 6.73$ ns $\pm$ 0.5 %
  - $t_{line} = 2200px \times t_{px} = 14.81$ μs
  - $t_{frame} = 16.7$ ms
- Frecuencia de píxeles: $f_p = (1920px{+}280px)(1080px{+}45px)60Hz = 148{,}5MHz$

A partir de la transmisión cableada entre la PC y su monitor se generan, debido a las emisiones electromagnéticas, campos que son proporcionales a la señal transmitida a través del cable.

La señal emitida al aire mantendrá la mayoría de las características de la señal que viaja por el cable, como su espectro y su frecuencia fundamental, pero

presentará atenuaciones debidas al canal inalámbrico que dependen de la distancia entre el receptor y la fuente de emisión.

Las siguientes características refieren a la señal emitida a través del aire, la cual contiene la información de interés:

- Frecuencia fundamental de la señal emitida: es igual a la frecuencia de píxeles $f_p = 148{,}5MHz$
- Espectro en bandabase: aproximadamente igual al espectro de la señal PAM generada dentro del cable VGA. Lo que interesa para la elección del hardware es que decae como un seno cardinal.
- Número de armónico: cada múltiplo del primer armónico tiene información útil de la señal pero a niveles de potencia decrecientes. Esto es debido a la atenuación del espectro con su forma de seno cardinal como se expresó en la ecuación (3.8).
- Ancho de banda: 50 MHz.
- Ruido: al minimizar el ancho de banda de la banda pasante del sistema, se minimiza la potencia de ruido a la entrada.

Finalmente, reuniendo todas las consideraciones a cumplir, se establece la siguiente lista de requerimientos que debe satisfacer la etapa previa al SDR de acondicionamiento de la señal de radiofrecuencia (que se denominará RF front-end):

1. Para minimizar la potencia de ruido y maximizar la relación señal a ruido (SNR), se requiere que el hardware tenga un filtro pasabanda de radiofrecuencia tal que el armónico seleccionado de la señal recibida se encuentre en el centro de la banda pasante.
2. Se requiere amplificar la banda pasante con mínima distorsión del espectro de la señal de interés y que la potencia de ruido equivalente a la entrada del conjunto de filtro-amplificador minimice la potencia de ruido a la entrada del dispositivo SDR.
3. El número de armónico seleccionado da un grado de libertad al seleccionar el centro de la banda pasante, sin perder de vista que la relación señal a ruido cae muy rápidamente al aumentar la frecuencia. Esto permite elegir un armónico en el que los filtros comerciales se ajusten de la mejor manera.
4. Para el ancho de banda, se necesita que al menos se utilicen los 25 MHz al funcionar a $f_s = 50$ MHz. Además, se requiere que la señal de interés esté suficientemente lejos de las frecuencias de caída 3 dB de la cascada RF. Esto implica tener una banda pasante de al menos 100 MHz en pasabanda, dejando la caída 3 dB a una octava por arriba de la señal.
5. Antena direccional que sea lo suficientemente selectiva a la banda pasante determinada por los otros componentes.

6. No se debe saturar la entrada del SDR. Esto sucede si se supera el máximo de potencia a la entrada del SDR de 0 dBm, según indica el fabricante.

Adicionalmente, entraron en consideración otros indicadores de rendimiento del RF front-end. En primer lugar, la figura de ruido o NF (Noise Figure), que es la degradación de la relación señal a ruido provocada por el o los dispositivos en la cascada de radiofrecuencia. En segundo lugar, la relación de matcheo de impedancias o SWR (Standing Wave Ratio). Ambos indicadores se pueden calcular a la entrada y a la salida de cada componente y para el total de la cascada de componentes.

$$F_T = F_0 + \frac{F_1 - 1}{G_0} + \frac{F_2 - 1}{G_1 G_0} + \frac{F_3 - 1}{G_2 G_1 G_0} + ... + \frac{F_{n-1} - 1}{G_{n-2}...G_2 G_1 G_0} \tag{5.1}$$

En el caso de la figura de ruido, se distingue la relación planteada en la ecuación (5.1), de la cual se deriva el NF total de un sistema de n dispositivos en cascada, NF $= 10log_{10}(F_T)$, siendo F_i la contribución al NF de cada etapa del sistema. Se observa que las contribuciones a la figura de ruido por las etapas posteriores a la del primer componente son atenuadas por el efecto de la ganancia de dicho componente [19]. Para minimizar el SWR total y tener el NF más cercano a 0 dB, se debe colocar primero el dispositivo que aporta mayor ganancia y menor SWR, seguido de los demás. Esto determinó que el RF front-end comience con la antena, continúe con el amplificador y finalice con el filtro pasabanda.

En la Figura 5.1 se puede ver el diagrama de bloques planificado en base a los puntos mencionados.

Para el diseño de los parámetros del sistema, entra en juego la elección del armónico a espiar para la resolución 1080p - principalmente al momento de elegir el filtro pasabanda. El armónico seleccionado será el segundo y esto determina que el filtro pasabanda debe estar centrado en $f_c = 2 \times f_p = 2 \times 148{,}5$ MHz $= 297$ MHz. La selección se debe a que, como se ve en la Figura 5.2, la recepción del primer armónico sufriría interferencias debido a las frecuencias de las señales de *broadcasting* [20]. En el tercer armónico, además de tener el mismo inconveniente, la potencia de la señal espiada sería significativamente más baja.

5.2. Diseño de Componentes

En esta sección se definen las características de los dispositivos a utilizar para el RF front-end desarrollado anteriormente.

En primer lugar, se define el componente principal de este trabajo, el SDR de Ettus-Research USRP B200mini como el mostrado en la Figura 5.3. Esta elección satisface el requerimiento de funcionamiento a 50 MSps, además de tratarse del dispositivo disponible en el Instituto de Ingeniería Eléctrica.

Este componente se conecta a través de USB 3.0 al ordenador y lleva a cabo el trabajo de recepción de ondas de radiofrecuencia (pasabanda), generando las muestras digitales de estas ondas en fase y cuadratura (bandabase). A continuación, se presentan algunas de sus características:

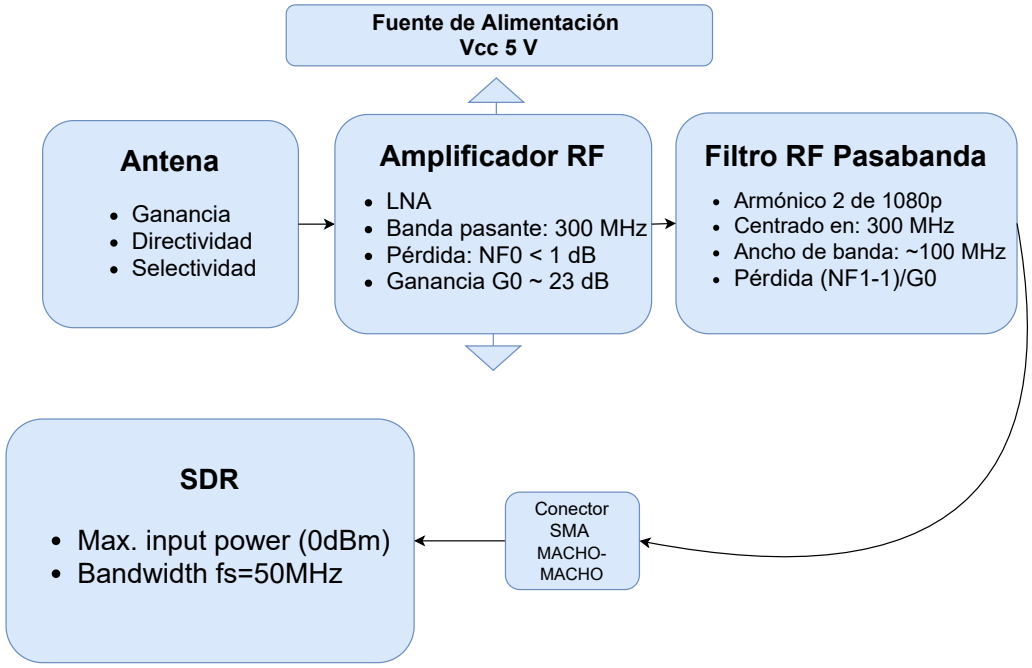


Figura 5.1: Diagrama de bloques del Hardware planificado.

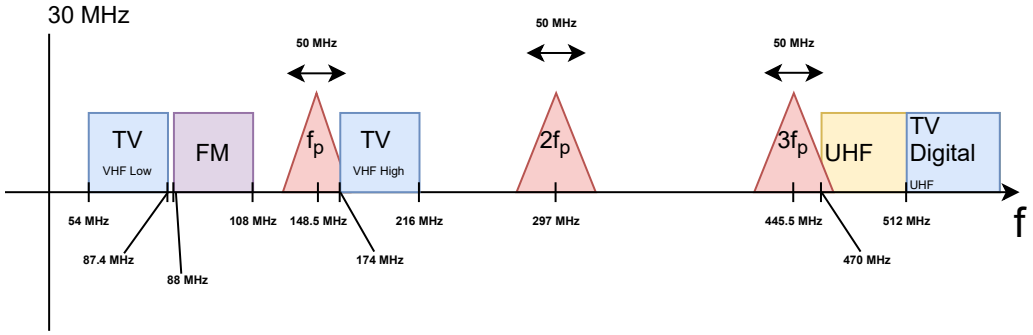


Figura 5.2: Bandas del espectro radioeléctrico, cercanas a las frecuencias de interés.

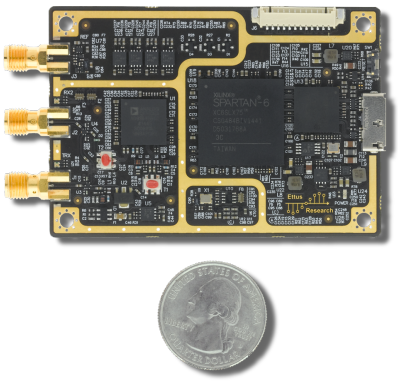


Figura 5.3: El SDR de Ettus-Research USRP B200mini.

- Marca Ettus Research .
- Modelo USRP B200mini [21].
- FPGA Xilinx Spartan-6 XC6SLX75 [22].
- Analog Devices AD9364 RFIC direct-conversion transceiver [23].
- Full duplex, SISO (1 Tx, 1 Rx). Tres puertos SMA-hembra: referencia/Rx/Tx [24].
- Permite bajar a banda-base las señales de radiofrecuencia en el rango de 70 MHz a 6 GHz.
- Tiene una máxima frecuencia de muestreo de 56 MHz. Satisface el requerimiento de obtener una muestra cada 50 MHz.
- Máxima Potencia a la entrada 0 dBm.

En la búsqueda de mercado no se encontraron filtros pasabanda RF sintonizados alrededor de 300 MHz con el ancho de banda necesario de al menos 100 MHz. Esto llevó a cambiar este componente por una cascada de un filtro pasa-bajos o LPF (Low Pass Filter) con un filtro pasa-altos o HPF (High Pass Filter).

Además, lo visto de NF deriva en el requerimiento de un amplificador especializado para radiofrecuencia y de bajo ruido, que tenga NF cercano a 0 dB y cumpla $SWR_{IN}(f) < 2$ dB y $SWR_{OUT}(f) < 2$ dB.

Convenientemente, se tenía a disposición un filtro LPF y un amplificador LNA con las características deseadas, y que por tanto fueron incorporados al RF front-end. Esto dejó pendiente la adquisición de un HPF que cumpliera con el requerimiento de f_c y ancho de banda al ser emparejado con el LPF mencionado, además de una antena.

Para la selección y compra del filtro, acudió a la empresa Mini-Circuits [25] como proveedora de componentes y el software QUCS como método de simulación del hardware de radiofrecuencia. A partir de las existencias disponibles en la primera, se listan los componentes a simular:

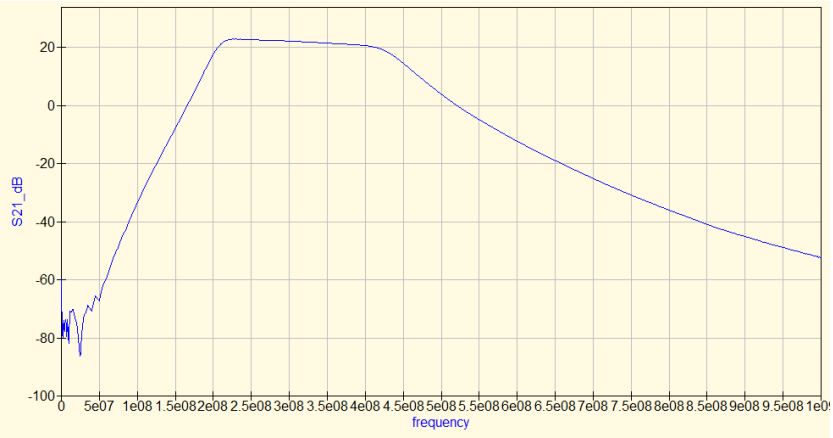


Figura 5.4: Simulación software QUCS LNA-LPF-HPF

1. Antena monopolo.
2. MINI-CIRCUITS LNA ZX60-P103LN+ [26].
   - Ganancia de mínimo 23 dB a 18 dB entre 50 MHz y 500 MHz, respectivamente.
   - NF = 1.2 dB a 0.4 dB entre 50 MHz y 500 MHz, respectivamente.
   - $SWR(f) < 2$ dB entre 50 MHz y 500 MHz.
   - Requiere fuente de alimentación externa DC 5V@95mA típicos.
3. MINI-CIRCUITS SMA male - SMA male Connector.
4. MINI-CIRCUITS HPF SHP-250+ [27].
5. MINI-CIRCUITS LPF SLP-450+ [28].

Las Figuras 5.4, 5.5 y 5.6 muestran los resultados de simular la transferencia y el SWR de entrada y salida, respectivamente.

Se puede ver que se cumplen las condiciones $SWR_{IN}(f) < 2dB$ y $SWR_{OUT}(f) < 2dB$. Además, como se en la Figura 5.4, la transferencia se ajusta correctamente a un centro de 300 MHz y ancho de banda de 100 MHz.

Luego de que las simulaciones verificaran la transferencia en frecuencia, se concretó la compra del filtro 4. HPF SHP-250+ junto con el conector 3. SMA macho - SMA macho.

La antena directiva fue el único inconveniente de las compras de hardware. Se determinó que no hay oferta en el mercado para antenas en las bandas cercanas a 300 MHz. Esta banda de frecuencias, denominada *NATO B band*, es utilizada para comunicaciones militares. Por este motivo, y dado que no es posible cambiar de armónico por lo visto en la sección anterior, se decidió adquirir una antena monopolo centrada en 315 MHz, ya que abarca correctamente la banda de interés

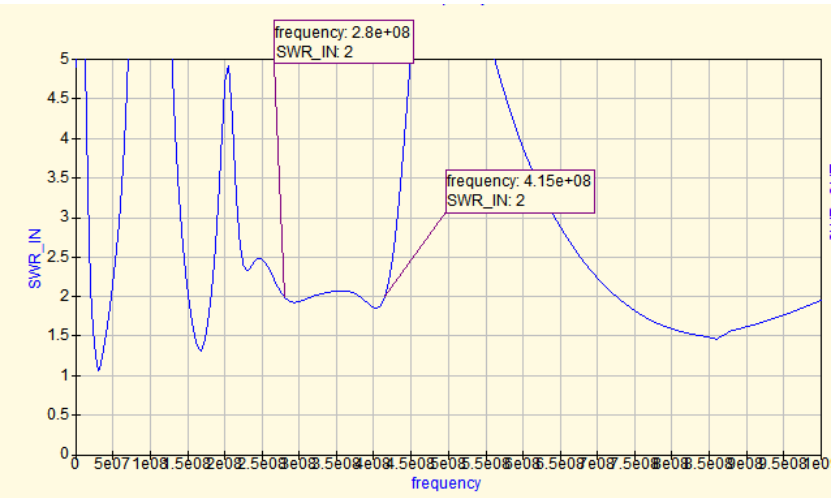


Figura 5.5: Simulación software QUCS SWR a la entrada LNA-LPF-HPF

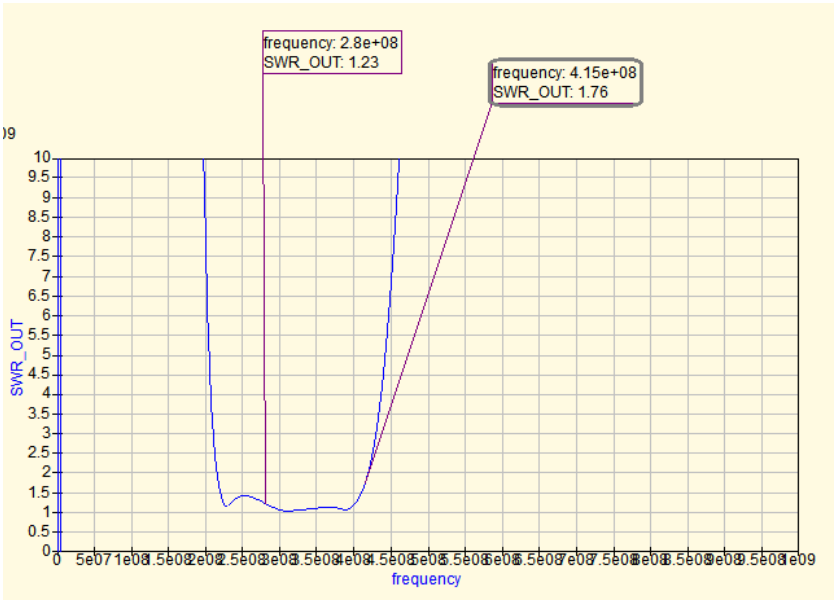


Figura 5.6: Simulación software QUCS SWR a la salida LNA-LPF-HPF

aunque carezca de la directividad deseada. Esto deja abierta una posibilidad interesante como trabajo a futuro de diseñar y construir una antena directiva para esta banda.

Para compensar la falta de ganancia en el hardware, se prevé la adición de un *splitter* conectado a la salida VGA del ordenador. La salida desconectada de este dispositivo aumentará la potencia de las emanaciones.

La imagen de la Figura 5.7 muestra la configuración de componentes obtenidos en base a lo desarrollado.

En la sección 7.1 se muestran pruebas y mediciones hechas para verificar la

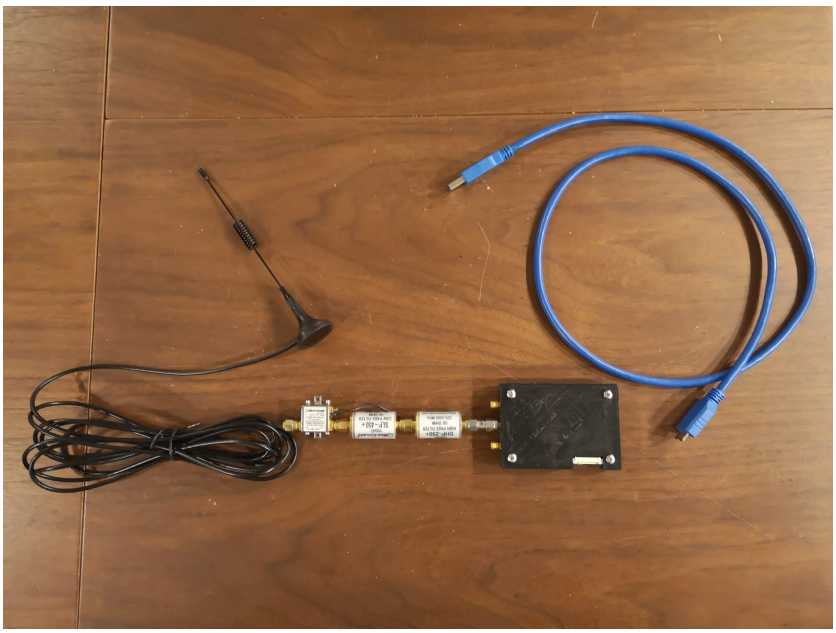


Figura 5.7: Configuración de hardware: Antena, LNA ZX60-P103LN+ (con Fuente DC 5V), HPF SHP-250+, LPF SLP-450+ y USRP B200mini

correcta funcionalidad de todos los componentes.

Capítulo 6

Implementación de Software

La implementación de mejoras y funciones nuevas al sistema preexistente define lo central del trabajo realizado en el proyecto. En capítulos anteriores se introdujeron las oportunidades de mejora identificadas en gr-tempest y se presentaron las herramientas requeridas para implementar esos cambios.

En este capítulo se presentan las funcionalidades modificadas o añadidas al sistema, brindando las justificaciones correspondientes y, si corresponde, las nociones de alto nivel necesarias para comprender sus implementaciones.

6.1. Inferencia de Resolución

Como se introdujo al analizar el funcionamiento del sistema original, los datos de resolución del sistema requeridos para procesar las muestras y visualizar la pantalla debían ser ingresados por un usuario que los conociera. De lo contrario, probar una por una las resoluciones en tiempo de ejecución hasta dar con la correcta era la forma de hallarla.

Aparece, entonces, la oportunidad de implementar una funcionalidad nueva al sistema: la posibilidad de reconocer los datos de resolución a partir de la propia señal espiada. Esto es, determinar su tamaño vertical y horizontal, sus tamaños de *blanking* y su frecuencia de refresco.

Para la estimación de dichos valores, se necesita calcular la autocorrelación de una ventana de muestras provenientes del hardware o una grabación. El tamaño de la ventana utilizada para generar la autocorrelación debe ser necesariamente mayor a un cuadro entero (1,5, por ejemplo), para asegurar que se incluya la información de la repetición entre un cuadro y el siguiente.

Para la resolución de 1080p, este es un número de muestras igual a $1,5 \times P_x \times P_y = 1{,}5 \times 2200 \times 1125 \approx 3,71$ MS. Calcular la autocorrelación de dicha ventana en el dominio del tiempo requiere tomar esta cantidad de muestras y operar con ellas con diferentes retardos, lo cual implica un nivel de cómputo difícil de mantener en el procesamiento del *scheduler* de GNU Radio.

Esto genera la necesidad de investigar otras formas de hacer este procesamiento. A partir de la literatura, se identificaron técnicas para lograr el cálculo de

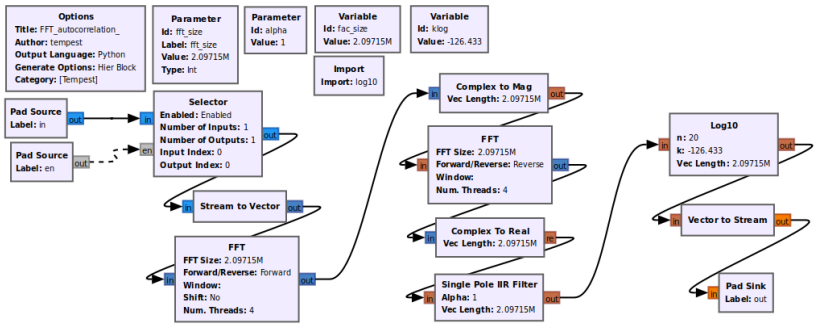


Figura 6.1: Flowgraph para el cálculo de la autocorrelacion de muestras

la autocorrelación. Se destaca el uso de la transformada rápida de Fourier o FFT (*Fast Fourier Transform*) que permite alcanzar resultados idénticos a los del cálculo de autocorrelación en el dominio del tiempo, pero manejando procesamientos de orden $O(n \times log(n))$, en lugar de uno de $O(n^2)$. Para esto se considera el enunciado del teorema de Wiener-Khintchin [29], calculando la autocorrelación a través de su densidad espectral de potencia, la cual se obtiene a partir de un estimador por método de Welch porque se estará promediando lotes o periodogramas separados a partir de las muestras de la señal [30].

Para la implementación de la autocorrelación descrita en la Sección 4.1.3, se arma un *HierBlock* (bloque conformado por otros bloques a través del GNU Radio Companion) denominado *FFT_autocorrelation*, cuyo *flowgraph* se puede ver en la Figura 6.1. En el mismo se utiliza el bloque *FFT* en cascada con *Complex to Mag* para el calculo de la densidad espectral de potencia. Luego se vuelve a usar el bloque *FFT* para antitransformar y obtener el valor de la autocorrelación. Los bloques restantes realizan operaciones para que la señal sea práctica para el procesamiento que se describirá a continuación.

Obtenida la autocorrelación de la señal recibida, requiere ser procesada para obtener los valores de interés: P_x, P_y y f_v. Para esto se implementa un bloque desarrollado mediante la herramienta *gr_modtool*.

En el bloque *infer_screen_resolution* se diseña un detector de periodicidades entre dos cuadros consecutivos. Se pone en cascada tras el *FFT_autocorrelate*, haciendo un análisis de la autocorrelación entre dos cuadros que consiste en determinar el tiempo en donde ésta se maximiza.

Para facilitar la búsqueda y mejorar los tiempos de respuesta, el sistema despliega la autocorrelación en la GUI de forma de que el usuario pueda identificar el (notorio) pico resultante del cálculo. El usuario ingresa, en tiempo de ejecución, un estimado del valor temporal donde visualiza el pico, de manera que el Algoritmo 1 acceda a este vía *callback* y realice su primera búsqueda: el máximo punto de autocorrelación en un rango reducido, centrado en el número ingresado, resultando en el tiempo inverso a la frecuencia de refresco.

Para su segunda búsqueda, se intenta encontrar el siguiente máximo relativo. La autocorrelación de este máximo será de menor magnitud que la dada entre un cuadro y el siguiente, ya que corresponde a la de una línea horizontal con la

siguiente. Esta diferencia temporal permitirá conocer la altura total de la imagen, es decir, la resolución vertical.

Algoritmo 1: INFER_RESOLUTION::GENERAL_WORK: Determina los parámetros de resolución de la pantalla espiada a partir la autocorrelación de su señal.

Input: autocorrelación, frec_muestreo
Output: null sink
inicialización;
pico_rr =
 max(autocorrelacion[margen_centrado_en_pico_visualizado]);
refresh_rate = $\frac{frec\_muestreo}{pico\_rr}$;
margen_chico = máximo_renglon_posible ;
pico_v = max(autocorrelacion[margen_chico]);
V_total = $\frac{frec\_muestreo}{pico\_v}$;
H_total = búsqueda_tabla(refresh_rate, V_total);
publicar(H_total,V_total,refresh_rate);

Los dos valores hallados son suficientes para definir totalmente la resolución, por lo que basta con utilizarlos como entrada en una tabla de búsqueda (o *look-up table*) para obtener, por establecimiento del estándar VESA, los valores correspondientes a la resolución horizontal y ambos tamaños de *blanking*.

La funcionalidad explicada, se puede notar, debe operar por separado al sistema principal. En caso de desconocer la resolución del monitor espiado, la estimación debe realizarse antes, o los procesos de corrección de muestreo y centrado no podrán operar. Por otro lado, una vez que se conoce la resolución, pierde sentido la continuidad de su cálculo durante el espionaje.

Por estos motivos, se ejecuta en un nuevo escenario (es decir, un nuevo *flowgraph* en un GRC separado) pensado para correrse con anterioridad al propio espionaje. El mismo pude verse en la Figura 6.2. Así, el usuario contará con los parámetros necesarios para sintonizarse.

Al correr ese GRC, se accede a una GUI con la gráfica de la autocorrelación y el valor *Time peak (us)* editable en tiempo de ejecución. Con la ayuda del cursor, el usuario puede reconocer un grueso estimado del tiempo en el que se maximiza la autocorrelación y digitarlo en el recuadro indicado. En la terminal del GNU Radio Companion se imprimirá la información de la resolución inferida.

Finalmente, se añade al constructor del bloque un pequeño mensaje a imprimir en la terminal, con la finalidad de instruir mínimamente al usuario en el uso del sistema.

Los resultados del escenario descrito para la operación del bloque de inferencia de resolución podrán observarse en la Sección 7.2, junto con el análisis correspondiente.

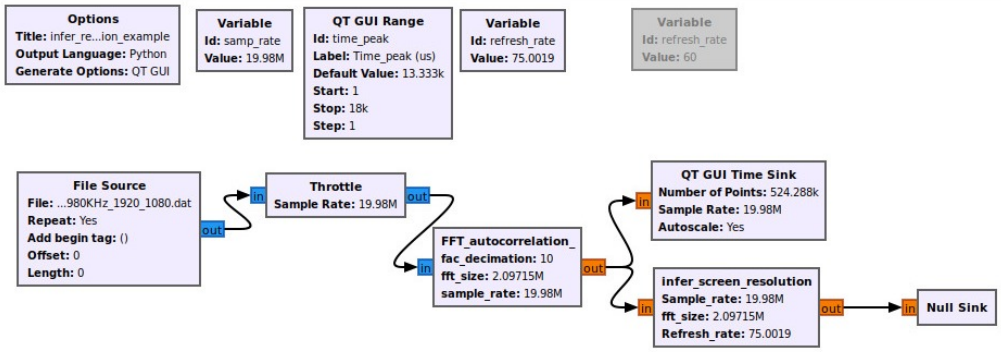


Figura 6.2: Diagrama de bloques para la inferencia de resolución

6.2. Detección de sincronización

La detección de sincronización juega un rol fundamental en el despliegue de las imágenes, tanto online como offline. Esto se debe a que, por más que el muestreo haya sido mayoritariamente corregido, errores pequeños (como pueden ser los de redondeo en la interpolación de muestras) pueden generar movimientos en la imagen que dificultan su visualización. Asimismo, la pérdida de muestras por defectos en el sistema (tanto hardware como software) también empeora la experiencia de visualización si el posicionamiento no es corregido.

El bloque existente en el sistema anterior, *Hsync*, lograba un buen resultado en la sincronización horizontal. Sin embargo, no contaba con el complemento vertical, por lo que los defectos mencionados generaban en la imagen un corrimiento vertical. Su metodología basada en el cálculo de autocorrelación entre líneas horizontales para hallar el *blanking* horizontal (ver sección 4.2.2) no tenía una traducción directa a la búsqueda del *blanking* vertical.

Por estos motivos, se opta por la implementación de un bloque completamente nuevo, *Sync Detector*, que sustituya al *Hsync* en la funcionalidad de fijar y centrar la imagen, agregando además la variante vertical. Los algoritmos de este bloque contienen una parte fuertemente basada en la implementación de Marinov, vista en la sección 4.1.2, y una parte de adaptación al funcionamiento de GNU Radio. El principal requerimiento de adaptación está dado por la limitación del *scheduler* utilizado por los bloques generales del *framework*: el límite de las muestras de entrada que se pueden retener y luego copiar a la salida es menor que un cuadro de video completo, por lo que la manipulación de los retrasos que hacen posible centrar la imagen debe realizarse por partes. Esto será visto a continuación.

6.2.1. Funcionamiento

El bloque se enfoca en el análisis vertical y horizontal de la energía de la señal, utilizado para hallar las separaciones de los *blankings* con los bordes, para luego imponer retrasos iguales a esas distancias y, a partir de ellos, copiar las muestras de entrada en la salida.

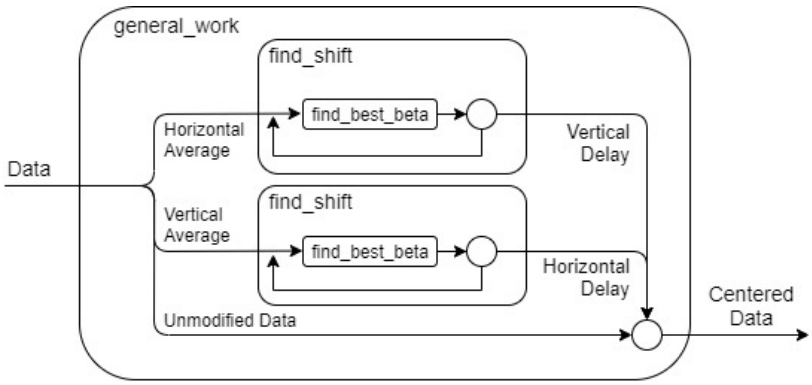


Figura 6.3: Diagrama del flujo de datos del bloque de sincronización.

Para esto se utiliza una función principal y algunas funciones auxiliares a las que se recurre para las búsquedas de *blankings*. El diagrama de la Figura 6.3 introduce una abstracción del uso de esas funciones: la función principal (o *general_work*) recibe las muestras de entrada, genera la información promediada de las filas y de las columnas de un cuadro, y corre *find_shift* para obtener la distancia al *blanking*, tanto para para la dirección vertical como la horizontal. Esta función, a su vez, hace uso de *find_best_beta* para encontrar la posición que maximiza la diferencia cuadrática de energía entre píxeles (correspondiente al *blanking*). Esta última es corrida algunas veces con variaciones en los parámetros, de manera de utilizar los tamaños que arrojen mayor diferencia.

Yendo al detalle del diseño, el Algoritmo 2 muestra la operativa de *find_best_beta* a alto nivel. Queda en evidencia, en los cálculos utilizados, el fuerte apoyo que tiene el desarrollo sobre lo visto en la sección 4.1.2.

Como se ve en dicha función, el *beta* encontrado para maximizar la diferencia cuadrática se calcula para un tamaño de blanking fijo, siendo en principio el correspondiente a la resolución del monitor puesto que es un valor tabulado. Sin embargo, debido a que las líneas verticales que determinan los límites del *blanking* horizontal concentran una energía significativa, el cálculo de diferencia de energías se vería afectado por este factor. Es por esto que la función *find_shift*, mostrada en alto nivel en el Algoritmo 3, repite el llamado a la función cambiando el tamaño a considerar de blanking que maximice esa diferencia. Esto se da considerando un tamaño mínimo de blanking, de lo contrario algunas secciones particulares de la imagen con poca energía podrían resultar en resultados muy errados por parte de la función.

Como ya se dejó ver, *find_shift* (y por ende *find_best_beta*) es independiente de la orientación de aplicación: es válida tanto para el cálculo vertical como el horizontal. La misma, luego de obtener el mejor resultado posible para *beta*, define el corrimiento a aplicar en esa dirección como la suma del *beta* (referido al comienzo del *blanking*) y medio *blanking*. De esta forma se inician los despliegues de cuadro en la mitad del *blanking*, quedando centrada la imagen y enmarcada por los cuatro espacios.

Una precaución es tomada al final del algoritmo: se usa un coeficiente para que

Algoritmo 2: SYNC_DETECTOR::FIND_BEST_BETA: para una línea entera (horizontal o vertical) y un tamaño de blanking fijo, calcula la diferencia cuadrática de las cantidades blanking y video normalizadas para cada posible ubicación del blanking y se queda con la mayor que encuentre.

Input: linea de píxeles (data), total_line_size, total_sum, blanking_size
Output: beta, beta_index
Inicialización de i, curr_sum (locales);
$screen\_size = total\_line\_size - blanking\_size$;
$curr\_sum = \sum_{i=0}^{blanking\_size} data[i]$;
$beta = \left(\frac{total\_sum - curr\_sum}{screen\_size} - \frac{curr\_sum}{blanking\_size} \right)^2$;
for *i=0, i < total_line_size, i++* **do**
  $curr\_sum = curr\_sum - data[i] + data[i + blanking\_size]$;
  $beta_i = \left(\frac{total\_sum - curr\_sum}{screen\_size} - \frac{curr\_sum}{blanking\_size} \right)^2$;
  if $beta_i > beta$ **then**
    $beta = beta_i$;
    $beta\_index = i$
  end
end

el corrimiento a aplicar en cada dirección sea considerado como suma ponderada del encontrado por esta iteración y los valores hallados anteriormente. De esta manera se atenúa el efecto generado tanto por valores atípicos o cambios bruscos como por movimientos pequeños.

La función *general_work* es la principal operadora del bloque y la que más difiere del diseño original de la funcionalidad de sincronización. Procesa la información por líneas, y lo primero que hace es aportar la información de la línea a los promedios vertical y horizontal a utilizar para los cálculos.

El resto del procesamiento consiste en la metodología alternativa implementada debido a la limitación de elementos disponibles en el *scheduler*. Para esto se cuentan las líneas procesadas y se verifican dos estados: por un lado, se cuenta hasta alcanzar un cuadro completo, ya que se contarán con los promedios completos para realizar los cálculos de *blanking*; una vez completado el cuadro, se activa un segundo estado en el que se espera la llegada del próximo blanking. Esto es, se obtuvieron las muestras de un cuadro completo con las que se hizo el cálculo, pero hasta no haber pasado el corrimiento vertical requerido por *find_shift*, no se comienza a desplegar ese cuadro pues se lo quiere centrar. Entonces, una vez conocidos los corrimientos, se espera a llegar al final del cuadro anterior (que se desplegó "tarde" por su ajuste vertical) para comenzar a desplegar el siguiente. El Algoritmo 4 evidencia este proceso, mostrando al final que también se lleva cuenta

Algoritmo 3: SYNC_DETECTOR::FIND_SHIFT: recibe la línea, suma su contenido y le pasa los parámetros a *find_best_beta*. Teniendo ubicación de la franja, actualiza la posición de despliegue considerando tanto la información nueva como la ubicación anterior.

Input: línea de píxeles (data), total_line_size, blanking_size, lowpasscoeff
Output: Corrimiento completo a realizar (shift)
Inicialización de beta, beta_index (locales)
$total\_sum = \sum_{i=0}^{total\_line\_size} data[i]$
$beta$ =find_best_beta(data, total_line_size, total_sum, blanking_size)
for *i=0,* $i < 4$, *i++* **do**
    $blanking\_size_i = blanking\_size\_attempt[i]$ `// vector predefinido`
    $beta_i$ =find_best_beta(data, total_line_size, total_sum, blanking_size$_i$) **if** $beta_i > beta$ **then**
        $beta = beta_i$
        blanking_size = blanking_size$_i$
$ubicacion = beta\_index + blanking\_size/2$
$shift = ubicacion \times lowpasscoeff + (1{,}0 - lowpasscoeff) \times shift$

de las líneas desplegadas para asegurar que los corrimientos son los apropiados.

Un punto a destacar en el ajuste vertical que se realiza es que se hace solo por retraso (y no adelanto) de líneas. De esta forma se evita la necesidad de retener la información de los píxeles hasta su despliegue, requiriendo almacenar solamente los valores promediados y los contadores de líneas.

Finalmente, es también destacable la importancia de que el bloque opere sobre cuadros con muestreo corregido. Todo el procesamiento, desde los cálculos de ubicación de *blanking* hasta el avance de punteros de entrada y salida para imponer los corrimientos, se apoya en el uso de las líneas con el tamaño fijo dado por la resolución de la pantalla espiada y la frecuencia de muestreo. Si este no fuera el caso y las líneas sufrieran variaciones no compensadas, el desarrollo con la explicada metodología habría escalado significativamente en su complejidad. Por este motivo, se ubica el bloque a la misma altura del flujo que se encontraba el *Hsync*: luego de la corrección de muestreo y previo a los procesos de normalización y despliegue, como se ve en la Figura 6.4. En la sección 7.3 se muestran los resultados de aplicar el bloque, junto con el análisis y las correcciones pertinentes.

6.3. Corrección de Muestreo

Al establecer los objetivos en el comienzo del proyecto, la corrección del muestreo era una característica dada del sistema, a ser utilizada en conjunto con las implementaciones nuevas y las modificadas. Fue en el trayecto, sin embargo, que

Algoritmo 4: SYNC_DETECTOR::GENERAL_WORK: hace el pasaje de los datos tales como se reciben de GNU a datos utilizables por las funciones para encontrar el blanking. Luego corre dichas funciones y devuelve en la salida la entrada con el corrimiento del offset encontrado.

```
Input: stream de entrada
Output: stream de salida
Inicialización de variables locales
for Cada línea recibida do
    Armo un arreglo con los módulos de los complejos
    Agrego sus datos a los promedios horizontal y vertical del cuadro
    Contador de líneas++
    if Contador de líneas == altura de un cuadro then
        Corro find_shift horizontal y vertical
        Contador de líneas = 0
        Reseteo promedios
        wait_for_blanking = 1
    if wait_for_blanking == 1 then
        Contador de espera++
        if Contador de espera == blanking vertical then
            Consumo la diferencia de blankings entre cuadros
            Contador de espera = 0
            wait_for_blanking = 0
            if frame_output == 0 then
                frame_output = 1
        wait_for_blanking = 0
    if frame_output == 1 then
        Copio una línea horizontal a la salida movida según blanking
         horizontal
        Contador de output++
        if contador de output == altura de un cuadro then
            frame_output = 0
            contador de output = 0
    Consumo un renglón
```

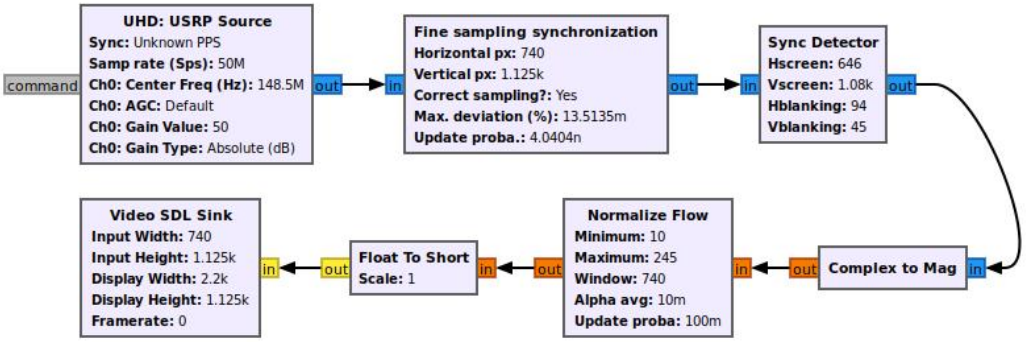


Figura 6.4: Diagrama del sistema original, implementando el bloque Sync Detector en lugar de Hsync.

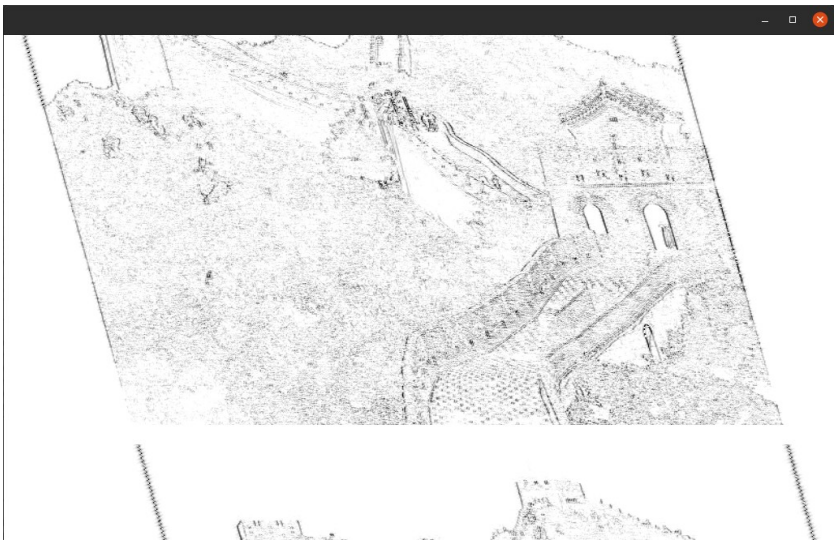


Figura 6.5: Salida regular del sistema simulado usando el bloque Fine Sampling Synchronization.

se descubrió una falla en el funcionamiento del bloque *Fine Sampling Synchronization* para el escenario simulado: el cálculo del índice de interpolación a utilizar se ejecuta por única vez al inicio de la ejecución del *flowgraph*.

El problema es claro en el escenario simulado (que es utilizado para pruebas y se explica en detalle en la Sección 7.3) porque ningún parámetro de tamaño es modificado en tiempo de ejecución dado que los valores se conocen con certeza. Entonces, si el cálculo no es acertado en el primer intento, el muestreo nunca logra sincronizarse correctamente y la imagen no queda orientada correctamente y quieta. La ejecución debe ser detenida y vuelta a iniciar.

La Figura 6.5 muestra el caso de uso que se busca explicar: la mayoría de las ejecuciones de la simulación no calculan correctamente la primera iteración del índice de interpolación, resultando en una imagen no sincronizada que no logra corregirse. Este es el punto central a corregir en el funcionamiento del bloque.

En el dimensionamiento del problema, se tuvo la intención inicial de mantener la lógica del bloque buscando la forma de corregir el problema a nivel del código. Dado que los resultados de los cálculos eran correctos, el punto de falla parecía estar en la ejecución de los mismos. A partir de un valor ingresado por parámetros, la función principal repite los cálculos aleatoriamente para ajustar la corrección. Al analizar esta lógica, no se encontraron errores en la repetición del cálculo. De hecho, en los escenarios de aplicación real, los tiempos de repetición pueden extenderse pero se alcanza eventualmente. No pudiendo corregir este punto desde el propio bloque, debió buscarse una alternativa por fuera del mismo.

En la Sección 6.1 se presentó la implementación del bloque para la inferencia de resolución a partir de las características de la señal. En particular, un análisis de la autocorrelación es realizado, revelando los picos que dan información de la diferencia temporal entre cuadros y entre líneas.

Esos mismos picos, se identificó, pueden utilizarse en el cálculo del índice de interpolación al compararse con la información esperada de la señal (referida a su resolución que es conocida). Es por eso que se decidió implementar una solución, de características parecidas al bloque de inferencia de resolución, para la búsqueda del índice a utilizar en la corrección de muestreo.

En primer lugar, se establece el uso del mismo bloque de cálculo de autocorrelación presentado en la Figura 6.1. Sobre su salida se diseña el algoritmo a utilizar.

Para el desarrollo del nuevo bloque, *FFT_peaks*, se intenta realizar una búsqueda en el arreglo de entradas por dos máximos relativos consecutivos, de forma análoga a como se operó para la inferencia de resolución. Con los índices de los valores máximos del arreglo de entrada, se puede obtener la distancia en cantidad de muestras que hay entre ellos. El valor de dicha distancia se utiliza con un *Moving Average* N para ser almacenado en forma acumulada en la variable *d_accumulator* de la siguiente forma:

$$d\_accumulator = \frac{peak\_index - peak\_index2}{N}$$

Luego de N ejecuciones del método *work*, la distancia acumulada en *d_accumulator* será un estimador del promedio de la cantidad de muestras que están siendo detectadas entre un cuadro y su consecutivo, es decir, la media muestral de la cantidad de muestras que entran en un cuadro a la frecuencia de muestreo actual.

Para esto, se procesa mediante la fórmula de estimación del error de tiempo de muestreo utilizada por gr-tempest original en *fine_sampling_synchronization*:

$$ratio = \frac{samples\_en\_un\_frame\_idealmente}{d\_accumulator} \tag{6.1}$$

Se hace uso de un *set_history(d_fft_size)* para paralelizar los datos de un cuadro entero y la función *volk_32f_index_max_32u*, de la librería VOLK [31], retorna el valor del índice del vector de entrada en el cual se halla un máximo.

A este dato se le suma el valor 1.0 para darle el formato correcto, siendo una corrección relativa a la frecuencia de muestreo que se está utilizando en el

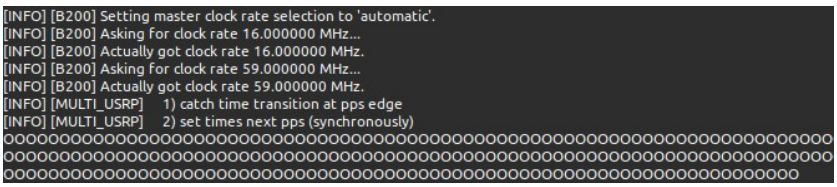


Figura 6.6: Salida en terminal del driver de SDR notificando de las muestras perdidas.

momento. Se lo publica por PMT al puerto *ratio*, que debe ser conectado al bloque interpolador para su uso.

La operativa del bloque consiste en un proceso que debe ejecutarse en tanto no se conozca el índice de interpolación, para calcularlo y realizar la corrección correspondiente. Un aspecto a notar de este proceso es que, una vez hallado, ya no es necesaria su ejecución. Entonces, la posibilidad de desactivarlo una vez que cumple su propósito cobra sentido.

Los propios mensajes PMT, en su carácter de transmisibles en tiempo de ejecución, son un medio apropiado para la habilitación del bloque. Esto se hace al recibir un comando en el puerto *en* cambiando una variable booleana. Su valor cambia de modo *toggle*: si está activa pasa a estar inactiva y viceversa.

El problema estará entonces en la forma de generar esos mensajes. Se consideró la posibilidad de ejecutar este proceso de manera automática. Para esto, se debería obtener un cálculo del índice de interpolación en una instancia posterior a la propia interpolación, de manera de conocer instantáneamente cuando ese índice se hace prácticamente 1. La limitante para este caso reside en el procesamiento: volver a calcular el índice por el método implementado implica volver a tomar las muestras para la autocorrelación y ejecutar el cálculo en simultáneo al realizado para interpolar. El consumo de los procesos conjuntos se hace muy difícil de sostener para el equipo utilizado en el espionaje.

La alternativa seleccionada para implementar la deshabilitación del cálculo de autocorrelación es la generación del mensaje booleano PMT mediante un botón en la GUI, obtenido de la librería GUI Extra [32]. De esa forma, la recepción en el puerto *en* conmuta la variable de habilitación, permitiéndole detener la operativa si se alcanzó un índice que se determine apropiado (el resultado estará siendo desplegado en pantalla por el sistema) o retomarla en caso que sea necesario.

La Sección 7.4 describirá los resultados de esta implementación, dando lugar al análisis correspondiente.

6.4. Aumento de la Frecuencia de Muestreo

Una limitante fundamental es la capacidad de procesamiento del CPU utilizado para correr la aplicación. Por una parte, se quiere recibir una señal muestreada a una alta frecuencia de muestreo para reconstruir lo mejor posible la imagen. Por otra parte, no cualquier procesador es capaz de recibir y tratar señales a tasas de muestreo tan altas. Esto último genera un efecto poco deseado en la operativa: la pérdida de muestras, como la mostrada para GNU Radio en la Figura 6.6.

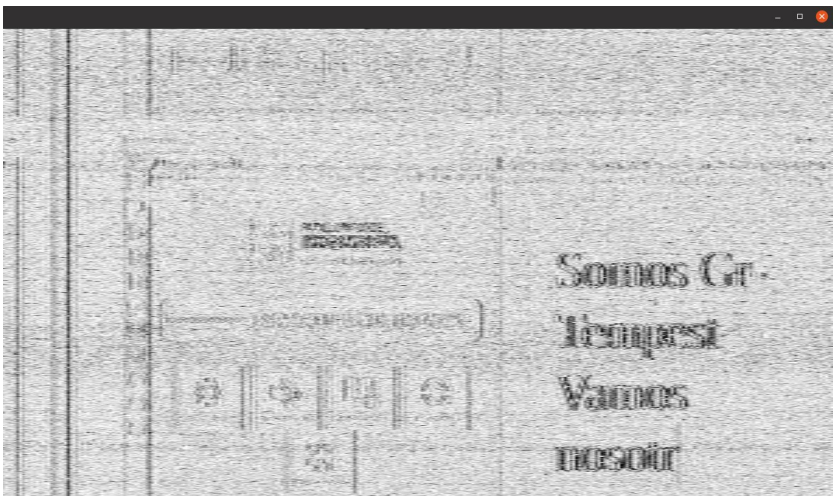


Figura 6.7: Ejemplo de señal submuestrada cuya imagen es difícil de reconstruir.

Dado que la señal de video es continua, perder muestras implica cortar un cuadro y perder la sincronización. En la práctica, esto se traduce en cortes o *glitches* en la imagen. El algoritmo de detección de sincronismo está hecho para tolerar y corregir ese efecto, en tanto ocurra con poca frecuencia. Mayor frecuencia de los *glitches* dificultará la visualización de la imagen, luego la capacidad de algoritmo de sincronizarse efectivamente y, finalmente, la capacidad del sistema de reconstruir la imagen.

Es por estas razones que se hace primordial algún tipo de optimización en el procesamiento, que permita al sistema trabajar con cierta holgura. Esto puede tomar muchas formas. La primera, y más básica, es simplemente reducir la tasa de muestreo, disminuyendo de forma significativa la tasa de datos a procesar por el sistema. Con un SDR como el de la Figura 6.6 que notifica la existencia de muestras perdidas, se podrá notar inmediatamente la reducción de la mismas. Sin embargo, y como se mencionó, perder frecuencia de muestreo tiene un efecto colateral de gran importancia: submuestrear la señal implicará pérdida de información y una menor capacidad del sistema para reconstruir la imagen original. Es decir, bajará significativamente la calidad de la imagen (ver Figura 6.7).

Por lo tanto, se quiere implementar alternativas que le quite peso a los filtros digitales de alto procesamiento del CPU, sin comprometer la integridad de la imagen recibida. Un pequeño experimento, como la prueba de consumo que se verá en la Sección 7.3, permite visualizar algo que se hizo claro a lo largo de las distintas ejecuciones de gr-tempest: la mayor parte del consumo de CPU ocurre en el proceso de interpolación para la corrección del muestreo.

En este sentido, se proponen dos formas de enfrentar el problema. Una primera aproximación puede utilizar el índice de interpolación calculado por el bloque *FFT_peaks* y transmitirlo como un mensaje al bloque driver del hardware *USRP Source* para que el ajuste de muestreo se haga en el dispositivo y no sea necesaria la interpolación por parte del software. La limitante de este diseño está dada por la pérdida de generalidad en caso de que otros tipos de hardware no soporten los

mismos mensajes y no ajusten correctamente el muestreo.

La alternativa consistiría en que, previo al procesamiento de interpolado del *Fine Sampling Synchronization*, se descarten muestras correspondientes a cuadros enteros de la imagen de video. Esto haría que muchas menos muestras deban ser procesadas, bajando el peso sobre el CPU. La consecuencia de un menor *frame rate* resulta menor en la aplicación de espionaje de monitores. El principal problema de este desarrollo está dado por el hecho de no conocer con certeza la cantidad de muestras correspondientes a un cuadro entero antes de la corrección de muestreo. En función de eso se presenta lo trabajado en esta sección.

La implementación final buscará proveer ambas aproximaciones, de manera que el usuario seleccione el método según considere pertinente en el contexto de su caso de uso.

6.4.1. Realimentación SDR

Se ha hecho fuerte incapié a lo largo de este trabajo en las ventajas del uso de SDR en aplicaciones como la presente, por su capacidad de modificar parámetros desde software. Para este punto en particular, se propone que la modificación se haga en tiempo de ejecución, permitiendo que la corrección de muestreo se haga en hardware, en la medida que el software va calculando el error a corregir.

Los mensajes PMT son los habilitadores en el software para realizar este cambio: el bloque *USRP Source* cuenta con una entrada PMT a ser utilizada para la transmisión de distintos comandos en tiempo de ejecución. Entre ellos se encuentran la modificación de la ganancia, de canal, de frecuencia central y del ancho de banda [33].

Para la modificación de la frecuencia de muestreo, debe emplearse la nomenclatura específica detallada en la documentación del bloque. En este caso, se transmite como sigue:

```
message_port_pub(
    pmt::mp("rate"),
    pmt::cons(pmt::mp("rate"), pmt::from_double((double)new_freq))
);
```

en donde *message_port_pub* corresponde a la función utilizada para publicar un mensaje PMT en un puerto, *rate* hace referencia al puerto puntual por el que se querrá publicar, y la variable local *new_freq* es donde se tiene almacenado el valor de la frecuencia de muestreo corregida.

De esta manera, se compone un mensaje a transmitir al driver del SDR como en la Figura 6.8. El bloque recibe el mensaje por la entrada específica y continúa operando con los cambios ejecutados.

El SDR y, por tanto, las muestras que ingresan al sistema, pasan a trabajar a una frecuencia de muestreo diferente a la que venía utilizándose anteriormente. Esto implica que las actualizaciones sucesivas de cálculos del índice de interpolación realizados por el bloque *FFT_peaks* deberán tener en cuenta estas modificaciones de frecuencia para continuar ajustando correctamente el muestreo.

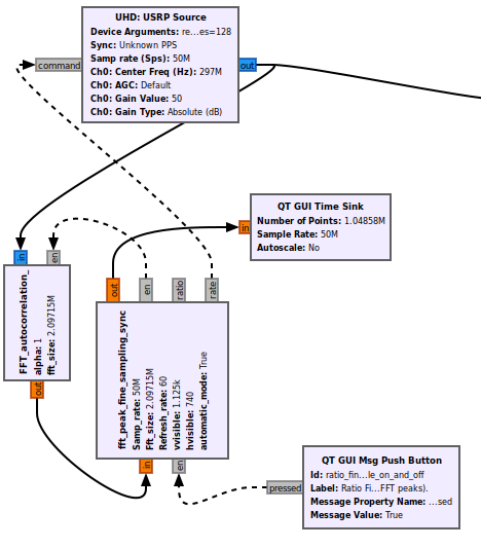


Figura 6.8: Diagrama del conexionado para la implementación del sistema con realimentación SDR.

El detalle de las actualizaciones, se recuerda, es consecuencia de que la conversión a tiempo de la señal obtenida depende directamente de la cadencia con la que ingresan sus muestras: si se cambia la frecuencia de muestreo de la señal pero no se ajusta el parámetro internamente en el bloque, los cálculos estarán convirtiendo la separación entre las muestras a un tiempo incorrecto, resultando en una corrección errónea.

Con esta consideración implementada, se cuenta ahora con un sistema de realimentación que halla la corrección a aplicar a la frecuencia de muestreo, modifica ese parámetro en el hardware, y aplica el resultado en los propios cálculos para seguir iterando hasta converger en una señal con muestreo corregido que no requiere interpolación por software.

6.4.2. Descarte de Cuadros Completos

Descartar cuadros completos se plantea como la alternativa por software para la reducción de procesamiento. Parte de una premisa ya mencionada: el grueso del procesamiento requerido de la CPU está dado por la interpolación de muestras para corregir el muestreo. Así se justifica el añadir tareas al sistema para alivianar su procesamiento.

Se plantea, entonces, un módulo que opere antes que el bloque encargado de la interpolación en el orden del flujo de datos. Por tanto, estaría recibiendo en su entrada la totalidad de las muestras que genera el driver de hardware, y poniendo a su salida solo una porción de esas muestras para disminuir la tasa de datos recibida por el interpolador.

Con esta idea clara, la problemática del desarrollo de este bloque reside en seleccionar correctamente qué muestras copiar a la salida del bloque y qué muestras

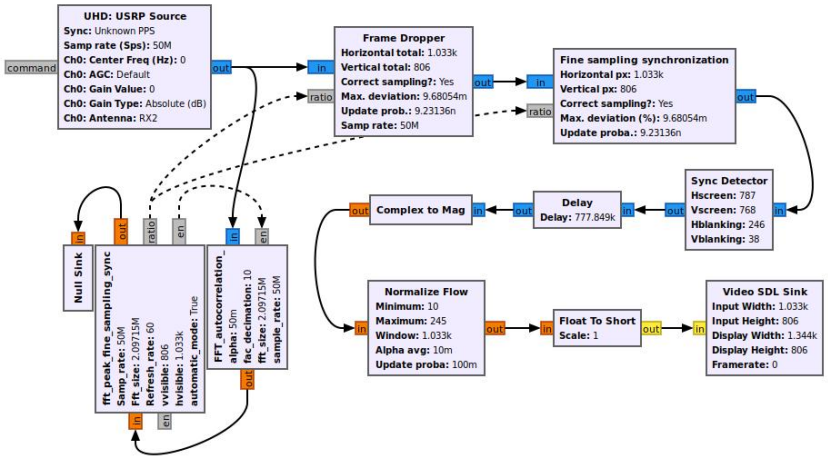


Figura 6.9: Diagrama del sistema, implementando Frame Dropper en las muestras recibidas del SDR.

descartar. Este trabajo, como se introdujo al comienzo de la sección, conlleva una dificultad importante puesto que el muestreo aún no fue corregido y el tamaño de las líneas no es fijo. Estas líneas serán las requeridas de forma de que, luego de interpolar, resulte en el tamaño de la línea deseado. Ese es el criterio primario para contar las líneas que constituirán los cuadros.

Para esto, el algoritmo debe conocer las muestras que luego va a requerir el bloque interpolador para generar las líneas. Ese bloque opera como se explicó en la Sección 6.3: dado que ahora se calcula el índice de interpolación dentro del bloque *FFT_peaks* y se pasa por mensaje PMT, el *Fine Sampling Synchronization* simplemente corre su función de interpolación usando el índice recibido y entrega la señal con el muestreo corregido (ver Figura 6.9). Esa función genera las muestras interpoladas y devuelve el número de muestras que fueron requeridas a la entrada para generar la salida.

La solución implementada, en línea con esta operativa, también recibe el índice de interpolación, y corre una función idéntica a la del interpolador pero que omite el paso de la propia interpolación. Es decir, calcula las muestras requeridas para interpolar tal como lo hará el siguiente bloque, pero no interpola. Se queda con el número de muestras requeridas para interpolar. En base a este número es que se resuelve el primer problema: determinar la cantidad de muestras que luego se corresponderán con cuadros enteros, para saber cómo limitar las iteraciones de muestras entrantes.

El Algoritmo 5 muestra la función principal del bloque *Frame Dropper*. Allí se ve cómo el bloque tiene una lógica más bien simple: luego de inicializar las variables, se inicia el conteo de las muestras del *scheduler* relativo a un contador que se mantiene a través de las distintas iteraciones del *general_work* (porque sabemos que la cantidad de muestras disponibles por vez son menores a las de un cuadro completo, aún sin interpolar). Hasta que el contador alcanza un cuadro entero, se copian las muestras de la entrada a la salida. Luego se cuentan los cuadros que se quieren descartar y, durante esas iteraciones del *for*, se consumen

las entradas sin avanzar el puntero de salidas.

Algoritmo 5: FRAME_DROP::GENERAL_WORK: Función principal del bloque, encargada de calcular y transmitir el índice de interpolación así como llevar la cuenta de las muestras y descartar cantidades proporcionales al tamaño de un cuadro completo.

```
Input: stream de entrada, cuadros a descartar, tamaño de cuadro
Output: stream de salida
Inicialización de variables locales
N = cuadros a descartar
frame_size =muestras requeridas para interpolar
for i=0; i < noutput_items; i++ do
    Contador_de_muestras++
    if Contador_de_muestras< frame_size then
        Out[i] = In[i]
        Puntero_salida++
    if Contador_de_muestras= N × frame_size then
        Contador_de_muestras = 0
        Puntero_entrada = i
    if Contador_de_muestras mod frame_size = 0 then
        frame_size =muestras requeridas para interpolar
if Puntero_entrada == 0 then
    Puntero_entrada += noutput_items
Actualizo remanente de interpolación
```

Un punto a notar para comprender la lógica del algoritmo es la premisa de que siempre se descartará una cantidad de cuadros mayor o igual a los que se dejan pasar. Así se explica que el primer cuadro es siempre uno y los siguientes son un número natural. Esto se desprende de una idea ya introducida: la imagen de video de un monitor tiene muy poca variación en sus usos normales y una alta frecuencia de refresco para la mayoría de esos usos. Esto implica que una fracción de los cuadros mostrados sigue siendo valiosa a los efectos del espionaje. Con números de cuadros descartados variando entre 1 y 9 por cada cuadro mostrado, se tienen frecuencias de refresco de 30 a 6 cuadros por segundo, que se consideran apropiados.

La otra consideración a tener para la implementación del algoritmo es la frecuencia con la que se calcula la cantidad de muestras requeridas para interpolar. Se notó en la práctica que volver a calcularlo en la mitad de un cuadro resulta en el corrimiento de una parte de la imagen de salida. Por este motivo se opta por solo actualizar la cantidad de muestras requeridas luego de completar cada cuadro.

El remanente utilizado para interpolar, sin embargo, se vuelve a calcular en el interpolador en cada iteración de su *general_work*. Por este motivo, el *Frame*

Dropper hace lo mismo para mantenerlo sincronizado, pero solo actualizando los tamaños al final del cuadro para no modificarlo a mitad de camino.

6.5. Ecualización

Las imágenes obtenidas del procesamiento explicado hasta ahora, como se podrá ver en las primeras secciones de análisis de resultado, dan una buena percepción del contenido de la imagen del monitor espiado, pudiendo reconocer contornos e identificar con cierta claridad de qué se trata en cada caso. Esto es un resultado que se mantiene desde lo logrado por la versión original de gr-tempest.

No obstante, en la interpretación un poco más fina de la información contenida en la imagen, el despliegue mostrado hasta ahora se encuentra limitado: la mayoría de la información que se puede obtener de un monitor de computadora personal consiste en texto y los contornos mostrados en la imagen muchas veces no permiten al usuario del sistema espía reconocer los caracteres y efectivamente recibir su contenido.

En esta sección se busca explicar el desarrollo realizado para la implementación de filtros sobre la señal, con el fin de tener una mejor visualización de la misma. La implementación se divide en dos partes: se comienza con el método de ecualización lineal y luego una corrección del corrimiento en frecuencia.

6.5.1. Ecualización Lineal

Como ya se adelanto en la Sección 3.3, la señal recibida no contiene información de continua y es atenuada a frecuencias bajas. En la presente sección, se explora la implementación de una técnica de ecualización lineal que busca compensar este efecto.

Recordando la forma del espectro (ecuación (3.7) y Figura 3.3), si bien no se podrán recuperar los valores ya anulados, se busca diseñar un filtro que compense la atenuación a frecuencias bajas.

En la Figura 6.10 se introduce el filtro propuesto para la ecualización lineal. En ella se puede ver en verde la representación en el dominio de frecuencia del filtro junto con $G(f)$ en rojo. $H_{eq}(f)$ es igual al inverso de $G(f)$ en el espectro (amplificando las componentes de baja frecuencia para compensar el efecto del seno cardinal), con la excepción de un pequeño entorno del cero para el cual se utiliza una interpolación entre los valores de los elementos del borde.

En la visualización de la pantalla espiada, se espera que la imagen ecualizada tenga atenuado el efecto de "detección de bordes". Al implementar el filtro $H_{eq}(f)$ se amplifican los valores de baja frecuencia, pudiéndose ver una imagen con mayor cantidad de píxeles consecutivos horizontales similares.

En el ambiente de GNU Radio, la ecualización lineal se hace a través del bloque *Interpolating FIR Filter*. Para la generación de los *taps* a utilizar por ese bloque, se necesita la información del armónico al cual se esta sintonizando, la tasa de muestreo y la tasa de píxeles (inverso del tiempo de píxel T_p). Como la última depende directamente de las características del monitor espiado, luego de inferir

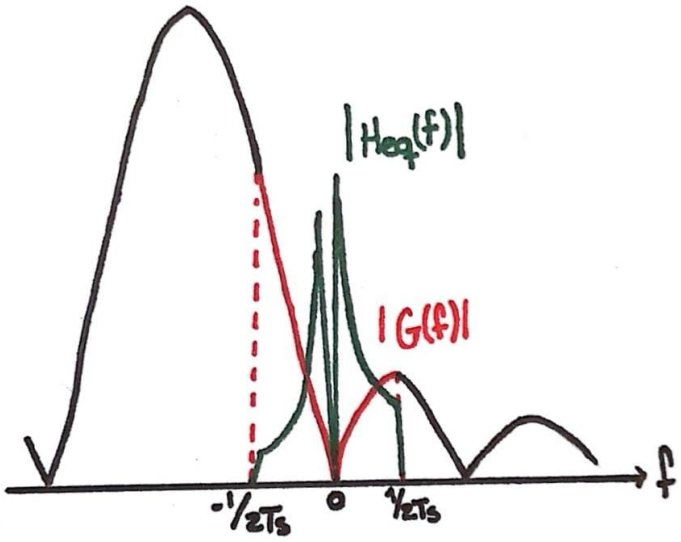


Figura 6.10: Bosquejo del espectro del filtro ecualización junto con el espectro de $G(f)$.

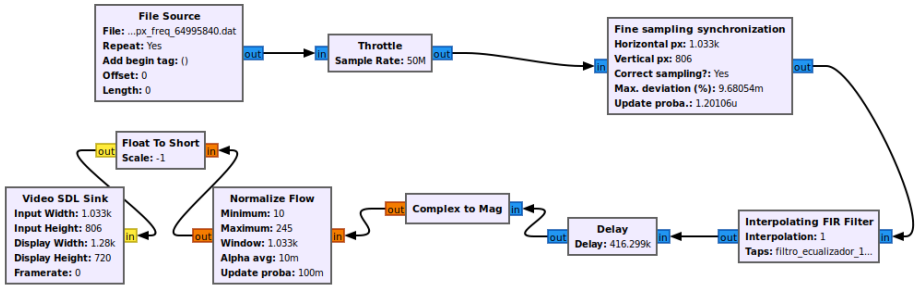


Figura 6.11: Flowgraph de prueba de filtro de ecualización lineal.

la resolución se está en condiciones para generar el filtro de ecualización conforme a lo explicado con la figura 6.10 (mediante un *script* de Python ahora incluido en gr-tempest).

En la Figura 6.11 se muestra el diagrama de flujo con el ecualizador lineal. Obsérvese que, previo a la visualización, se realiza un cambio de números complejos a la magnitud de los mismos. Al realizar esta operación, la información de signo se pierde y así también los errores de frecuencia, razón por la cual se implementaba en el sistema original. En la visualización de la imagen, esto implica cambios de blanco a negro sin escala de grises. Al realizar corrección del corrimiento en frecuencia, como se detallará en la proxima sección, esto se modificará.

6.5.2. Corrimiento en Frecuencia

A continuación se describe la implementación para la corrección del corrimiento de frecuencia causado por la sintonización imperfecta descrita en la Sección 3.3.

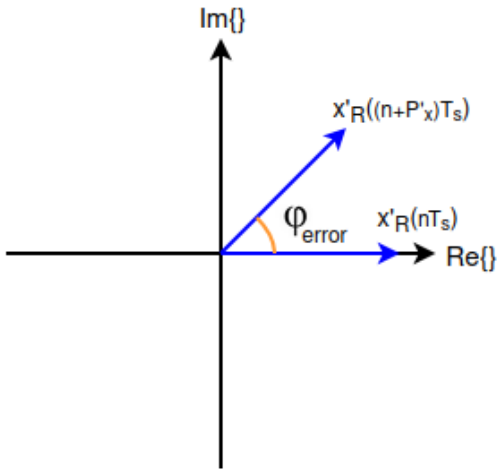


Figura 6.12: Diagrama vectorial de dos muestras de píxeles adyacentes verticalmente.

Recordando la expresión (3.9), se tiene que la señal recibida gira en el plano de los complejos con velocidad $2\pi f_\Delta$. Como ya se explicó, la señal recibida no contiene componente de continua, por lo que $x_R(nT_s)$ puede tomar valores negativos. Nótese que, si se toma el módulo de la señal, se tiene $|x'_R(nT_s)| = |x_R(nTs)|$ pero se pierde la información del signo. Por otro lado, si se toma la parte real, se obtiene una fracción del valor de $x_R(nT_s)$. La corrección del corrimiento de frecuencia consiste en detectar el corrimiento que causa el giro en el plano de los complejos para a través de la multiplicación por $e^{-j2\pi f_\Delta nT_s}$, corregirlo y así obtener una señal corregida $x_{Req}(nT_S)$ igual a la recibida en condiciones ideales: $x_{Req}(nT_s) = x_R(nT_s)$.

En la Figura 6.12 se presenta el diagrama vectorial de dos muestras consecutivas verticalmente. Dada la forma secuencial horizontal de la señal VGA, estas son dos muestras separadas por P'_x, donde P'_x es el correspondiente en muestras de una línea horizontal de píxeles, es decir:

$$P'_x = P_x \frac{f_s}{P_x P_y f_v}$$

De una línea a otra no se esperan grandes cambios en la imagen, por lo que se puede suponer que el desfasaje de las muestras ϕ_{error} es mayormente debido al corrimiento de frecuencia.

Una forma para estimar ϕ_{error} es tomando el argumento de la multiplicación conjugada de las muestras separadas P'_x:

$$x_{aux}((n+P'_x)T_s) = x'_R(nT_s)\overline{(x'_R((n+P'_x))T_s)}.$$

Recordando la expresión (3.9) para $x'_R(nT_s)$:

$$x_{aux}((n+P'_x)T_s) = e^{j2\pi f_\Delta nT_s} x_R(nTs) e^{-j2\pi f_\Delta (n+P'_x)T_s} x_R(n+P'_x).$$

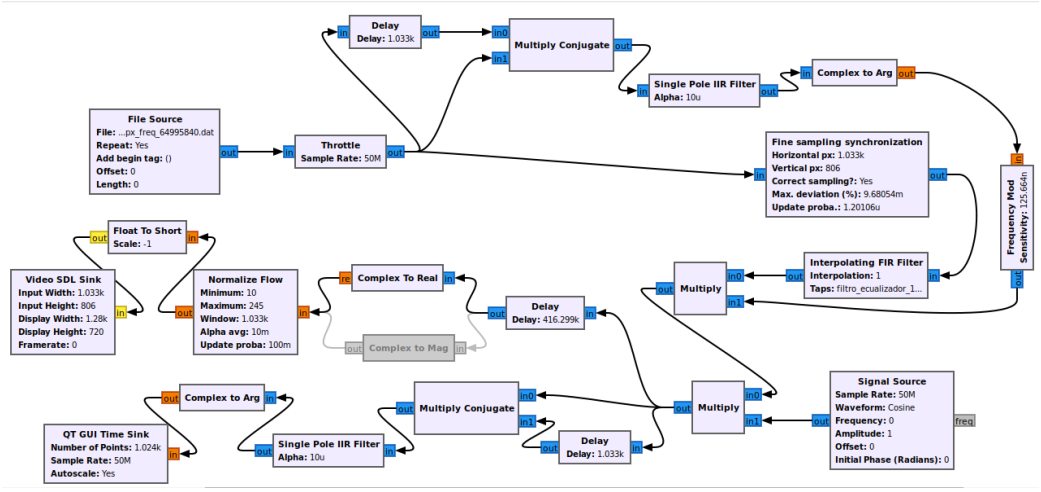


Figura 6.13: Flowgraph con sistema de ecualización.

Suponiendo que de una línea a otra no hay grandes cambios, $x_R(nTs) \approx x_R(n + P'_x)T_s$:

$$x_{aux}((n + P'_x)T_s) = e^{-j2\pi f_\Delta P'_x T_s}|x_R(nTs)|^2$$

Al tomar el argumento se tiene:

$$\phi_{error} = -2\pi f_\Delta P'_x T_s \tag{6.2}$$

Para la implementación en el ambiente de GNU Radio se utiliza el *flowgraph* mostrado en la Figura 6.13. Véase que el bloque *Complex to Mag* es sustituido por *Complex to Real*.

El cálculo de ϕ_{error} se realiza con los bloques *Delay*, *Multiply Conjugate*, *Single Pole IIR Filter* y *Complex to Arg*. De esta manera, se promedian los resultados de multiplicar una línea por el conjugado de la anterior para obtener la diferencia de fase.

La corrección del desfasaje se realiza en dos etapas, una primera automática y una segunda manual. En la primera, el bloque *Frecuency Mod* realiza la operación del coseno complejo con argumento ϕ_{error}. Al multiplicarlo con la señal de entrada, se modula la señal con el argumento detectado.

En la segunda etapa se calcula el ϕ_{error} remanente y la corrección se realiza visualizando en la gráfica el valor del argumento aún sin corregir para ingresar su valor en una *QT GUI Variable*. En este se realiza el despeje de f_Δ en (6.2) y via *callback* el coseno complejo del bloque *Signal Source* vuelve a modular la señal. La detección implementada calcula un error aproximado, para luego manualmente buscar la frecuencia que otorga una mejor visualización. En la Figura 6.14 se muestra el *GUI* donde en la gráfica se ve el valor de ϕ_{error} aún sin corregir y en el *QT GUI Variable* llamado "graphic value" el valor corregido hasta el momento.

Es importante señalar que el sistema puede detectar un desfasaje de hasta 2π. Si es mayor y no se corrige, se visualiza como cambios de blanco a negro en líneas verticales. Este fenómeno se puede ver en las Figuras 6.15, 6.16 y 6.17, donde

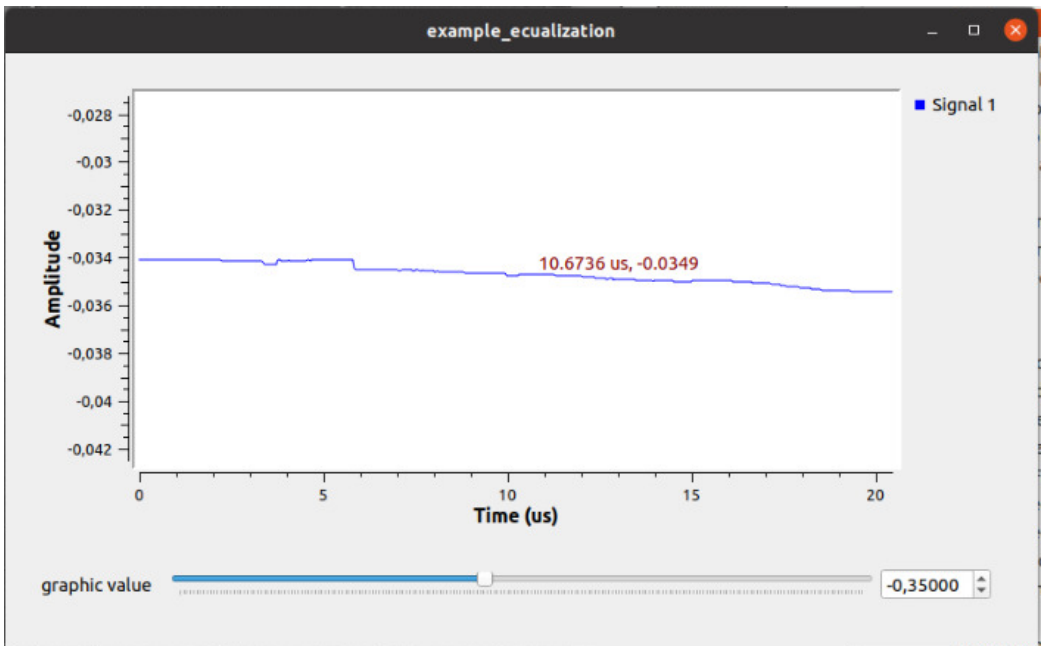


Figura 6.14: GUI para la implementación de corrección de f_Δ.

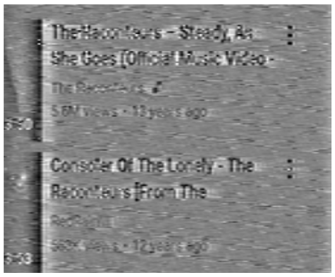


Figura 6.15: Imagen espiada con un error en frecuencia central $f_\Delta \approx 0.06$ kHz.

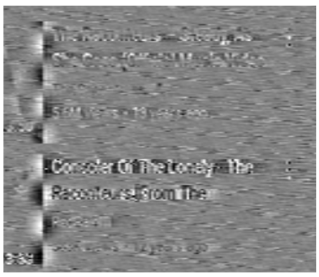


Figura 6.16: Imagen espiada con un error en frecuencia central $f_\Delta \approx 1.02$ kHz.

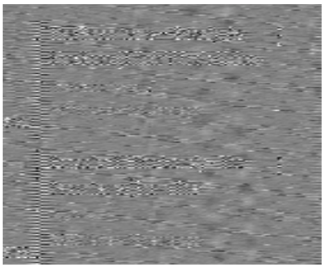


Figura 6.17: Imagen espiada con un error en frecuencia central $f_\Delta \approx 18.93$ kHz.

se hace una comparativa de tres versiones del mismo texto espiado con distintos valores de corrimiento de frecuencia f_Δ.

La Sección 7.6 brindará varios y variados resultados de las aproximaciones presentadas para mejorar la visualización de la imagen.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 7

Medidas de Desempeño y Análisis de Resultados

Las implementaciones vistas en los capítulos anteriores cumplen su objetivo de mejorar o extender la funcionalidad del sistema original en tanto verifiquen los criterios de éxito elegidos para su resultado. En este capítulo se establecen esos criterios para cada implementación y se presentan las medidas tomadas en función de ellos, para luego analizar los resultados en cada caso.

7.1. Pruebas de Hardware

Las primeras medidas tomadas fueron sobre el sistema de Hardware presentado en el Capítulo 5. Esta cronología se debió a que el RF front-end fue implementado al comienzo del proyecto, con lo que se consideró apropiado que las las pruebas correspondientes fueran realizadas antes de establecer el sistema como base para el aplicativo de gr-tempest.

Las pruebas del hardware se dividen en dos partes: las realizadas al conjunto Amplificador-Pasabajos-Pasaaltos para analizar su transferencia, y las pruebas del sistema completo realizadas utilizando GNU Radio. En ambos casos se vincula el resultado con los requisitos definidos al desarrollar la implementación del hardware, realizando el análisis correspondiente.

7.1.1. Medidas VNA

En primer lugar, se midieron con el ROHDE & SCHWARZ ZVB 8 - VNA las transferencias correspondientes a las simuladas para compararlas. Como se verá a continuación, los resultados fueron muy satisfactorios y se logró verificar la correcta funcionalidad de todos los componentes del RF front-end.

Como se ve en las Figuras 7.1 y 7.2, se midieron los parámetros S21 (dB Mag), S12 (dB Mag), S11 (SWRin) y S22 (SWRout) correspondientes a la transferencia y la SWR a la salida y entrada del circuito.

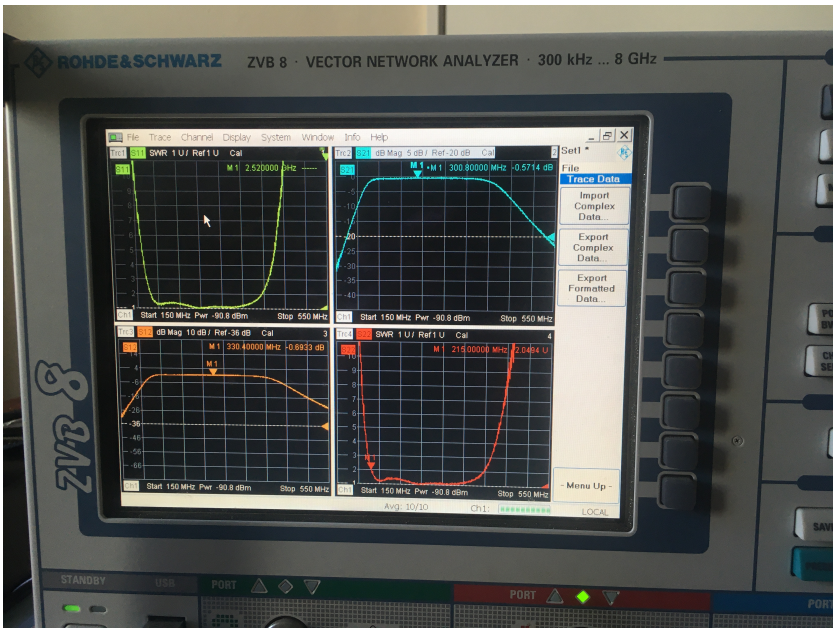


Figura 7.1: Salida del ROHDE & SCHWARZ ZVB 8 - VNA para el circuito probado.

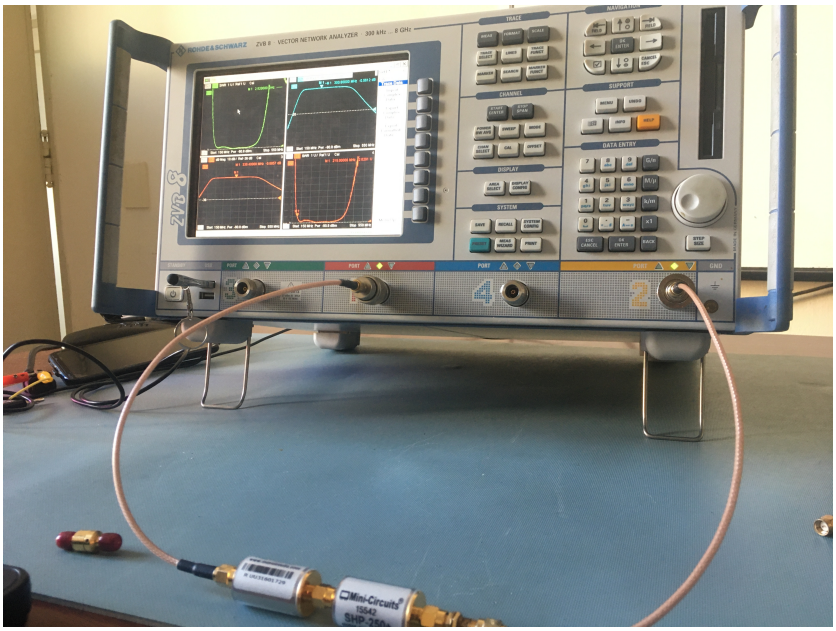


Figura 7.2: ROHDE & SCHWARZ ZVB 8 - VNA con el conjunto LPF-HPF en el ambiente de laboratorio.

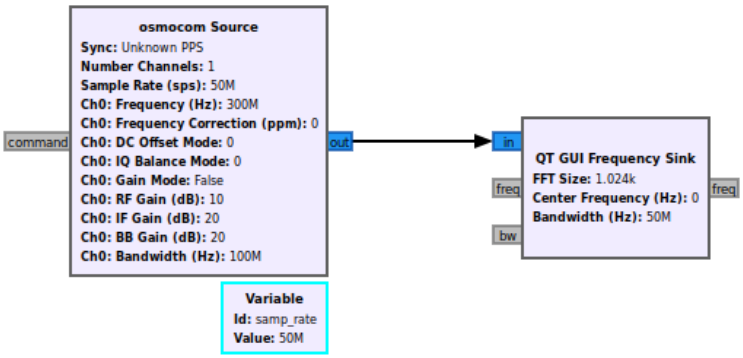


Figura 7.3: Flowgraph en GNU Radio Companion para prueba de hardware.

Se seleccionó un rango de frecuencias donde registrar las medidas que muestre toda la sección de interés sin perder resolución en frecuencia. Los puntos en la pantalla del VNA son limitados y la frecuencia de muestreo de su conversor analógico-digital afecta la separación entre puntos (resolución). Se establecieron los valores Start Freq. = 150 MHz y Stop Freq. = 550 MHz.

Una vez seleccionada la frecuencia, se utilizó el hardware de calibración para deshacerse de los efectos de pérdidas por reflexión de onda e impedancias no matcheadas en los cables coaxiales SMA del VNA. Esto se realiza mediante un procedimiento en el que el VNA se calibra automáticamente, pidiendo que el usuario le coloque entre la entrada y la salida el hardware de calibración. Luego, en distintos pasos, se varían cuáles de los múltiples bornes coaxiales SMA del hardware de calibración se deben conectar a la entrada y cuáles se deben conectar a la salida.

El requisito ($SWR_{IN} < 2dB$) queda determinado por el LNA. No se cuenta con un método para atenuar sus 2 dB, aparte de conseguir un amplificador LNA más especializado a cambio de un mayor costo. Se concluye que este valor de SWR_{IN} está en el borde de los requisitos pedidos y es suficiente para no distorsionar las características de la señal recibida.

7.1.2. Medidas GNU Radio

Para tomar las medidas de todo el sistema se utiliza el flowgraph de la Figura 7.3 y se verifica que se tiene la ganancia deseada de al menos 20 dB, asegurándose que no saturar el SDR, lo cual puede verificarse fácilmente con un bloque de GNU Radio para visualizar el espectro de los números complejos recibidos. Se agregó el uso de un Generador Agilent E4438C ESG VECTOR SIGNAL GENERATOR para controlar la entrada al sistema.

En el generador de señales Agilent, se colocó una señal sinusoidal pura a 300 MHz (centrada en el área de interés), con potencia -70 dBm (un valor razonable que no sature el sistema). Esto fue cableado por un cable SMA coaxial a la entrada

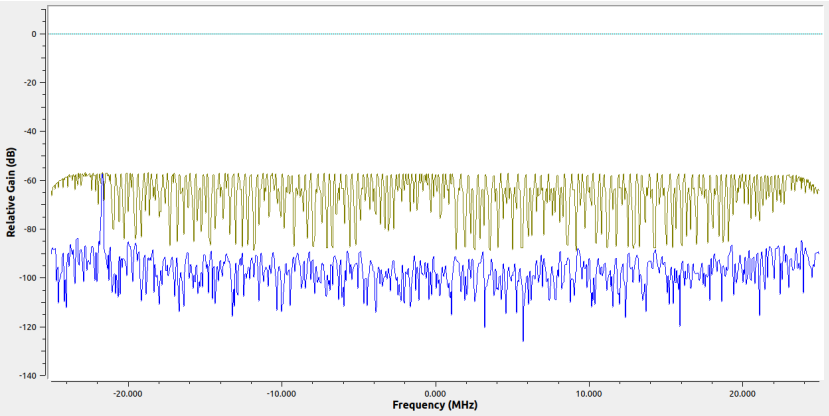


Figura 7.4: Respuesta en frecuencia dentro de GNU Radio con centro en 300 MHz, muestreando 50 MSps Generador Agilent LNA-LPF-HPF B200mini

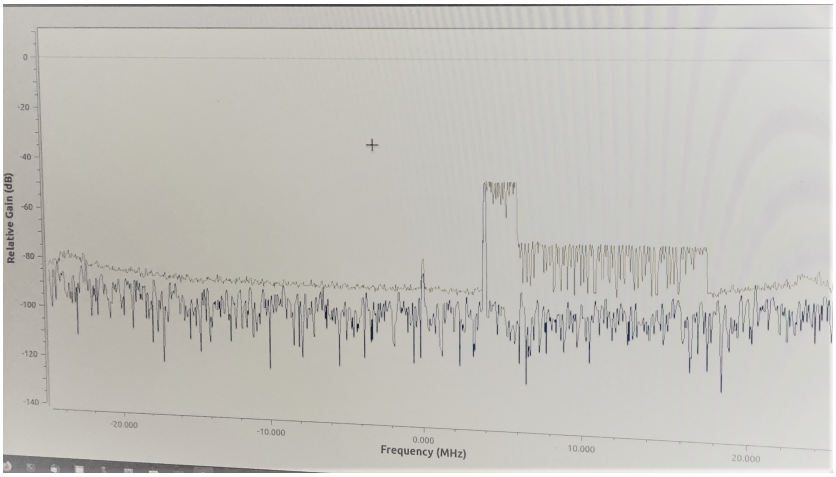


Figura 7.5: Prueba de Ganancia de 22dB relativos en Gnuradio QT GUI Freq Sink. Agregando el LNA y removiéndolo para apreciar los cambios.

del amplificador LNA.

En el resultado de la Figura 7.4 se aprecia una respuesta plana en toda la banda de interés, lo cual verifica las pruebas de transferencias y SWR.

En la Figura 7.5 se puede observar, no muy claramente, que existe una ganancia de 22 dB relativos gracias al LNA. Esto se comprobó retirando el LNA de la cadena y comparando la altura de la delta. Resultando en -46 dB (con LNA) y -69 dB (sin LNA). Se ve el máximo correspondiente al valor con LNA, y a la derecha está el valor sin LNA.

Los resultados obtenidos confirman que el sistema RF front-end verifica los objetivos para los que fue diseñado, siendo un medio apropiado para la implementación del software sobre el que se centra el proyecto.

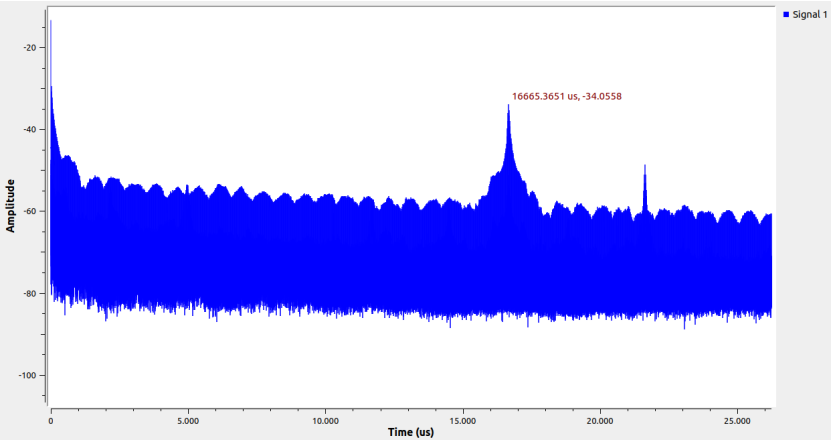


Figura 7.6: Autocorrelacón de las muestras provenientes del HW

7.2. Resoluciones Inferidas

La funcionalidad de inferencia de resolución, a diferencia del resto de las implementaciones de software, tiene una cierta simpleza en su evaluación. Debe poder obtener la información completa de la resolución del monitor espiado para cualquier resolución que reciba.

Dado que el RF front-end fue diseñado para operar en 1080p, esta es la señal de referencia para realizar el análisis. Luego, quitando el filtrado pasabanda en la señal de radiofrecuencia, se generaron grabaciones en diversas resoluciones para alimentar la entrada del bloque y evaluar su capacidad de inferir la información deseada.

En la Figura 7.6 se muestra la gráfica de la autocorrelación entre las muestras provenientes del hardware. La gáfica presenta dos picos notables a analizar. El primero, en $t = 0$, es el máximo de los dos por propiedad de la autocorrelación. El segundo, en $t = 16{,}665$ ms, es el tiempo en el cual se trasmiten todas las muestras de un cuadro completo y se genera el máximo relativo de interés (el inverso de este tiempo será, por definición, $f_v = 60$ Hz).

La gráfica, como era de esperar, está conformada por una sucesión de picos, correspondientes a los máximos relativos de la autocorrelación entre líneas horizontales. La distancia entre estos picos, aproximadamente $\Delta t = 14{,}837 \mu s$, es el tiempo transcurrido entre la primera muestra de una linea y la de la siguiente. Recordando la igualdad (4.3), se tiene que $P_y = 1123$, lo cual da una noción de que el funcionamiento del *FFT_Autocorrelate* es apropiado, ya que la resolución 1080p totaliza 1125 píxeles verticales incluyendo el *blanking*. Esto está atado a que el resultado temporal presentado resulta de colocar el cursor donde figura el pico, con lo que el pulso y visión del usuario introducen incertidumbre, pero estos son meros estimados para hacerse una idea del correcto funcionamiento del primer bloque. El bloque de inferencia de resolución maneja internamente los máximos exactos hallados, con lo que esta aproximación no le aplica a su análisis.

De la mano del resultado planteado en base a aproximaciones, surge una con-

sideración a tener para los valores intermedios de los que se vale el bloque de inferencia. La norma VESA establece los tamaños para las diferentes resoluciones incluyendo *blankings*, y los valores verticales contenidos en este estándar son los únicos resultados válidos que considera el algoritmo. De esta manera, se tienen umbrales importantes para la selección de la resolución en base al valor de píxeles verticales obtenidos, lo cual aporta a la robustez del bloque en su operativa.

Las mencionadas grabaciones fueron tomadas para todas las resoluciones disponibles en los monitores del ambiente de laboratorio y probadas una a una como entradas del bloque de inferencia, obteniendo en cada una el resultado deseado. En cada caso, en la terminal se imprimen todos los valores de interés que serán necesarios más adelante para el propio espionaje.

Además, se probó el bloque con el sistema obteniendo las muestras directamente del hardware, resultando igualmente en los valores correctos de resolución para las imágenes de video espiadas.

Tanto online como offline, señales claras o ruidosas, con y sin pérdidas de muestras, derivan en autocorrelaciones correctas que permiten hallar acertadamente la resolución correspondiente a la imagen del monitor espiado. El bloque busca el máximo y determina los parámetros con razonable robustez ante cambios abruptos de la autocorrelación por la no idealidad de la señal, concluyendo que el objetivo de inferencia de resolución en el sistema se alcanza satisfactoriamente.

7.3. Sincronización

Se introdujo la implementación del bloque de detección de sincronización horizontal y vertical de la imagen estableciendo un objetivo claro: fijar la imagen de forma que se observe de forma continua el video del monitor espiado, en lugar de verse cortado por los *blankings* vertical y horizontal. Asimismo, se desea que sea robusto ante movimientos de la imagen de video o saltos grandes que se den eventualmente por pérdidas de muestras u otros motivos.

En esta sección se busca establecer los criterios para evaluar al bloque en su cumplimiento de estos objetivos. Estos criterios no son tan directos de definir, puesto que no hay una métrica fácil de establecer para su evaluación. En su lugar, se aplica el algoritmo para diversas imágenes de video con muestreo corregido, tanto online como offline, para observar su capacidad de centrar la pantalla en las dos direcciones.

Se cuenta con varias grabaciones utilizadas para el desarrollo y evaluación del bloque de inferencia de resolución. Se cuenta, además, con un conjunto de grabaciones previas que se generaron en el desarrollo de gr-tempest original y para pruebas de la tesis de Pablo Menoni. Elementos de ambos conjuntos se utilizan para realizar las pruebas mencionadas. Las Figuras 7.7 y 7.8 muestran los resultados en dos escenarios distintos.

Puede verse que se sincroniza correctamente, pudiendo reconocer la ubicación de los *blankings* y comenzando el despliegue a partir de las mismas, aun en condiciones de ruido distintas. Se puede distinguir que el punto exacto del *blanking* a

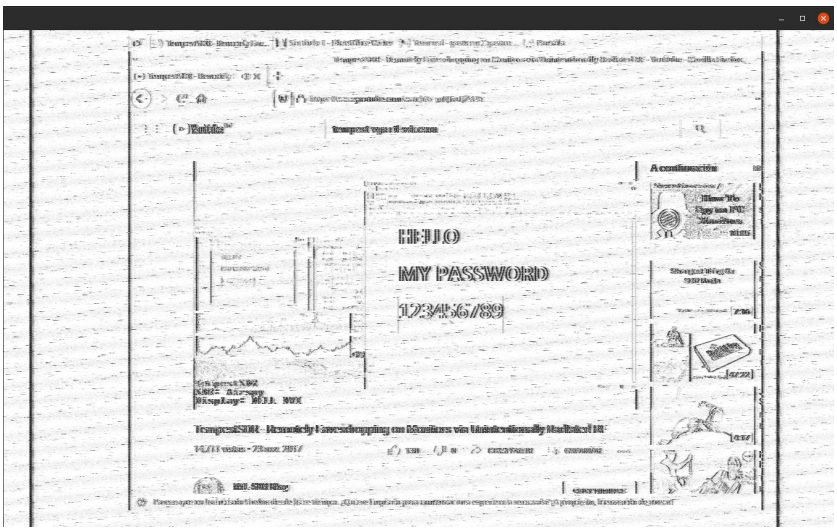


Figura 7.7: Grabación de resolución 1024x768 con despliegue centrado por Sync Detector.

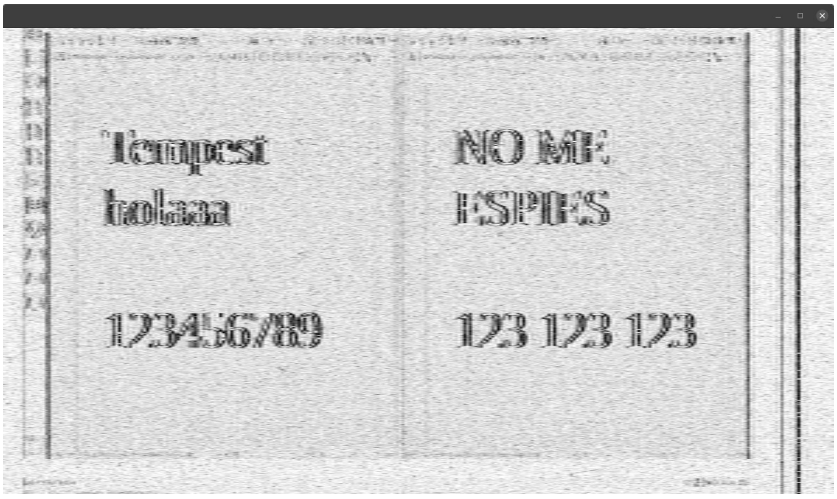


Figura 7.8: Grabación de resolución 1920x1080, ruidosa, con despliegue centrado por Sync Detector.

partir del cual se despliega es ligeramente distinto, pero se consideran correctos en tanto no haya discontinuidades en la imagen de video.

Una modalidad que se considera importante para la evaluación de las funcionalidades de software es la ya introducida simulación de la señal, dado que permite reconstruir la imagen espiada en un ambiente en el que las no idealidades pueden ser establecidas y ajustadas por el usuario. En la Figura 7.9 puede verse el *flowgraph* para la simulación mencionada: los píxeles generados por la fuente son conformados por un pulso rectangular e interpolados para emular continuidad en las muestras. Luego se modula a bandabase y se le hace *resamplng* para pasar a un escenario similar al que resultaría del conversor analógico-digital incluido en el SDR. Finalmente, un modelado de canal permite incorporar las no idealidades

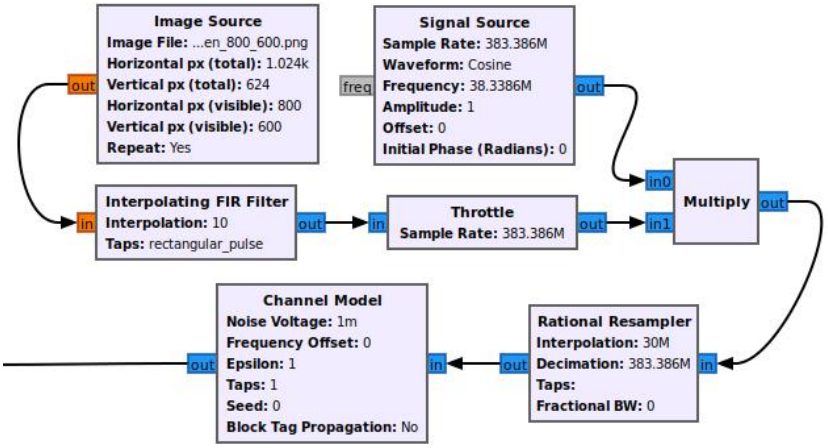


Figura 7.9: Parte del diagrama utilizado para simular la recepción de una señal.

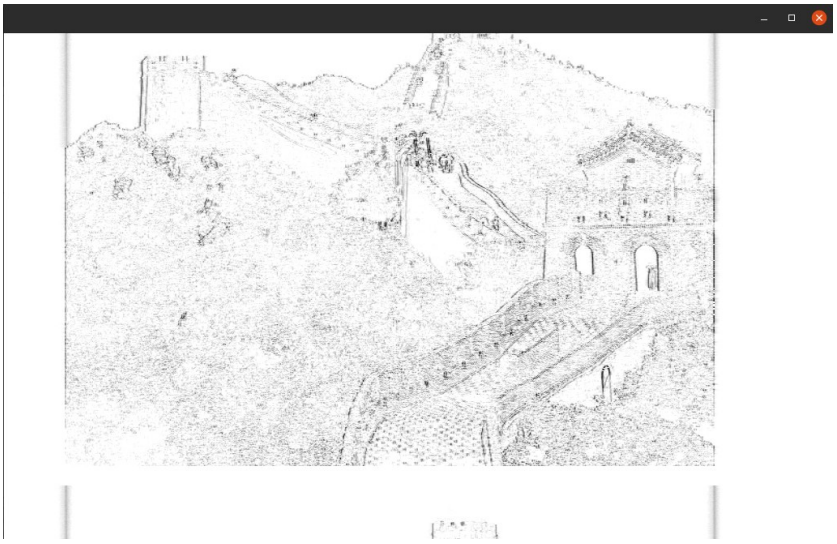


Figura 7.10: Imagen simulada con despliegue centrado por Sync Detector.

mencionadas. Este sistema será de utilidad en la evaluación de este y otros bloques.

Varias imágenes fueron añadidas al escenario simulado, generando sus correspondientes señales de video espiado. Las Figuras 7.10 y 7.11 muestran el resultado de aplicar la detección de sincronización a dos fotografías (imágenes fijas).

Se puede ver que la detección horizontal se da correctamente en ambas. Verticalmente, sin embargo, puede notarse que los corrimientos son distintos, llegando al punto de cortar la imagen. Esto tiene dos posibles explicaciones, que seguramente afecten el resultado simultáneamente: por un lado, *blankings* más pequeños son más difíciles de distinguir contra el resto de la imagen, por más que la proporción en píxeles sea similar; por el otro, los límites de la imagen no están marcados de la manera que lo están para los *blankings* horizontales, con lo que la distribución de la energía en los promedios utilizados para la búsqueda tienen una mayor variación.

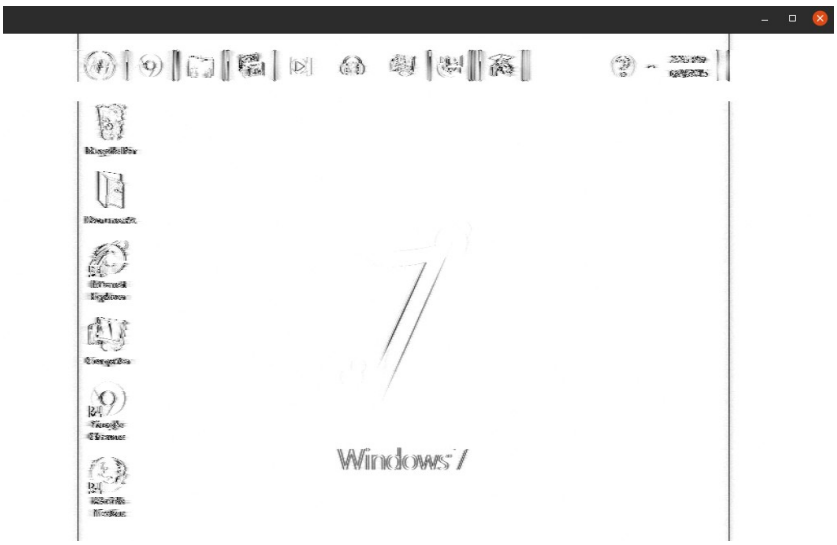


Figura 7.11: Escritorio simulado con despliegue centrado por Sync Detector.

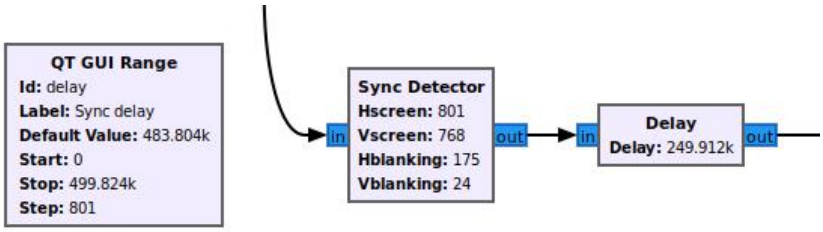


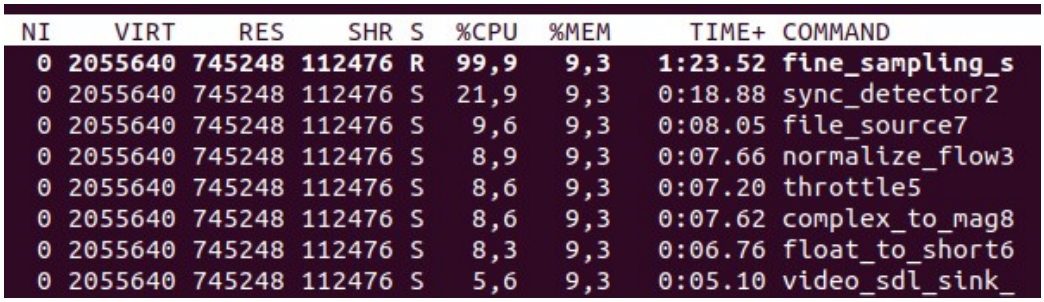
Figura 7.12: Retraso añadido al Sync Detector para el ajuste vertical manual.

El problema presentado para la búsqueda del *blanking* vertical tiene un atenuante fundamental: más allá de dónde se lo encuentre, el cálculo se da sobre una distribución energética consistente, con lo que la imagen es siempre fijada verticalmente. De esta manera, el problema es fácilmente subsanable con un retraso vertical como el del la Figura 7.12.

El bloque *Sync Detector* centra la imagen horizontalmente y la fija verticalmente, para que luego el usuario realice el centrado vertical de forma manual (vía un *slider* en la GUI que hace variar el valor del retraso en pasos del tamaño de una línea horizontal).

Las pruebas de esta operativa en el funcionamiento online confirman el correcto funcionamiento de este conjunto, logrando que la imagen quede completamente centrada con solo un ajuste manual al comienzo del espionaje. El único cuidado a tener, según se comprueba en el espionaje online, es que cambios significativos en el contenido de la imagen alteran el promedio de la energía y por tanto el resultado del cálculo del *blanking* vertical. Por este motivo, movimientos como el cambio de pestaña o apertura o cierre de aplicaciones pueden requerir un ajuste manual del corrimiento vertical, que luego quedará fijo.

En lo relativo a la robustez del algoritmo, hay tres escenarios que interesa



NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
0	2055640	745248	112476	R	99,9	9,3	1:23.52	fine_sampling_s
0	2055640	745248	112476	S	21,9	9,3	0:18.88	sync_detector2
0	2055640	745248	112476	S	9,6	9,3	0:08.05	file_source7
0	2055640	745248	112476	S	8,9	9,3	0:07.66	normalize_flow3
0	2055640	745248	112476	S	8,6	9,3	0:07.20	throttle5
0	2055640	745248	112476	S	8,6	9,3	0:07.62	complex_to_mag8
0	2055640	745248	112476	S	8,3	9,3	0:06.76	float_to_short6
0	2055640	745248	112476	S	5,6	9,3	0:05.10	video_sdl_sink_

Figura 7.13: Indicador de consumo de Linux para distintos procesos operando en el escenario simulado con el sistema original y el bloque Sync Detector.

evaluar: en primer lugar, una imagen totalmente quieta (muestreo perfectamente corregido); luego, la imagen con ligeros corrimientos (pequeño error en el muestreo); por último, efecto de pérdida de muestras repentinas o *glitches* en la señal de video.

Se pudo comprobar en las pruebas anteriores que, para el escenario ideal en el que la imagen se logra fijar, el resultado del *Sync Detector* es siempre el deseado. Por otra parte, una grabación cuyo muestreo no logra corregirse completamente permite evaluar el segundo escenario: la imagen, luego de la corrección de muestreo, tiene un movimiento paulatino y constante en un sentido de la dirección horizontal. Al aplicarle la detección de sincronización a ese resultado, se ve cómo la corrección se aplica como se espera y, al avanzar el pequeño movimiento, se vuelve a ajustar hacia la izquierda. Esto resulta en un pequeño y prácticamente imperceptible movimiento oscilatorio que no altera la capacidad del usuario de visualizar la imagen del monitor espiado.

Se concluye con estos escenarios que la robustez del algoritmo es mayor a la esperada, puesto que el diseño original está hecho para imágenes completamente fijas.

El último punto a considerar en este sentido es la respuesta a *glitches* o saltos aislados en la imagen. No solo se corrigen correcta y rápidamente, pudiendo retomar el estado anterior al salto, sino que la corrección se realiza más rápido que en el sistema TempestSDR, sobre el que se basa el algoritmo. Esto, junto con lo mencionado sobre sincronización vertical, puede evidenciarse y contrastarse con su versión anterior en el video [34].

Finalmente, y como consecuencia de ser parte de un sistema más grande que lleva a cabo varios procesos en simultáneo, es pertinente un breve análisis del consumo de CPU del *Sync Detector*. La Figura 7.13 muestra la salida de la función *top -H* ingresada por línea de comandos en la terminal de Ubuntu. Allí se destaca el consumo total del bloque en términos de porcentaje del CPU.

El valor de 21,9 % (20 % en promedio) sugiere un bajo requerimiento de procesamiento en función al resto del sistema, lo cual es esperado. Como referencia, se tiene que el bloque original *Hsync* acarreaba un consumo promedio de 72 % del CPU para un sistema idéntico, con lo que se puede concluir que el resultado es muy positivo dado el aporte a la optimización de los recursos utilizados para

sincronizar.

Adicionalmente, se agrega la posibilidad de habilitar y deshabilitar la funcionalidad de sincronizarse por comando PMT, para que no tenga la necesidad de operar desde el comienzo. Un primer motivo para esto es que los cálculos de FFT realizados por otros bloques se dan al principio de la ejecución, con lo que interesa darle la mayor capacidad de procesamiento a esa etapa. El segundo motivo, quizás más claro, es el ya explicado: el algoritmo está diseñado para operar sobre una señal con el muestreo ya corregido. Pierde sentido ejecutar la funcionalidad antes de alcanzar el índice de interpolación deseado porque, por un lado, no se sincronizará correctamente. Por otro, dificultará la tarea del usuario de evaluar el avance en la corrección de muestreo, generando movimientos que no son consecuencia de ese avance sino de un algoritmo mal implementado.

El resultado de lo desarrollado en la sección es un bloque de detección de sincronismo que cumple con los objetivos para los que se implementó, incluyendo las consideraciones pertinentes para operar en el contexto de un sistema con requerimientos varios que lo afectan directa e indirectamente.

7.4. Cálculo del Índice de Interpolación

El bloque *FFT_peaks* se introduce en la Sección 6.3 como sustituto del *Fine_Sampling_Synchronization* en el cálculo de la corrección de muestreo, por lo que su objetivo implica sostener las acciones exitosas de este último mientras corrige la problemática explicada su imposibilidad de converger a una solución si no realiza el cálculo correctamente en el primer intento.

Ese objetivo permite definir los criterios a considerar para la evaluación del bloque generado, reduciendo su escenario de pruebas al mismo visto hasta ahora pero con el conjunto *FFT_Autocorrelate-FFT_peaks* trabajando en paralelo al *Fine_Sampling_Synchronization* y proveyéndole del índice vía mensajes PMT, en lugar de que este último sea quien lo calcule.

Los criterios se reducen a tres principales puntos a evaluar: que el muestreo efectivamente se corrija en la imagen resultante sin tiempos que puedan considerarse excesivos para el usuario, que logre hacerlo para cualquier escenario (sin depender del cálculo inicial) y que su consumo de procesamiento no empeore significativamente afectando la operativa global.

La evaluación del primer punto consiste simplemente en correr el bloque para varios escenarios online, offline y simulados, que permitan confirmar que el muestreo es corregido. En primer lugar, se prueba el bloque en el escenario de la Figura 7.14.

Tanto al operar con hardware como en base a grabaciones, el bloque siempre llega a calcular la corrección de muestreo de manera que la imagen desplegada muestre las lineas horizontales alineadas verticalmente. Es decir, que el resultado logra evitar la situación de la Figura 6.5.

El resultado mostrado hasta ahora iguala al del bloque anterior, con un pequeño defecto: el cálculo se repite continuamente y se aplica cada ciertas iteraciones del *work* del *FFT_peaks*. Esto genera que, si la convergencia no se alcanza en pocas

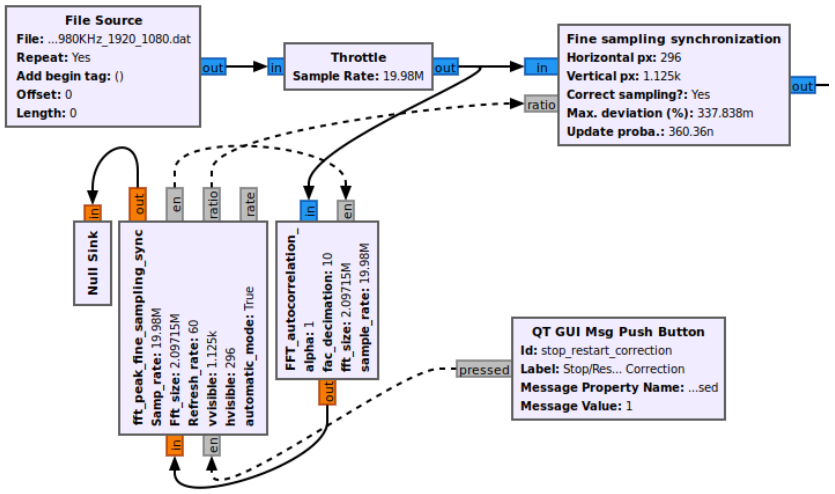


Figura 7.14: Parte del diagrama de flujo que implementa el FFT peaks comandado por botón, pasándole el índice al Fine Sampling Synchronization para que interpole.

repeticiones del cálculo, el tiempo de corrección puede aumentar. Sin embargo, solo algunas excepciones (operando a altas frecuencias de muestreo) requirieron tiempos de hasta 15 segundos para converger. En los demás casos, la media temporal ronda los 5 segundos y comienza a desplegar las líneas verticales de la imagen paralelas a las líneas verticales del monitor espía.

Una segunda consecuencia de esta continuidad de cálculos por iteraciones es que, si una vez obtenida la corrección, se siguiera calculando y se encontrara alguna pérdida de muestras u otro motivo que pudiera generar anomalías en la autocorrelación, la sincronización de la frecuencia de muestreo se perdería, retornando a valores que distorsionan la imagen.

Para atacar esta desventaja se cuenta con el mencionado botón de deshabilitación y rehabilitación del bloque. Por más que su propósito principal sea el ahorro de procesamiento en un proceso que no requiere ser ejecutado luego de fijar la corrección, cumple el rol adicional de detenerlo en el valor deseado y evitar que se modifique erróneamente.

La prueba restante para evaluar la capacidad de corrección de muestreo del bloque es la que lo requirió en primer lugar: la corrección en el escenario simulado. Efectivamente, la misma operativa ejecutada en un contexto de simulación como el presentado en la sección anterior da lugar a un resultado cuyo muestreo es completamente ajustado, dejando la imagen fija y sus bordes perpendiculares. Esto confirma que la implementación corrige la característica que se deseaba corregir y puede ser visto en el video [35].

El último punto consiste en contrastar la operativa del nuevo conjunto con el sistema anterior en términos del porcentaje de CPU que requiere para su ejecución. Como se vio en la Figura 7.13, el proceso de interpolación original (incluyendo el cálculo) rondaba el 100 % para el *Fine_Sampling_Synchronization*. En el nuevo sistema, el mismo bloque consume un aproximado del 60 % de CPU en su tarea

de interpolar. El *FFT_peaks*, por su parte, tiene un consumo promedio de 70 % mientras realiza el cálculo, para reducirse a 0 % una vez que el valor es alcanzado y se señaliza con el botón su deshabilitación.

A partir de lo expuesto, es posible concluir que la implementación del bloque *FFT_peaks* es buena, cumpliendo los objetivos para los que se diseñó a un costo (tanto temporal como computacional) razonable para la aplicación.

7.5. Rendimiento de CPU

La implementación de los cambios propuestos por la realimentación del sistema basado en SDR y la operativa del *Frame Dropper* tienen el objetivo de reducir el procesamiento requerido por el sistema, de forma de mantener el funcionamiento a mayores frecuencias de muestreo que reconstruyan una mejor imagen de video.

El procesamiento individual de cada bloque implementado en el sistema busca ser reducido mediante el uso consistente de la librería VOLK para optimizar diversas operaciones a realizar sobre las muestras. Sin embargo, ya se abordó el tema de que la mayoría del peso está dado por la interpolación de muestras para la corrección de muestreo, por lo que el apoyo en alguno de los métodos presentados se vuelve fundamental para lograr el funcionamiento deseado.

Dado que el objetivo se basa en mejorar el rendimiento del sistema implementado, se utilizará como referencia el consumo del sistema original para realizar las observaciones sobre los resultados de cambiar de métodos. Se espera que el procesamiento del CPU a causa de la operativa de Tempest se vea reducido. Esto habilitará el aumento la frecuencia de muestreo, traduciéndose en menos muestras perdidas para frecuencias que, en el sistema original, sobrecargarían la misma PC.

7.5.1. Corrección en Software

Se analiza, en primer lugar, la opción de continuar corrigiendo el muestreo en software, intentando reducir el consumo en el proceso. El bloque implementado para la reducción del procesamiento, *Frame Dropper*, se debe evaluar en los dos requerimientos que se plantearon para su implementación: que baje la tasa de datos que recibe el interpolador para reducir el poder de cómputo requerido para ejecutarlo, sin afectar la percepción de la imagen recibida a menos de una reducción en la frecuencia de refresco.

El primer requerimiento tiene implícito que no solo se reduzca el uso de CPU del proceso de interpolación, sino que la combinación de usos del *Frame Dropper* y el interpolador también sean menores. El segundo requerimiento, por su parte, tiene un carácter más subjetivo en su evaluación, por lo que se asemejará al criterio utilizado en la Sección 6.2.

Se vuelve a implementar un sistema simulado para establecer el ambiente de pruebas del bloque. Dado que depende del factor de interpolación calculado por fuera, debe ser integrado al funcionamiento de los demás bloques para probarse correctamente. En la Figura 7.15 se muestra el flujo utilizado para el funcionamiento.

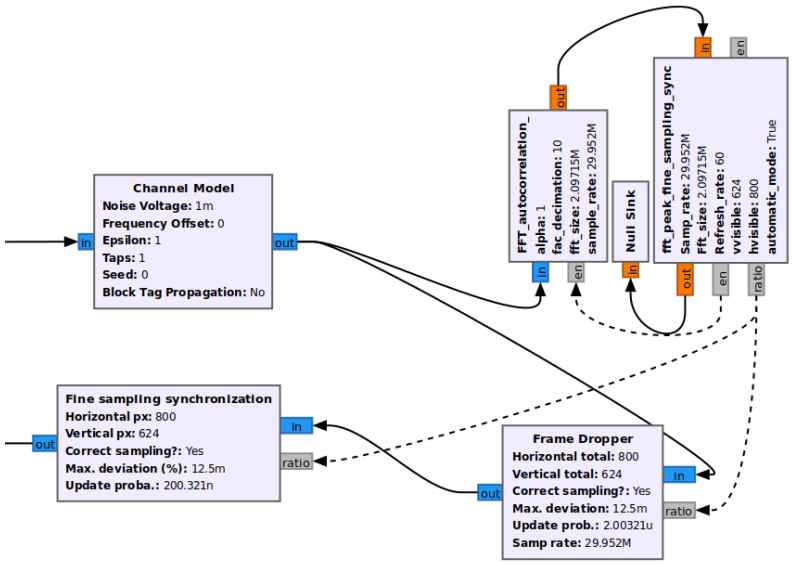


Figura 7.15: Parte del diagrama utilizado para implementar el Frame Dropper en el sistema simulado.

s

Tabla 7.1: Procesamiento requerido en porcentaje de CPU para los bloques Fine Sampling Synchronization y Frame Dropper, con distintos descartes de muestras.

Escenario	Frame Dropper	Fine Sampling Synchronization	Total
Sin Dropper	N/A	30 %	30 %
15 FPS	13 %	5 %	18 %
6 FPS	12 %	3 %	15 %
4 FPS	11 %	2 %	13 %

Se realiza una medición del consumo para el sistema de la figura. Luego, se repite en las mismas condiciones pero sin el *Frame Dropper*. La Tabla 7.1 contrasta los resultados obtenidos en cada caso, con la consideración de que sus valores son relevados una vez que se estabiliza el cálculo para la corrección de muestreo (en el inicio son mayores).

De estos resultados se puede apreciar que la reducción del consumo es significativa sobre el interpolador. Se destaca que la reducción de su procesamiento es superior a la lineal con respecto a los cuadros por segundo descartados en el caso de 15 FPS (descartando tres cuadros por cada uno que se deja pasar). En este sentido, el bloque logra su objetivo de reducir el consumo del interpolador.

En términos de valores totales, se puede ver que el procesamiento se reduce a la mitad al bajar a una tasa de refresco de 6 cuadros por segundo. Por más que la reducción porcentual sea menor con respecto al del interpolador, se considera que el resultado verifica el objetivo.

El otro punto a considerar es la calidad de la imagen resultante. La Figura

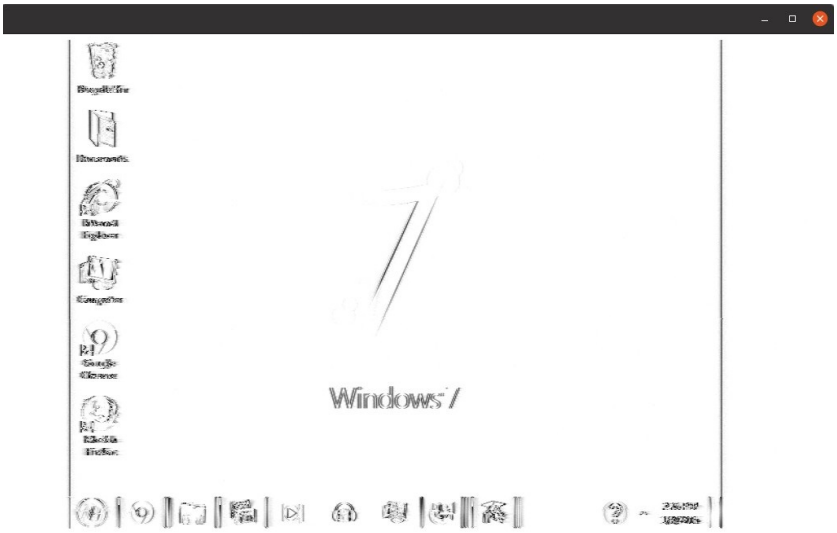


Figura 7.16: Simulación procesada por sistema con Frame Dropper.

7.16 muestra la salida del sistema simulando que implementa el *Frame Dropper*. La misma reconstruye correctamente la señal simulada, tal como si no se hubiera añadido el bloque en cuestión.

Dada la complejidad requerida para la sincronización, este se considera un resultado ampliamente satisfactorio en la aplicación simulada; más aún con la consideración de los resultados de consumo arrojados. El complemento del resultado obtenido es el uso del mismo bloque en casos de aplicación real, ya sea online u offline. En la misma se puede ver que existe un pequeño desfasaje en la imagen, como si solo algunas muestras fueran perdidas en el pasaje de cada cuadro. Esto es particular dado que no ocurría en el escenario simulado.

Utilizando una grabación, se obtiene para el mismo bloque un resultado como el mostrado en la Figura 7.17.

Un arduo análisis llevó a la prueba de muchas implementaciones que concluyeron en la presentada en el capítulo anterior. En el camino, diversas formas de calcular el número de muestras requeridas para interpolar fueron analizadas. La seleccionada toma varias precauciones (algunas ya mencionadas) con respecto al momento de calcular el número con el mismo método que el bloque interpolador.

El hecho de traducirse en unas pequeñas muestras, que además aumentan al aumentar el número de cuadros descartados por cada uno que pasa, lleva a pensar que se pueda tratar de un problema vinculado al redondeo de los cálculos para cada cuadro. Sin embargo, si este fuera el caso, se apreciarían las mismas consecuencias en el escenario simulado.

Es por esto y por sus buenos resultados en procesamiento que se resuelve añadir el bloque al proyecto de todas formas; se considera útil tener una alternativa en software para la reducción de consumo de CPU ya que la metodología por hardware acarrea la mencionada pérdida de generalidad según el equipo SDR que se utilice. Queda para futuras iteraciones de gr-tempest la completa corrección de

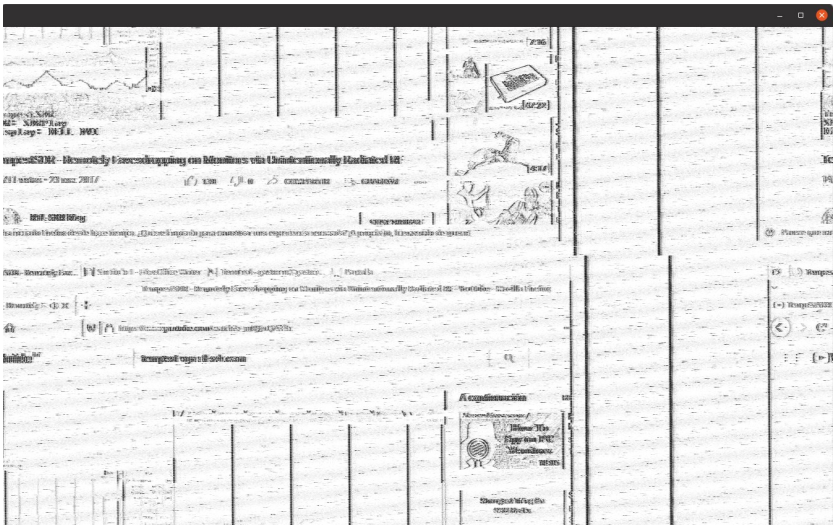


Figura 7.17: Grabación procesada por sistema con Frame Dropper.

este desplazamiento entre cuadros.

7.5.2. Corrección en Hardware

La concepción de corregir el muestreo mediante hardware es, por un lado, idónea, sacando la máxima ventaja de los beneficios de SDR y optimizando la distribución de recursos, para que el procesamiento en software pueda destinarse a mejorar el despliegue de la señal. Por otro lado, se introdujo que puede implicar una pérdida de generalidad, puesto que los formatos de mensaje presentados en la Sección 6.4.1 pueden no ser compatibles con todos los drivers de SDR a utilizar en el *framework*.

De todas formas, y en el marco de los resultados expuestos en el punto anterior, se define la corrección en hardware como método central para la reducción de procesamiento en el sistema de espionaje online, para el cumplimiento del objetivo de muestrear a 50 MSps.

La implementación de la realimentación se ejecuta como se propuso, encontrando resistencia por parte del *USRP Source* que, aún sin procesos de interpolación, notifica de muestras perdidas por *overflow* a través de la terminal de GNU Radio al operar a 50 MSps.

Se distingue que el proceso de mayor consumo en este escenario es el conjunto *FFT_Autocorrelate-FFT_peaks*, dado que a lo analizado en la Sección 7.4 se le agrega una cantidad significativa de muestras al pasar a la tasa deseada. Esto implica que, para ver la misma franja temporal para el cálculo de autocorrelación, se debe manejar un mayor *fft_size* para los cálculos.

Es por esto que debe buscarse una forma de disminuir el procesamiento del conjunto mientras realiza sus cálculos. Se presentó al desarrollar la implementación del bloque *FFT_peaks* que su funcionalidad es llevada a cabo desde el principio de la ejecución del GRC, pero que se cuenta con un botón para que pueda deshabilitarse

al encontrar la corrección del muestreo puesto que (al menos temporalmente) no es necesario continuar el cálculo. El mismo botón permite volver a habilitarlo en caso que fuera pertinente.

Gracias a esta metodología, surge una posibilidad para el escenario realimentado: durante el cálculo es necesario mayor poder de cómputo y es prioritario hallar la corrección de muestreo. No es sino hasta concluir esa corrección que pasa a ser prioridad el visualizar correctamente la imagen de video. Por este motivo y por lo aprendido en la sección anterior, se evalúa el descarte de muestras al comienzo como facilitador del cálculo de autocorrelación y corrección de la frecuencia de muestreo.

La principal problemática del *Frame Dropper* es determinar la cantidad de muestras a descartar para que se correspondan con un cuadro completo luego de interpolar. Como se vio, las consecuencias de no calcular esas muestras correctamente se traducen en un corrimiento horizontal de una fracción del cuadro, dado por el conjunto de muestras que falta o sobra en los cuadros descartados. En el escenario realimentado, sin embargo, no hay interpolación, por lo que el tamaño de un cuadro será conocido y el *Frame Dropper* sabrá exactamente el número de muestras a descartar. Es decir que la traba que se le había identificado deja de representar un problema.

Sin embargo, el consumo del *Frame Dropper* sí puede llegar a ser un impedimento en la búsqueda de minimizar el procesamiento durante la etapa de corrección de muestreo. Se trae a consideración la utilidad que puede aportar el bloque *Keep 1 in N* del *Core* de GNU Radio. El mismo es capaz de dejar pasar y descartar cantidades grandes de muestras, permitiéndo tanto que cálculo de autocorrelación se realice correctamente como que el usuario sea capaz de distinguir cuando el muestreo fue corregido y lo detenga con el botón.

El bloque *Keep 1 in N* se utiliza, este caso, con los agregados de pasaje de serial a paralelo *Stream to Vector* de tamaño *fft_size* y el consiguiente pasaje de paralelo a serial *Vector to Stream* dentro del *hier block Keep_1_in_N_Frames*. Es la técnica empleada para el *FFT_Autocorrelate*, como se vio en la Sección 6.1.

Algo que resulta práctico de este bloque es su capacidad de modificar el valor de N en tiempo de ejecución. Ajustar este parámetro con un *slider* resulta en una experiencia práctica para el usuario, en especial para tomar fotos con bajos FPS y poder volver a subirlos cuando el sistema se estabiliza.

De esta forma, la etapa inicial del sistema queda como en la Figura 7.27, donde las muestras son descartadas de a tramos grandes mientras la corrección de muestreo se termina de calcular. Una vez hallada, el SDR está operando de forma tal que ninguna interpolación es necesaria y el conjunto *FFT_Autocorrelate-FFT_peaks* puede deshabilitarse, dejando a la operativa solo los procesos de centrado y despliegue con muestreo cercano a los 50 MSps (no idéntico pues ya fue corregido).

En la Figura 7.19 se puede visualizar la imagen desplegada durante el proceso de cálculo (a) y luego de quedar operando con el muestreo corregido (b). Se aprecian resultados muy positivos con respecto a la calidad de la imagen, prácticamente sin muestras perdidas a una alta frecuencia de muestreo, verificando el objetivo propuesto.

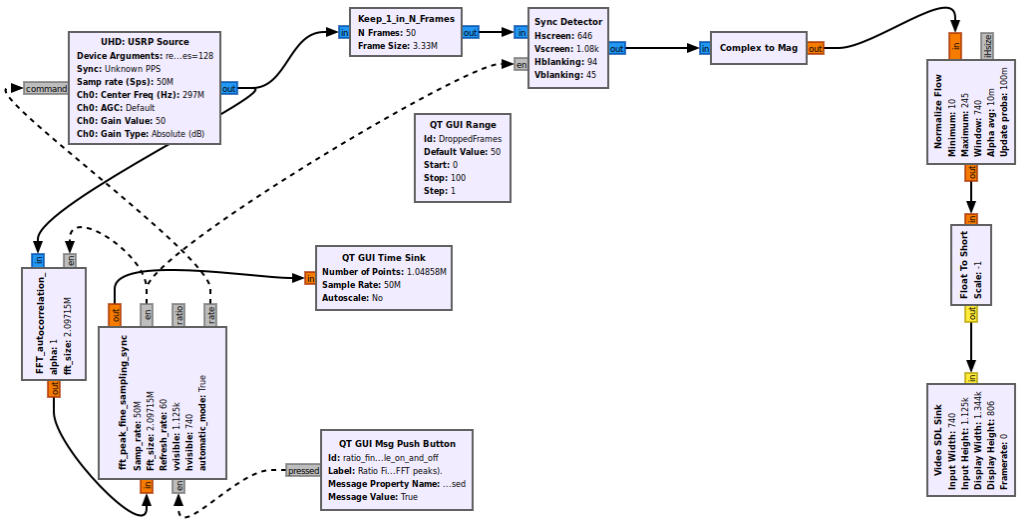


Figura 7.18: Diagrama utilizado para implementar la corrección del muestreo por hardware a 50 MSps.

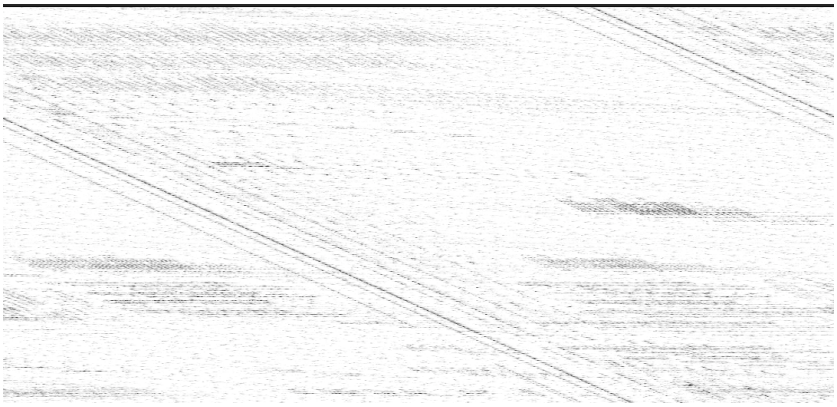
Cabe destacar la utilización de los siguientes valores en los parámetros del bloque USRP Source:

```
Device Arguments = "num_recv_frames = 128, recv_frame_size = 65536"
```

Estos parámetros determinan la cantidad de espacio en memoria que se encuentra disponible para que el driver del USRP reciba datos del dispositivo Ettus en la "capa de transporte". Se refiere a la entidad de software encargada de la interfaz entre el I/O del dispositivo y la interfaz receptora de paquetes que ejecuta el driver UHD en la PC, denominada *transport object* [36] . Dicha entidad es una clase de C++ dentro del driver, encargada del manejo de memoria para la recepción de los datos seriales de los dispositivos soportados [37].

En definitiva, los valores por defecto son 32 cuadros, de tamaño 1472 bytes. Es decir que hay 47,1 kB de espacio en memoria donde el driver puede ubicar paquetes recibidos del dispositivo antes de llenar todos los cuadros y que se empiecen a perder muestras. Mientras tanto, esos datos en memoria son consumidos por la aplicación. Entonces, considerando el *throughput* del hardware como una cantidad relativamente constante cuando se está en buen funcionamiento, incrementar el tamaño de memoria mencionado otorgará más tiempo a la aplicación para completar tareas antes de que se llene la entrada de datos de la "capa de transporte".

Esto ha sido comprobado en esta sección ya que con los valores por defecto resulta imposible aumentar la tasa de muestreo a más de 20 MSPs sin tener saltos en la pantalla constantemente, una o dos veces por segundo se genera un *overflow*. Y, con el cambio en el tamaño de los cuadros de recepción, se logra espiar en tiempo real una pantalla a 50 MSps, con ocurrencias de un *overflow* cada 5 o 10 segundos aproximadamente, lo cual hizo viable la implementación del sistema con suficientes cuadros del video para trabajar sin pérdidas de muestras.



(a) Salida mientras el índice de interpolación sigue siendo calculado.

(b) Salida con el índice de interpolación correcto.

Figura 7.19: Comparación de salida durante el cálculo de la corrección de muestreo

7.6. Imágenes Ecualizadas

El análisis de los resultados de ecualización, al igual que otros puntos vistos, tiene la dificultad de limitarse a un carácter cualitativo, en función de lo que el usuario percibe como una mejor imagen de video. Esto se refleja principalmente en todo lo que no es texto, puesto que el análisis este último sí puede tener un carácter más objetivo (aunque aún cualitativo) tratándose del límite entre poder o no poder identificarlo.

El orden a tomar la evaluación de la ecualización es el siguiente: se comienza mostrando los resultados de corregir el error en frecuencia sin ecualización lineal; luego, se analizan el método de ecualización lineal con y sin corrección de frecuencia. Para el caso sin corrección del corrimiento en frecuencia, se utiliza el módulo de las señales, con su ventaja y desventaja.

Para la obtención de las imágenes a presentar, se utiliza una grabación de espionaje de monitor 1024x768@60 muestreado a 50 MSps. En el monitor espiado

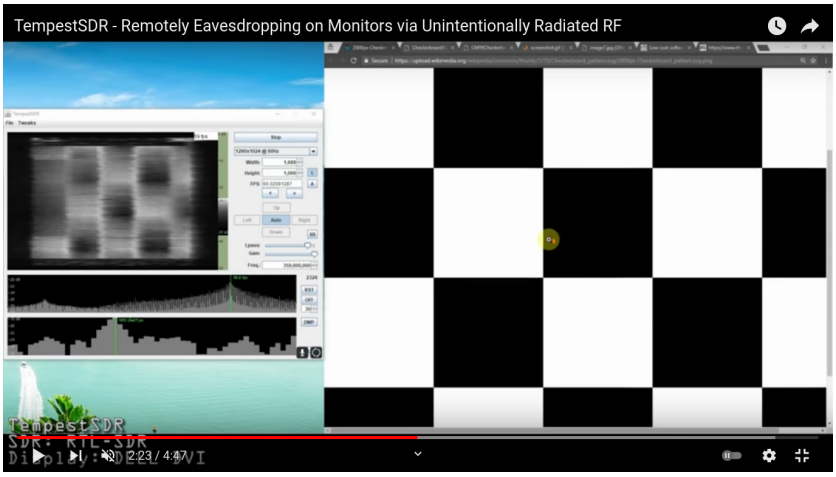


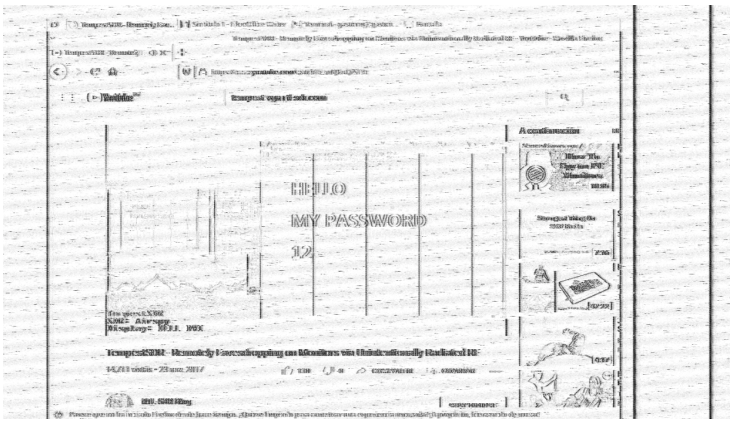
Figura 7.20: Captura real del vídeo en el monitor espiado

se está visualizando un video de la plataforma Youtube, cuyas imágenes, se podrá notar, tienen diferencias debidas al momento en el cual se realiza la captura de la imagen. En la Figura 7.20 se puede ver el video en uno de los momentos que es espiado.

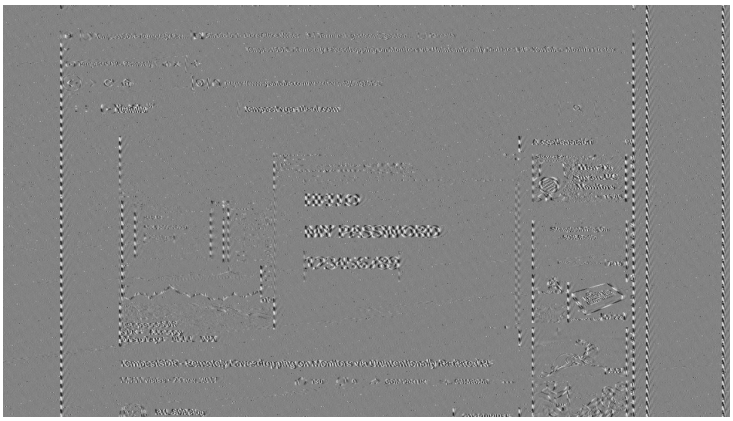
7.6.1. Corrección del error de frecuencia

En la Figura 7.21 se presentan, a modo comparativo, la imagen espiada tomando el módulo de la señal y la imagen espiada utilizando la parte real de la señal espiada $x'_R(nT_s)$ (con y sin corrección). Lo primero que se observa al cambiar en el sistema el bloque *Complex To Mag* por *Complex To Real* es la aparición más notoria de grises a lo largo de todo el cuadro debido a la ya adelantada pérdida de información del signo de la señal al tomar el módulo. Lo segundo es que las líneas verticales se ven con franjas blancas y negras. A medida que se corrige el corrimiento f_Δ, la frecuencia de las franjas disminuye, confirmando lo visto en 6.5.2 sobre que el corrimiento de frecuencia generaba un giro en la muestras. Esto se traduce en un cambio periódico del brillo en la visualización de líneas verticales (negro - blanco).

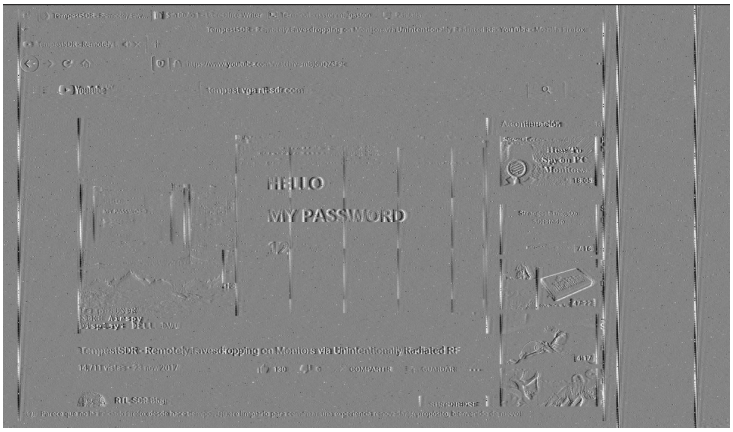
A primera vista, el contraste entre el blanco y el negro hace que la imagen reconstruida, al tomar la magnitud de la señal, pareciera de mejor calidad. Sin embargo, al observar minuciosamente, se puede ver que la nitidez de las letras y los íconos mejora al tomar la parte real de la señal corregida. A medida que se minimizan los cambios de blanco a negro en las líneas verticales con la corrección del corrimiento en frecuencia, la visualización mejora y se pueden percibir más detalles.



(a) Sin ecualizar, tomando el módulo de la señal.



(b) Sin ecualizar, tomando la parte real de la señal.



(c) Con corrección de f_Δ.

Figura 7.21: Imágenes espiadas con gr-tempest sin y con corrección de frecuencia

7.6.2. Ecualización Lineal

Se realiza el análisis cualitativo de la ecualización lineal implementada, tanto desde la mirada global como de pequeñas secciones de las imágenes espiadas. Para esto, se divide la imagen en partes, mirando en detalle el sector del video, el sector de URL y pestañas del navegador y, por último, algunos detalles como íconos y fotos pequeñas.

En la Figura 7.22 se presenta el sector de la imagen espiada correspondiente al video de la Figura 7.20. Se puede comparar la visualización cuando solo se corrige el error de frecuencia, cuando solo se implementa la ecualización lineal y la combinación de ambos.

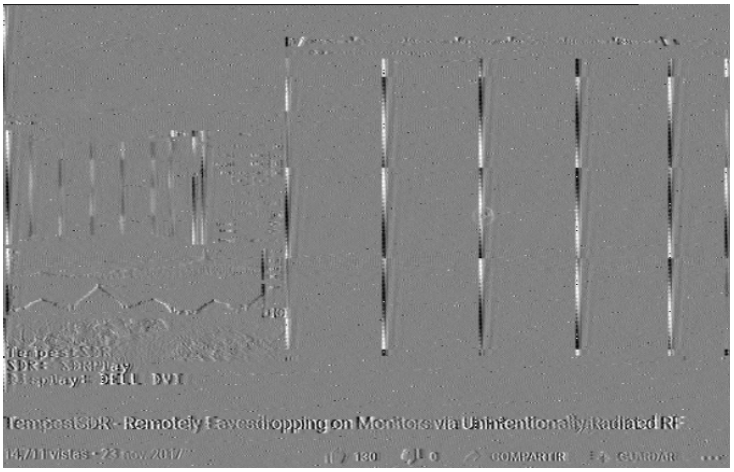
Al comparar la Figura 7.21(a) correspondiente al sistema original gr-tempest con la Figura 7.22(b) obtenida implementando el ecualizador lineal, se puede apreciar que las líneas verticales tienen mayor grosor al ecualizar. Esto se debe a haber amplificado las componentes de baja frecuencias. Ahora, al comparar la Figura 7.21(a) sin ecualizar y la 7.22(c) con ecualizador, ambas con corrección de f_Δ, se aprecia como este efecto de engrosar los bordes ayudan a la reconstrucción de la imagen espiada, tanto en la visualización de la cuadrícula como en la lectura del título del video.

Cada método por separado tiene sus ventajas. Por ejemplo, los cambios de color blanco y negro de la cuadrícula se pueden percibir cuando se realiza la corrección de corrimiento de frecuencia y no cuando solo se realiza la ecualización lineal tomando el modulo de la señal. El título del video también es preferible en ese escenario. Por otro lado, en la Figura 7.22(b) (ecualización lineal) se distinguen mejor las formas presentes en el lado izquierdo de la pantalla. Se considera que al combinar los dos métodos es cuando se consigue la mejor visualización.

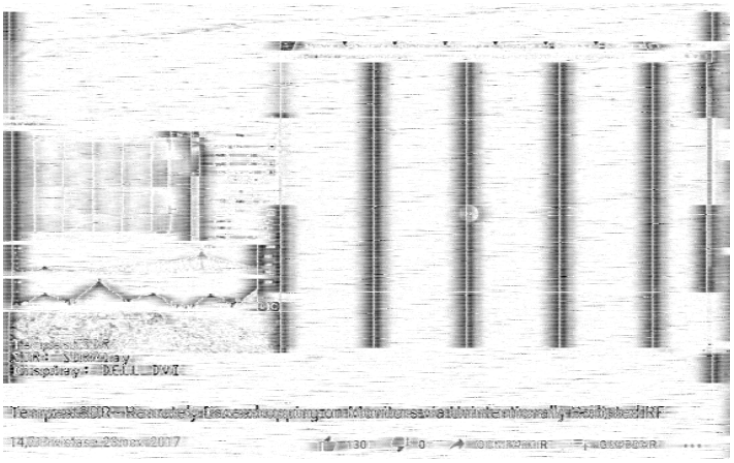
Para continuar, se realiza una ampliación al sector donde se sugieren los videos disponibles para ver a continuación. En particular, uno de ellos tiene el rostro de una mujer apenas perceptible previo a la ecualización (ver Figura 7.23). Pero, al corregir el f_Δ, los bordes definidos dan lugar a contornos y sombras. Al agregar el filtro de ecualización lineal, como se observa en 7.25, se puede ver en mayor detalle el rostro de una mujer. Esto vuelve a ser producto de una mayor sensación de sombras que se reconocen al permitir que la energía no se concentre solo en los contornos de la figura.

Por último, se evalúa la importancia del reconocimiento de texto en el marco del espionaje. En la Figura 7.26 se presenta el sector superior de la pantalla, donde se encuentra la barra del navegador y las pestañas. Tanto al realizar solo corrección de frecuencia como al incluir la ecualización lineal, se llega a distinguir la totalidad de las letras en el tramo evaluado. A su vez, es notorio que el proceso de ecualización completo mejora la visualización de los íconos si se los compara con la Figura 7.21(a) obtenida sin ningún tipo de ecualización.

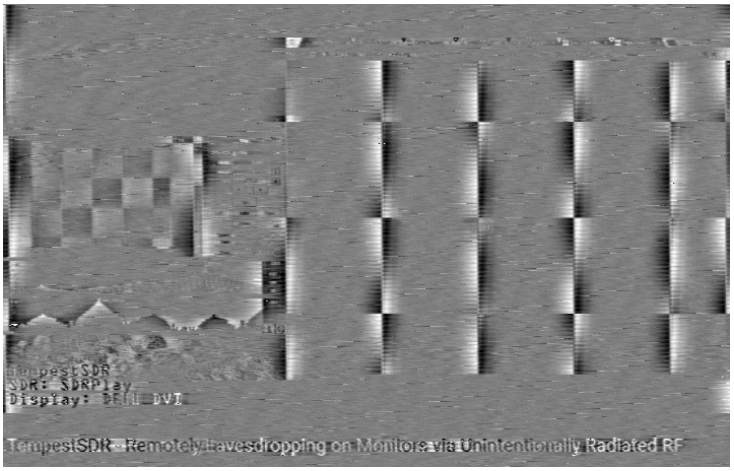
A la luz de lo visto en cada dimensión de los métodos de ecualización explorados, se concluye que el resultado de la implementación es satisfactorio. Tanto en la mejor comprensión de las imágenes como en la mayor capacidad de extraer información del texto, las técnicas basadas en la bibliografía y en las respectivas pruebas demuestran que el uso de métodos de implementación aportan claridad a



(a) Con solo corrección de f_Δ.



(b) Con solo ecualización lineal (tomando el módulo).



(c) Con ecualización lineal y corrección de f_Δ.

Figura 7.22: Sector del video en la imagen espiada

Figura 7.23: Ampliación de la imagen tomando el modulo de la señal

Figura 7.24: Ampliación de la imagen con corrección de f_Δ

Figura 7.25: Ampliación de la imagen con corrección de f_Δ y ecualización lineal

(a) Con corrección de f_Δ.

(b) Con ecualización lineal y corrección de f_Δ.

Figura 7.26: Recorte de sector superior de la imagen espiada

la imagen y mejoran su percepción desde una perspectiva cualitativa.

7.7. Sistema Completo

Tras analizar y evaluar los resultados de las implementaciones de hardware y software de manera individual o parcial, se hace esencial corroborar el funcionamiento del sistema completo operando en simultáneo, para asegurar que los procesos y los rendimientos no generen conflictos en el aplicativo final.

En esta sección se presentan los resultados obtenidos al espiar un monitor 1920x1080@60 utilizando el sistema gr-tempest 2.0 con todas sus funcionalidades en modo online (con la excepción de la inferencia de resolución, cuya instancia separada hace que el análisis de la Sección 7.2 sea suficiente). El sistema puede encontrarse en su totalidad en el repositorio utilizado para este proyecto [38].

El diagrama de bloques utilizado en la versión final se puede observar en la

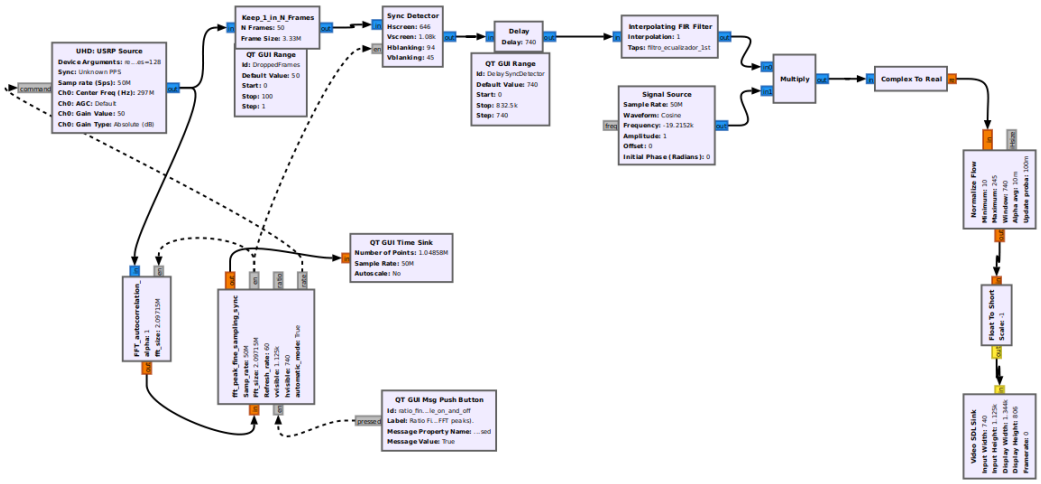


Figura 7.27: Diagrama utilizado para implementar la corrección del muestreo por hardware a 50 MSps con ecualización lineal y corrección de frecuencia central.

Figura 7.28: James Hetfield

Figura 7.29: Público de Metallica

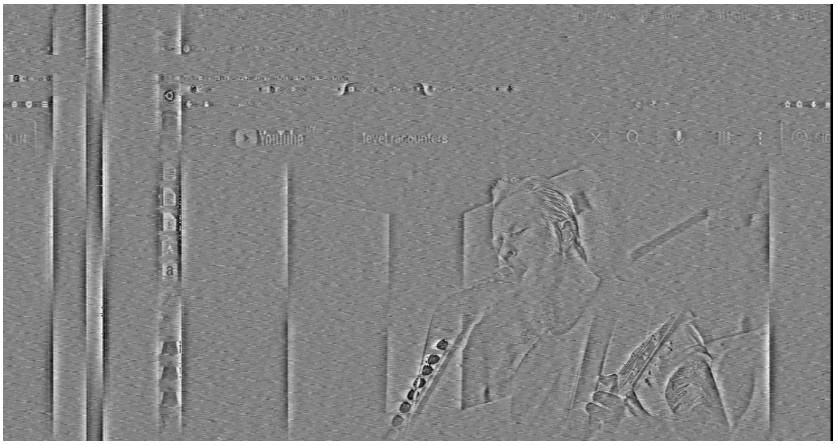
Figura 7.27. Allí se muestra la integración del sistema de cálculo para la corrección de muestreo realimentando al driver del SDR, que pasa por la etapa de descarte para luego ser centrado horizontal y verticalmente. Finalmente, los dos métodos de ecualización logran ser implementados en la modalidad online, gracias a la disponibilidad de procesamiento que genera el no requerir interpolación.

El monitor espiado muestra videos de la plataforma Youtube, cuyas capturas se presentan en las Figuras 7.28 y 7.29. Corresponden a un concierto en vivo de la banda Metallica, cuyo cambio constante de cámaras y ángulos plantea un desafío importante para el espionaje.

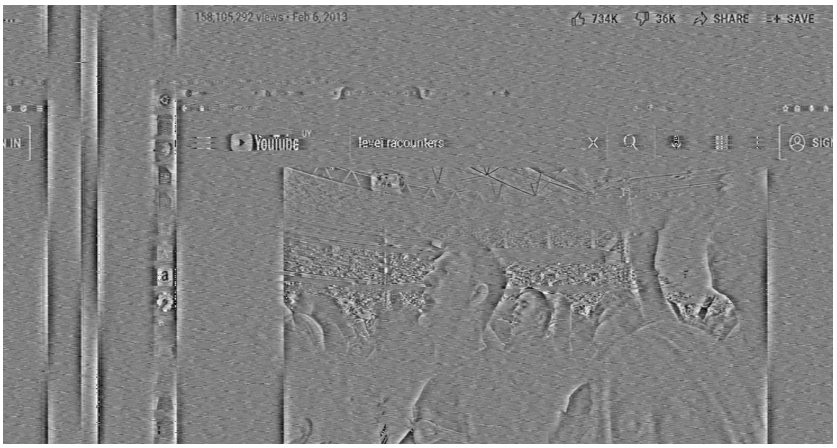
Las imágenes de la Figura 7.30 muestran los resultados en este complicado escenario, en el cual tanto letras como contornos de figuras tienen una notoria distinción que permiten identificarlas, aun sin haber visto en detalle el monitor espiado.

Surge en este punto, por el carácter específico del video en continuo movimiento y con cuadros descartados, un pequeño conflicto con el funcionamiento del bloque *Sync Detector*.

La implementación del sincronismo horizontal consiste en la búsqueda del *blanking* horizontal a través de la fórmula (4.1). Esa búsqueda se realiza con la información de un cuadro para luego corregirlo en el siguiente. Con una energía de los cuadros tan variable (por los movimientos del video) y una disminución de los cuadros por segundo que provoca saltos más bruscos entre los cuadros, se entiende



(a) Pantalla espiada con James Hetfield



(b) Pantalla espiada con público de Metallica

Figura 7.30: Captura de espionaje de monitor 1920x1080@60 online

posible que se genere una diferencia no despreciable entre el cuadro en el que se realizan los cálculos de sincronización y el cuadro donde efectivamente se aplican.

No obstante, lo mismo no ocurre en escenarios más esperables del uso de monitores. En imágenes como la espiada en la Figura 7.31 la detección de sincronización tiene un comportamiento como el evaluado previamente en este capítulo, requiriendo solo de un ajuste vertical manual para que la salida sea la deseada. Además, la figura expone la diferencia en los resultados al implementar la ecualización lineal y corrección de frecuencia central, principalmente en la visualización del texto. El video [39] puede dar evidencia de esto, mostrando cómo el contenido de una página de ingreso a datos bancarios es legible en el resultado desplegado por el sistema operando online.

Estas funcionalidades pudieron ser incorporadas a la modalidad online gracias a la liberación de recursos mencionada, resultando en lo observado en la Figura

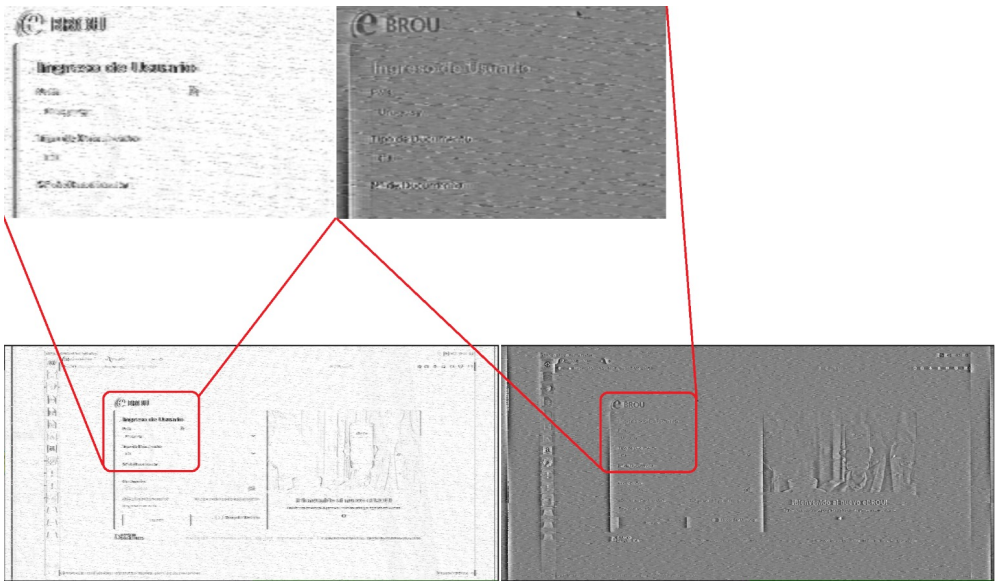


Figura 7.31: Comparación del resultado de utilizar el sistema completo cuando se toma la magnitud de la señal contra el sistema completo cuando se toma la parte real de la señal, se hace ecualización lineal y corrección de frecuencia central.

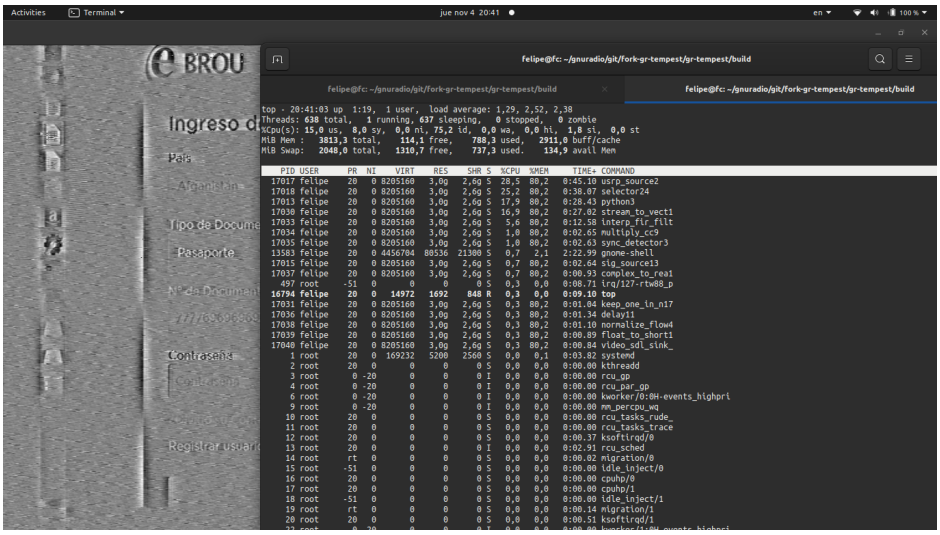


Figura 7.32: Resultado y uso del CPU en porcentajes 1080p 50 MSps N = 10.

7.32: del 15 % de uso total en una CPU de dos núcleos y cuatro *threads*, gr-tempest está tomando una parte igual al 81.1 % cuando se suman todas las distintas contribuciones de su *flowgraph*.

Esto permite cerrar el capítulo evaluando al sistema completo como efectivo en su operativa de funcionalidades conjuntas, habilitando el espionaje completo y perceptible a una tasa de 50 MSps con bajos requerimientos de hardware.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 8

Conclusiones y Trabajo a Futuro

8.1. Conclusiones

En este trabajo se abordó la problemática del espionaje por emisiones electromagnéticas desde un enfoque práctico, buscando culminar en una implementación que demuestre mejorías con respecto a su versión anterior y contribuya al avance de los estudios de TEMPEST.

Es apropiado comenzar el capítulo final afirmando que los objetivos del proyecto fueron alcanzados satisfactoriamente. En términos generales, se pudieron implementar las modificaciones al software gr-tempest de forma que, en base al análisis realizado de la versión original y al desarrollo teórico asociado, se agregaran o corrigieran las funcionalidades determinadas como oportunidades de mejora.

Además, se implementó un sistema de hardware que posibilitó la recepción de las emanaciones electromagnéticas de la imagen de video para una resolución de 1920x1080@60Hz. Este sirvió de base para el desarrollo del software y de método de prueba para la operativa y el rendimiento online del sistema completo. También dio pie al análisis en mayor profundidad de la señal de radiofrecuencia y, en particular, a su interacción con el componente clave de cualquier sistema sobre el que gr-tempest está diseñado para operar: SDR. Se destacaron en la teoría y en la práctica los beneficios de la radio definida por software, pudiendo comprender su funcionamiento y confirmar su relevancia.

En lo particular, se consiguió incorporar una funcionalidad nueva, a ser ejecutada con anterioridad al espionaje de las emanaciones, a través de la cual se obtienen todos los valores relevantes de la resolución del monitor espiado. Esta función aporta generalidad al aplicativo, permitiéndole espiar aunque no se conozca la información de resolución.

Se obtuvo también una función de detección de sincronización que, basándose en el algoritmo de TempestSDR con los ajustes correspondientes a la modalidad de procesamiento de GNU Radio, logra centrar la imagen horizontalmente y fijarla verticalmente. De esta manera, al acompañarse de un retardo en múltiplos del largo de la línea, la imagen logra ser completamente centrada.

El bloque a cargo del cálculo de la corrección del muestreo de la señal fue

sustituido, pasando a un método de autocorrelación que logra la convergencia del índice de interpolación sin requerir la ejecución de *callbacks* manuales. El resultado puede ser utilizado tanto para indicarle a un interpolador en software cómo corregir el muestreo como para realimentar al SDR y que ninguna interpolación sea necesaria.

Esto último, a su vez, permitió mejorar el rendimiento de un sistema que ya no requiere interpolar en espionaje online, sino que logra sacar provecho de la capacidad del SDR de modificar sus parámetros en tiempo de ejecución y se ajusta en hardware, reduciendo significativamente el consumo de CPU del sistema y permitiéndole alcanzar frecuencias de muestreo tan altas como 50 MSps.

Una alternativa en software fue diseñada para la reducción del procesamiento, descartando cuadros completos para que el interpolador opere con menos muestras. Incluso perdiendo algunas muestras no deseadas en el cálculo del cuadro, la posibilidad de reducir el procesamiento sin depender de la compatibilidad del SDR usado se considera valioso para la implementación. Más aun dado que el desperfecto no impide al usuario identificar y percibir la información de la imagen.

Finalmente, se exploraron dos técnicas de ecualización que, operando tanto por separado como en simultáneo, resultan en una notable mejora en la percepción de la imagen de video. Los efectos de la ecualización lineal y de la corrección del corrimiento en frecuencia generan figuras más distinguibles y textos más legibles, mejorando la totalidad de la sensación visual y la información extraíble.

Este conjunto de elementos, operando online y en simultáneo, reafirman la relevancia del paradigma SDR en la aplicación y confirman el avance de gr-tempest en su versión 2.0 hacia un software más optimizado, estable y robusto. Además, las nuevas funcionalidades aportan a convertirlo en una herramienta más completa y más orientada al espionaje real de una señal no conocida.

De la mano de esto, se enfatiza la finalidad principal del abordaje de TEMPEST como línea de investigación: exponer la brecha de seguridad fundamental existente en las comunicaciones de despliegue de imágenes, ampliamente utilizadas y para las cuales no se toman los recaudos suficientes.

8.2. Trabajo a Futuro

Por las limitaciones de alcance y de tiempo que impone el marco del proyecto, se abren formas de continuar con el trabajo realizado que quedan pendientes para próximos desarrollos. Como ejemplo de algunas de menor alcance, a operar como siguientes mejoras en gr-tempest, se listan:

- Trabajar con vectores en vez de un *stream* de datos en algunos bloques. Por ejemplo, en el *Sync Detector* se podría recibir un cuadro entero, procesarlo y entregarlo alineado al centro de la pantalla. Esto podría mejorar el centrado de la pantalla cuando se descartan muchos FPS.

- Utilizar el método del *Frame Dropper* como bloque de descarte que se sincroniza con el *Fine Sampling Synchronization* y no quiebra la pantalla. De

esta forma se realizaría corrección de muestreo completamente por software para eliminar la dependencia de la utilización de hardware SDR soportado por drivers de GNU Radio.

- Optimizar el cálculo de la autocorrelación por FFT utilizando correlación cruzada con desplazamiento para que se vea solamente una pequeña ventana alrededor del primer pico del cuadro. De esta forma se podría evitar el uso de una ventana tan grande como un cuadro entero. Con la metodología implementada en esta versión, la correlación cruzada no funciona correctamente ya que no tiene el pico tan marcado.

También se contemplan posibilidades, quizás más lejanas, en el procesamiento a llevar a cabo con la imagen espiada y sus limitaciones en la recepción.

Una posibilidad consistiría en utilizar algoritmos de *denoising* más sofisticados que la simple ecualización lineal. En particular, investigar algoritmos basados en aprendizaje a partir de datos, ya sea utilizando modelos de aprendizaje profundo o modelos clásicos simples y efectivos que puedan correr online.

Con el trabajo presente se puede generar una base de datos, con imágenes limpias a partir del monitor espiado con sus respectivas parejas que son las imágenes levantadas en la PC espía. Durante el proceso de este proyecto se encontraron diversos repositorios y publicaciones enfocados en alimentar algoritmos de Machine Learning [40] que aprendan las correspondencias entre un dato ruidoso y un dato limpio.

No solo podría mejorarse la nitidez de la imagen reconstruida, sino que también se podría inferir el color de las distintas secciones de la imagen espiada, como se hace con las técnicas de *inpainting*. Esta es una característica de la señal emitida por el cable que es casi imposible de recuperar sin este tipo de técnicas.

Finalmente, en términos de exposición de las vulnerabilidades de los datos, se podría utilizar la misma base de datos para detectar textos y procesar palabras claves o para etiquetado de información sensible o detección de anomalías.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] Eutic 2019 encuesta de usos de tecnologías de la información y la comunicación. Technical report, Uruguay Presidencia, Instituto Nacional de Estadistica y AGESIC, 2019.

[2] Estudio de conocimientos, actitudes y prácticas de ciudadanía digital. principales resultados 2021. Technical report, Uruguay Presidencia y AGESIC, Junio 2021.

[3] Ecosistema de ciberseguridad en uruguay. un análisis cualitativo. Technical report, Uruguay Presidencia, agesic, KPMG, Junio 2020.

[4] Tempest: A signal problem. Technical report, NSA FOIA Case 51633. Approved for Release by NSA, EEUU, 2007.

[5] W van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? Technical report, Elsevier Science Publishers B.V, North-Holland, 1985. 0167-4048/85.

[6] Markus G Kuhn. Compromising emanations: eavesdropping risks of computer displays. Technical report, University of Cambridge Computer Laboratory, 2003. UCAM-CL-TR-577.

[7] Furkan Elibol, Ugur Sarac, and Isin Erer. Realistic eavesdropping attacks on computer displays with low-cost and mobile receiver system. In *In Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, page 1767–1771. IEEE, 2012.

[8] Martin Marinov. Remote video eavesdropping using a software-defined radio platform. Tesis de maestría, University of Cambridge, Cambridge CB3 0FD, Reino Unido, 11 de Junio 2014.

[9] Pablo Menoni. Espionaje por emisiones electromagnéticas. Tesis de maestría, Universidad de la República Facultad de Ingenieria, Montevideo, Uruguay, 12 de Diciembre 2018.

[10] Federico 'Larroca' La Rocca. Github gr-tempest. Repositorio. `https://github.com/git-artes/gr-tempest`.

[11] John R. Reitz, Frederick J. Midford, and Rober W. Christy. *Fundamentos de la teoría electromagnética*, chapter 16 - Ecuaciones de Maxwell. Addison-Wesley Iberoamericana, cuarta edition.

[12] Asim Kadav and Michael M. Swift. Understanding modern device drivers. Technical report, Computer Sciences Department, University of Wisconsin-Madison, 2012.

[13] Alexandre Rouma. Sdrplusplus. Repositorio. `https://github.com/AlexandreRouma/SDRPlusPlus`.

[14] Paolo Romani. *SDRsharp The Guide*. AIRSPY, 3.0 edition, Agosto 2021.

[15] Víctor González-Barbone and Federico 'Larroca' La Rocca. Gnu radio. desarrollo de módulos "out-of-tree". `https://docs.google.com/document/d/1uCrgirhM6piOSQkI8XvEOKZz_86FuYNSTJN355a1acQ/edit`.

[16] Colan Biemer. C++ to python: Bindings with swig. Repositorio. `https://gist.github.com/bi3mer/238b3c890decc5fdbc6e99025a6db66d`.

[17] VESA. *VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT)*. Video Electronics Standards Association, 39899 Balentine Drive, Suite 125. Newark, CA 94560, 1.0 rev. 13 edition, 8 de Febrero 2013. `https://glenwing.github.io/docs/VESA-DMT-1.13.pdf`.

[18] Michael Robin and Michel Poulin. *Digital Television Fundamentals*. Mcgraw-Hill, second edition, 2000.

[19] Keysight Technologies. Fundamentals of rf and microwave noise figure measurements, September 2017. `https://eva.fing.edu.uy/pluginfile.php/211845/mod_resource/content/1/5952-8255E.pdf`.

[20] Broadcast band., Julio 2021. `https://en.wikipedia.org/wiki/Broadcast_band`.

[21] Ettus Research. Usrp™ b200mini series product overview. `https://www.ettus.com/wp-content/uploads/2019/01/USRP_B200mini_Data_Sheet.pdf`.

[22] Xilinx. Fpga spartan-6 family overview. `https://www.xilinx.com/support/documentation/data_sheets/ds160.pdf`.

[23] Analog Devices. Rf agile transceiver ad9364 technical documentation. `https://www.analog.com/media/en/technical-documentation/data-sheets/AD9364.pdf`.

[24] Ettus Research. Usrp™ b200mini series schematics. `https://files.ettus.com/schematics/b200mini/b200mini.pdf`.

[25] Sitio web de mini - circuit. `https://www.minicircuits.com/`.

[26] Mini Circuits. Zx60-p103ln. https://www.minicircuits.com/WebStore/dashboard.html?model=ZX60-P103LN%2B.

[27] https://www.minicircuits.com/pdfs/SHP-250+.pdf.

[28] https://www.minicircuits.com/WebStore/dashboard.html?model=SLP-450%2B.

[29] Wikipedia enciclopedia libre. Wiener–khinchin theorem. https://en.wikipedia.org/wiki/Wiener%E2%80%93Khinchin_theorem.

[30] Wikipedia Enciclopedia Libre. Welch's method. https://en.wikipedia.org/wiki/Welch%27s_method.

[31] Volk. Doxygen documentation. https://www.libvolk.org/doxygen/index.html.

[32] ghostop14. Github: gr-guiextra. Repositorio. https://github.com/ghostop14/gr-guiextra.

[33] Usrp source, Junio 2020. https://wiki.gnuradio.org/index.php/USRP_Source.

[34] gr-tempest 2.0: desempeño de detección de sincronización, Noviembre 2021. https://www.youtube.com/watch?v=JUZbaFCLf40.

[35] gr-tempest 2.0: desempeño de corrección de muestreo, Noviembre 2021. https://www.youtube.com/watch?v=bjwde83XcCQ.

[36] Ettus Research. Usrp hardware driver and usrp manual: uhd::transport namespace reference. https://files.ettus.com/manual/namespaceuhd_1_1transport.html.

[37] Ettus Research. Usrp hardware driver and usrp manual: Transport notes. https://files.ettus.com/manual/page_transport.html.

[38] gr-tempest 2.0. Repositorio github: gr-tempest-2.0. fork de git-artes/gr-tempest. Repositorio. https://github.com/fcarraustewart/gr-tempest.

[39] gr-tempest 2.0: desempeño de la ecualización en el sistema completo online, Noviembre 2021. https://youtu.be/dM9IElFhXdw.

[40] Florian Lemarchand et al. Electro-magnetic side-channel attack through learned denoising and classification. Technical report, ICASSP, 2020. https://github.com/opendenoising.

Esta página ha sido intencionalmente dejada en blanco.

Índice de tablas

7.1. Procesamiento requerido en porcentaje de CPU para los bloques Fine Sampling Synchronization y Frame Dropper, con distintos descartes de muestras. 74

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

1.1. Diagrama esquemático del escenario de espionaje, incluyendo tanto al sistema espiado como al espía. 4

2.1. Diagrama de bloques del SDR: en amarillo lo perteneciente al hardware; en azul el software y los parámetros que por él se modifican. 8
2.2. Ejemplo de diagrama de GNU Radio Companion a partir del cual se describe un archivo .grc para la comparación de pulsos conformadores. 10

3.1. Definición de parámetros de trama de video 14
3.2. Definición de los parámetros temporales para la resolución 1920x1080@60. Tomada de la norma VESA Display Monitor Timing Standard Copyright 1994-2013 Video Electronics Standards Association. 16
3.3. Bosquejo del espectro de la señal de video $X(f)$ junto con $X_s(f)$ y el pulso conformador. La componente banda base de $X(f)$ no se representa porque no se propaga. 18

4.1. Procesamiento usando el sistema con GUI de Java. El fondo blanco representa código en C y fondo negro en Java, mientras que en gris están los procesos de hardware externo. Tomado de [8] 22
4.2. GUI y salida de TempestSDR. 24
4.3. Definición de c, ω y n, en este caso con la relación que maximiza β. Tomado de [8]. 25
4.4. Para los c dados para los arreglos horizontal y vertical, se alinea el cuadro para centrar el video. Tomado de [8] 25
4.5. Diagrama de recepción de gr-tempest original con corrección de muestreo automático. 27
4.6. Salida de video con la versión original de gr-tempest. 28

5.1. Diagrama de bloques del Hardware planificado. 34
5.2. Bandas del espectro radioeléctrico, cercanas a las frecuencias de interés. 34
5.3. El SDR de Ettus-Research USRP B200mini. 35
5.4. Simulación software QUCS LNA-LPF-HPF 36
5.5. Simulación software QUCS SWR a la entrada LNA-LPF-HPF . . . 37
5.6. Simulación software QUCS SWR a la salida LNA-LPF-HPF 37

5.7. Configuración de hardware: Antena, LNA ZX60-P103LN+ (con Fuente DC 5V), HPF SHP-250+, LPF SLP-450+ y USRP B200mini . . 38

6.1. Flowgraph para el cálculo de la autocorrelacion de muestras 40
6.2. Diagrama de bloques para la inferencia de resolución 42
6.3. Diagrama del flujo de datos del bloque de sincronización. 43
6.4. Diagrama del sistema original, implementando el bloque Sync Detector en lugar de Hsync. 47
6.5. Salida regular del sistema simulado usando el bloque Fine Sampling Synchronization. 47
6.6. Salida en terminal del driver de SDR notificando de las muestras perdidas. 49
6.7. Ejemplo de señal submuestrada cuya imagen es difícil de reconstruir. 50
6.8. Diagrama del conexionado para la implementación del sistema con realimentación SDR. 52
6.9. Diagrama del sistema, implementando Frame Dropper en las muestras recibidas del SDR. 53
6.10. Bosquejo del espectro del filtro ecualización junto con el espectro de $G(f)$. 56
6.11. Flowgraph de prueba de filtro de ecualización lineal. 56
6.12. Diagrama vectorial de dos muestras de píxeles adyacentes verticalmente. 57
6.13. Flowgraph con sistema de ecualización. 58
6.14. GUI para la implementación de corrección de f_Δ. 59
6.15. Imagen espiada con un error en frecuencia central $f_\Delta \approx 0.06$ kHz. 59
6.16. Imagen espiada con un error en frecuencia central $f_\Delta \approx 1.02$ kHz. 59
6.17. Imagen espiada con un error en frecuencia central $f_\Delta \approx 18.93$ kHz. 59

7.1. Salida del ROHDE & SCHWARZ ZVB 8 - VNA para el circuito probado. 62
7.2. ROHDE & SCHWARZ ZVB 8 - VNA con el conjunto LPF-HPF en el ambiente de laboratorio. 62
7.3. Flowgraph en GNU Radio Companion para prueba de hardware. . 63
7.4. Respuesta en frecuencia dentro de GNU Radio con centro en 300 MHz, muestreando 50 MSps Generador Agilent LNA-LPF-HPF B200mini 64
7.5. Prueba de Ganancia de 22dB relativos en Gnuradio QT GUI Freq Sink. Agregando el LNA y removiéndolo para apreciar los cambios. 64
7.6. Autocorrelacón de las muestras provenientes del HW 65
7.7. Grabación de resolución 1024x768 con despliegue centrado por Sync Detector. 67
7.8. Grabación de resolución 1920x1080, ruidosa, con despliegue centrado por Sync Detector. 67
7.9. Parte del diagrama utilizado para simular la recepción de una señal. 68
7.10. Imagen simulada con despliegue centrado por Sync Detector. . . . 68
7.11. Escritorio simulado con despliegue centrado por Sync Detector. . . 69
7.12. Retraso añadido al Sync Detector para el ajuste vertical manual. . 69

7.13. Indicador de consumo de Linux para distintos procesos operando en el escenario simulado con el sistema original y el bloque Sync Detector. 70
7.14. Parte del diagrama de flujo que implementa el FFT peaks comandado por botón, pasándole el índice al Fine Sampling Synchronization para que interpole. 72
7.15. Parte del diagrama utilizado para implementar el Frame Dropper en el sistema simulado. 74
7.16. Simulación procesada por sistema con Frame Dropper. 75
7.17. Grabación procesada por sistema con Frame Dropper. 76
7.18. Diagrama utilizado para implementar la corrección del muestreo por hardware a 50 MSps. 78
7.19. Comparación de salida durante el cálculo de la corrección de muestreo 79
7.20. Captura real del vídeo en el monitor espiado 80
7.21. Imágenes espiadas con gr-tempest sin y con corrección de frecuencia 81
7.22. Sector del video en la imagen espiada 83
7.23. Ampliación de la imagen tomando el modulo de la señal 84
7.24. Ampliación de la imagen con corrección de f_Δ 84
7.25. Ampliación de la imagen con corrección de f_Δ y ecualización lineal 84
7.26. Recorte de sector superior de la imagen espiada 84
7.27. Diagrama utilizado para implementar la corrección del muestreo por hardware a 50 MSps con ecualización lineal y corrección de frecuencia central. 85
7.28. James Hetfield 85
7.29. Público de Metallica 85
7.30. Captura de espionaje de monitor 1920x1080@60 online 86
7.31. Comparación del resultado de utilizar el sistema completo cuando se toma la magnitud de la señal contra el sistema completo cuando se toma la parte real de la señal, se hace ecualización lineal y corrección de frecuencia central. 87
7.32. Resultado y uso del CPU en porcentajes 1080p 50 MSps N = 10. . 87

Esta es la última página.
Compilado el martes 16 noviembre, 2021.
http://iie.fing.edu.uy/

gr-tempest 2.0: Improvements to the implementation of TEMPEST in GNU Radio

Pablo Bertrand, Felipe Carrau, Victoria Severi

***Abstract*—the first release of gr-tempest managed to achieve a working version of Martin Marinov's TempestSDR using GNU Radio as framework, in order to facilitate the maintenance and extension of the code. However, it struggled to perform some functionalities and lacked others, which were used to establish the objectives of this work: to design and implement the changes required for those functionalities to be complete. The result was a monitor eavesdropping application with horizontal and vertical synchronization, monitor resolution inference, equalization, improved sampling correction and better overall performance.**

***Index Terms*—TEMPEST, eavesdropping, emanations, SDR, GNU Radio.**

I. Introduction

TEMPEST was the name given by the United States' National Security Agency to the investigations related to involuntary electromagnetic emanations that could be compromising in terms of security [1]. These were classified for decades and then partially released, revealing how Bell Lab's researchers found that information going through cables could be extracted by receiving their emanations.

The first unclassified technical report regarding espionage by electromagnetic eavesdropping was published by Wim van Eck [2] in 1985 and it focused on monitor cables and the espionage of the video that they carry. Later, on 2003, Markus Kuhn [3] publishes another significant report, incorporating FPGA boards and AM receptors to the reception system, obtaining better results but still requiring high cost equipment with specific -and fixed- characteristics.

It was in 2014 when Martin Marinov released his project, TempestSDR [4], as part of his master's thesis. In it, he developed a library based in C language and a host in Java that are meant to spy VGA video while interacting with an SDR-based [5] front end system. This allowed the otherwise limited reception ability to adapt and change according to the spied monitor's parameters. The now general purpose hardware made for a much cheaper and portable system allowed the application to be used in simpler scenarios with off-the-shelf affordable components.

Pablo Menoni [6] was the first to take on TEMPEST at Universidad de la República, Uruguay, for his master's degree thesis. After presenting a very detailed analysis of the VGA signal and how it is received by a remote spy, he extended the studies to the HDMI protocol and provided some equalization and image improvement theoretical results.

During the SARS-Cov-2 pandemic, Federico La Rocca, Menoni's tutor, realized the limitations of TempestSDR given by its self contained nature: it is harder to maintain and extend. Thus, he designed and developed gr-tempest inspired in TempestSDR, with some of the same algorithms and some of his own, but through the framework GNU Radio which will be introduced in the next section.

A first functioning version of gr-tempest was released in 2020 [7], being able to effectively eavesdrop monitors in real time. However, some of the functionalities weren't available and others were not optimized, leaving room for improvement.

That is the context for this project to take place, making its main objective to get gr-tempest's behaviour closer to TempestSDR's. As a secondary objective, the design of a radio frequency circuit (RF front end) is taken on, in order to receive the signal in real-time and test the software for a specific resolution: 1080p.

The SDR-based hardware and the mentioned software operate in a scenario as shown in Figure 1: the CPU sends video signals to the monitor through VGA or HDMI cables, which emanate electromagnetic fields as a consequence of voltages and currents varying.

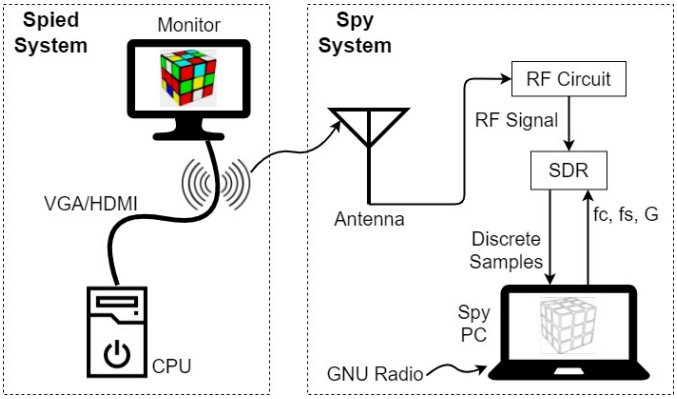


Figure 1. Espionage Scenario with gr-tempest.

These fields propagate through the air and are received by a distant antenna, connected to a RF circuit meant to do signal conditioning before going through the SDR for modulation, filtering and sampling. The SDR is connected to a PC that runs GNU Radio, which receives the complex samples and is able to return commands to the SDR that modifies its parameters during run-time, via gr-tempest.

The software is therefore in charge of interacting with the SDR to update sample frequency, center frequency and gain, and to use the samples of the signal to correct sampling, center the video and display it on the spy monitor.

The result of this work includes the appropriate hardware for spying 1080p resolution monitors and the GNU Radio module with algorithms for vertical and horizontal image centering, sample correction, resolution inference -required as a previous phase to later do the eavesdropping itself- and equalization.

II. SDR: Software Defined Radio

One of the keys to the added value of the recent implementations of TEMPEST is given by the use of Software Defined Radio or SDR as a paradigm for the acquisition and processing of signals. SDR implies the translation of several radio functionalities to software-based solutions, making them more versatile to modify and adapt.

Traditional solutions consist on dedicated hardware that are limited to a specific application because of fixed frequency parameters that determine a certain behaviour during run time. SDR allows for those parameters to change, adapting the carrier frequency and sampling rate to make for a general purpose component.

The diagram in Figure 2 represents the SDR reception components: after low noise amplification, the signal is demodulated in phase and quadrature and then sampled. Through software, the user is able to change the gain, the carrier frequency for demodulation and the sampling rate [8].

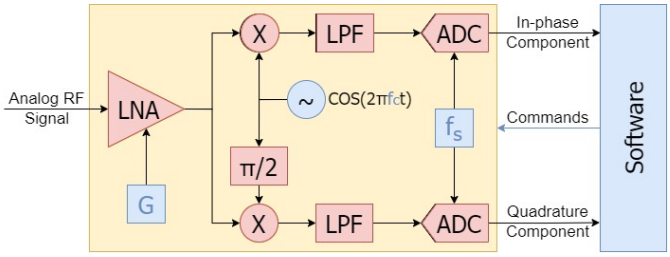


Figure 2. Block diagram of SDR components: hardware in yellow; software in blue, as well as parameters modifiable through software.

A. GNU Radio

To enable the hardware functionality, a counterpart must take place in the form of software to process the signal and retrieve the commands for the SDR to operate, as well as interfacing with the serial communications between the PC's operating system and the SDR's firmware.

Drivers interact with the inputs and outputs to translate high-level instructions to low-level and vice versa. Specific libraries are employed by the OS's kernel to carry out tasks such as memory usage and synchronization through multiple threads [9].

GNU Radio [10] is a framework that allows for these interactions to happen while also implementing a wide variety of signal processing possibilities. It contains the GNU Radio Companion executable, a software application with a graphical interface that allows to connect different processing blocks in a user-friendly manner.

GNU Radio runs a software architecture based on a scheduler - an entity in charge of managing the CPU at an application level, to order, prioritize and block different user-assigned tasks, using multithreading or multitasking. This way, the blocks are displayed in a flow form so the samples that run through them are continuously processed according to each separate logic.

Additionally, GNU Radio includes a command line interface called *gr_modtool* that allows the user to generate, modify and install GNU Radio Companion blocks to perform new functionalities. This tool enables the development of the modules required to take on the TEMPEST implementation challenge.

III. Video Signal Characterization

The understanding of the video stream is necessary to properly describe the electrical signal that is transmitted to generate it. The basic area unit used is the pixel, and it extends with a certain height and width that determine the frame's resolutions. Frames are to be refreshed with a frequency high enough for the viewer to perceive a sequence of still frames as movement. This frequency is the frame rate or refresh rate (f_V), and its most common value in monitors is 60 Hz.

This behaviour requires the processor in charge of generating the pixels to transmit them sequentially at a fast enough rate, which is done from the top left corner of the picture through a full horizontal line, and then back to the first pixel at the left of the next horizontal line. Also, as an inherited feature from analog television, the picture has blank spaces used for the image synchronization. These are called blankings, they have null energy and they surround all four sides of the picture as seen in Figure 3.

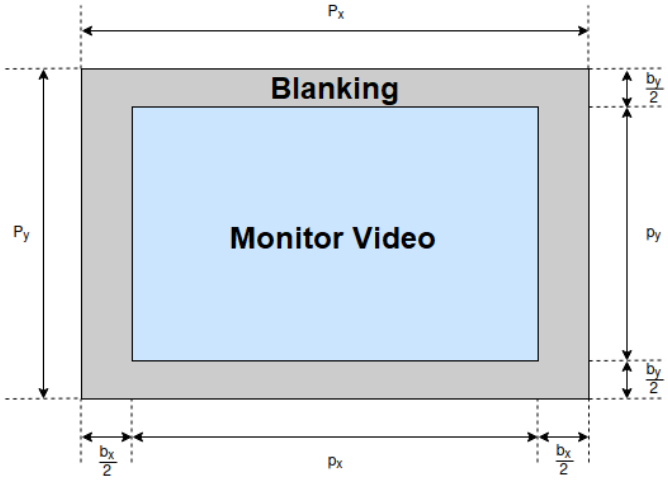


Figure 3. Video parameters.

The Figure also introduces the parameters used for the involved magnitudes. Since a full frame is formed by $P_x \times P_y$ pixels and are transmitted at a refresh rate of f_v, the duration of each pixel is given by the expression:

$$T_p = \frac{1}{P_x P_y f_v} \tag{1}$$

The stream parameters are standardized by the VESA norm [11] to guarantee the compatibility between systems and monitors. The resolutions are noted as $p_x \times p_y @ f_v$, so they are defined by the vertical and horizontal video size at a certain

refresh rate. An example of resolution relevant to this work is $1920 \times 1080@60$ Hz. The rest of the parameters, such as blanking size, are obtainable from the standard once the resolution is known.

In addition to the video stream information, the protocol used by the corresponding interface is of essence to define the electromagnetic emanations that will take place. The VGA video signal uses three PAM signals with unipolar NRZ pulses for each primary color of light: red, green and blue. The intensity of the color is described by the value of the voltage in each signal, which reside in the range between 0 and 0.7 V. The signal can be represented as:

$$x(t) = \sum_{i=-\infty}^{\infty} x_i p(t - iT_p)$$

where x_i is the intensity of a certain color in pixel i and $p(t)$ the pulse with width T_p, the duration of a pixel.

It is the variation of the currents in the cable that generate the electromagnetic emanations which are eavesdropped by gr-tempest. The NRZ pulse has the peculiarity of assigning constant voltages to constant pixel intensities. This implies that the received signal will be able to perceive the borders of the spied video image but not the filling of the shapes, leaving those pixels with no energy. The results in the next sections will demonstrate this observation.

The spectral representation of the VGA signal allows to analyze the logic for the reception via SDR techniques. The Fourier transform of the VGA signal is useful, where $X(f) = \mathcal{F}\{x(t)\}(f)$ and $P(f) = \mathcal{F}\{p(t)\}(f)$:

$$X(f) = P(f) \sum_{i=-\infty}^{\infty} x_i e^{-j2\pi i f T_p}$$

Using the Discrete Time Fourier Transform (DTFT) and evaluating $\Omega = 2\pi f T_p$ the following expression can be reached:

$$X(f) = P(f) X_s(e^{j2\pi f T_p})$$

Using $X_s(f)$ instead of $X_s(e^{j2\pi f T_p})$ as a practicality, the same expression can be written as $X(f) = P(f)X_s(f)$. Since $p(t)$ is a rectangular pulse, it is known that $P(f) = sinc(fT_p)$. Therefore, the product of the signals can be represented as in Figure 4.

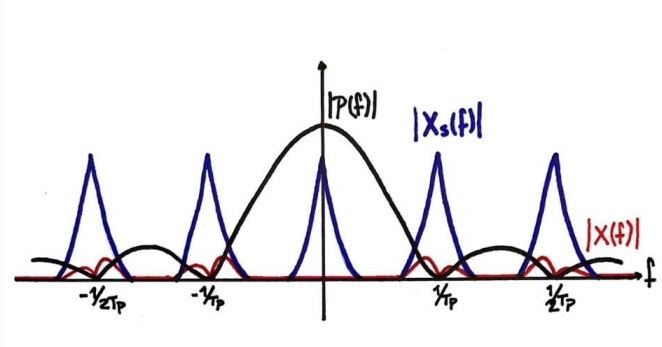


Figure 4. Sketch of the video signal $X(f)$ with $X_s(f)$ and the pulse $P(f)$.

The spectrum of $X_s(f)$ is periodical with period $\frac{1}{T_p}$ and its information is centered in the lower frequencies. The baseband component is not represented since it is not transmitted. Observing the spectrum, it can be concluded that the oscillator of the SDR should be set in some multiple of $\frac{1}{T_p}$.

Since the sinc function is nulled in the multiples of $\frac{1}{T_p}$, and the harmonics of $X_s(f)$ are centered precisely in those points, the product will not contain information of the continuous component of the signal, which is consistent with the border detection effect described previously.

IV. Technical Background

Although the context that enables the development of this work is given by decades of increasingly complex eavesdropping systems, the actual references used can be reduced to two main projects: Martin Marinov's TempestSDR and Federico La Rocca's original gr-tempest.

A. TempestSDR

TempestSDR was the first implementation of electromagnetic eavesdropping using SDR together with the concepts viewed in previous sections. It's main library was designed in C and it is executed through a Java host that includes the graphic user interface (GUI) from which an user can input and modify parameters and observe the resulting espionage.

The project is self-contained, which makes it not only easier for the user to execute, but also lighter. This, however, comes with two associated issues: the maintenance and extension of the code. The software is developed from scratch, including libraries and plug-ins for interacting with hardware, so every update must be added specifically.

Once the host runs the library, the work flow of the system is the following: the RF signal is transmitted from the hardware to the OS driver and is then received by the plugin input. The library takes care of demodulation, re-sampling and post-processing, which includes several functions that will be useful for gr-tempest. Finally, the java host receives the processed signal to manage its output. An example of the GUI is shown in Figure 5.

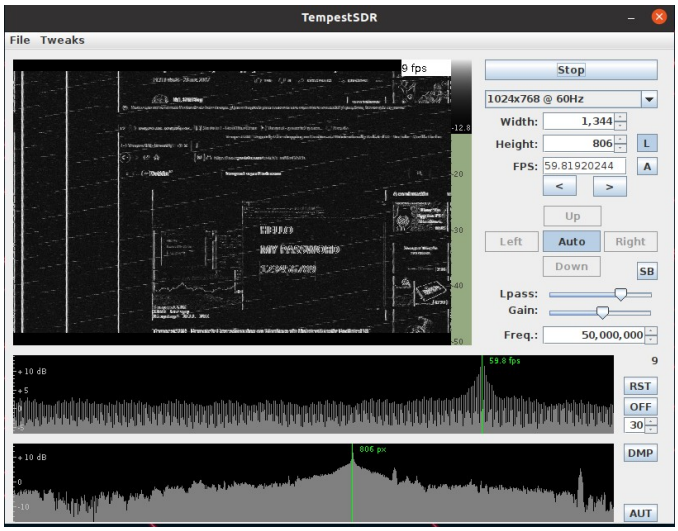


Figure 5. GUI and output example of TempestSDR.

There are two particular functions that were of interest for the design of improvements on gr-tempest: synchronization detection, meaning the centering of a sample-corrected video, and the resolution inference, as a separate solution for the user to know the parameters of the target monitor.

The image centering is necessary since the beginning of the pixel stream is entirely random, so the first pixel displayed is hardly the same as the first picture in a frame. Therefore, a functionality is designed to find the limits between the video and the blankings and use them to center the image. The way to calculate the location of the blankings is through a search for the strip of samples with the lowest energy in contrast with the rest of its line - given that the size of the blanking is known for the spied monitor's resolution, and knowing that the pixel value of every sample in it is zero, Marinov's algorithm sets a window the size of the blanking and runs it through a line in order to compare the energy, keeping the location were its contrast with the rest of the line was bigger.

This algorithm can be run both horizontally and vertically, finding the location of both blankings and obtaining the required delay that should be set at the start of the stream output for the video to be centered in the user's view, as seen in Figure 6

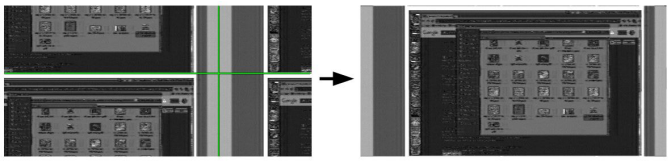


Figure 6. Image centering in TempestSDR. Taken from REF.

The second function is the resolution detection or inference. As mentioned, several parameters of the target monitor are required to eavesdrop. An example is image centering: both the screen size and blanking size are mandatory to run the window through the line and find the blanking location.

In order to obtain this information, TempestSDR uses one of the received signal's characteristics: it's discrete autocorrelation.

$$R_{vv}[j] = \sum_{i=-\infty}^{\infty} v_i \bar{v}_{i-j}$$

where v represents the discrete samples and $\bar{v}$ their conjugate. The autocorrelation is a measure of likelihood of a signal with a retarded version of itself, so the value of R will be greater in for the j distances where they are more alike.

Since there is very little movement between frames in a video, the autocorrelation will peak when comparing the signal with a conjugate retarded by exactly the time of a full frame, which allows for a quick calculation to return the monitor's frame rate:

$$f_v = \frac{1}{t_{peak}}$$

To estimate the amount of lines of the still unknown target, a similar logic takes place: with the premise that consecutive horizontal lines will not vary greatly, smaller peaks are to be found in the autocorrelation whenever the signal is retarded by the time of a full line, allowing for the vertical size to be obtained from:

$$P_y = \frac{1}{\Delta t f_v}$$

The now known frame rate and vertical size are enough to fully describe the monitor's resolution by running a simple search in the VESA table, which will return the horizontal size of the monitor together with the blanking and video sizes in both directions.

B. *gr-tempest*

As introduced before, an opportunity was detected to introduce tempest to a framework-based software to facilitate the maintenance and extension of code. This was carried out in gr-tempest, implementing a eavesdropping system that allows to successfully spy on a monitor but lacks some functionalities and has room for improvement.

The GNU Radio Companion (GRC) allows for a clear flow view of the data with a user-friendly GUI. In the case of gr-tempest, a notion of the complete operative is granted by the block sequence shown in Figure 7.

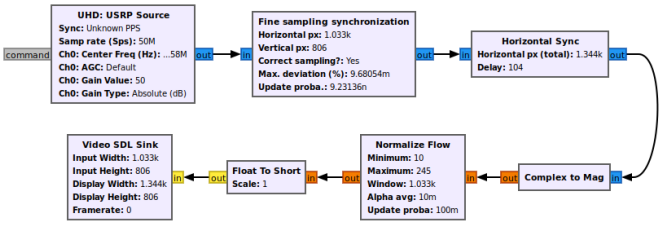


Figure 7. Original gr-tempest flow graph.

The data input is enabled through a Source block that handles the SDR's drivers and plugins, which then sends the complex samples to the Fine Sampling Synchronization block in charge of re-sampling the video signal through an auto-correlation calculation. It is then sent to the Horizontal Sync block to center the image in the horizontal direction, to later convert the samples to reals as keeping their magnitudes avoids any issue due to argument misalignment. The Normalize Flow bock normalize the values of the samples' magnitudes and de Video SDL Sink block displays the eavesdropped video with a result as the one in Figure 8.

However, this original version of gr-tempest still has some room for improvement regarding the following areas:

- Lacks vertical video centering
- Fine Sampling Synchronization requires high processing power to continuously calculate the re-sampling and execute the interpolation
- Fine Sampling Synchronization has an issue in the constructor when calculating the sample correction
- There is no resolution inference function
- There is no function meant to improve the image quality and allow for details to be interpreted

These points set the objectives for this project.

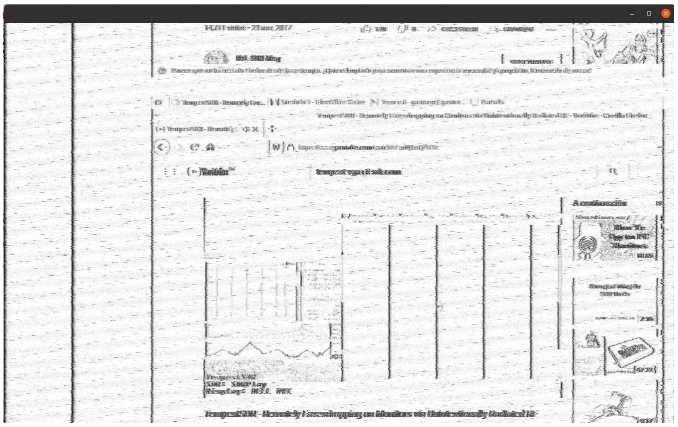


Figure 8. Original gr-tempest output example.

V. Hardware Design

Using the signal characterization described before, it is now necessary to design a hardware system that can enable the implementation of TEMPEST for real life monitor signals.

We have established by now that the SDR is the main component of the system, which is in charge of the signal filtering, demodulation and sampling. Regardless, some additional signal conditioning is required to capture the specific emanations from a selected harmonic.

An RF cascade of components consisting of a pass-band filter, an RF power amplifier and a directive antenna will be selected through this section's analysis to compose the RF front end.

A. System Requirements

The signal's center frequency at the second harmonic emanation of VGA 1080p is 297 MHz. Estimating a bandwidth of about 100MHz, the signal conditioning will deliver a cleaner signal that could be successfully gathered by an SDR device's RX input port and later sampled at a sampling rate of 50 MHz. Bearing in mind that there are few devices that can process information faster than this on the considered range of the SDR's off-the-shelf market, an inherent problem for the project is set: as the video signal pixel pulses are transmitted at a rate of 148.5 MHz, the spied video signal received will always remain under-sampled by the SDR's ADC. The discussed RF band is not particularly interfered by FM radio signals and TV digital transmission signals in Uruguay, as shown in Figure 9. Therefore, selecting this harmonic for eavesdropping is the best option.

This scenario leaves the following requirements for the hardware system:

1) Passing band centered around the selected harmonic at $2 \times f_p = 297$ MHz.
2) The RF cascade components should minimize the input noise power delivered to the SDR device. Noise Figure considered, the first stage of the cascade will attenuate the noise figure of each of the following stages by the value of its device's gain, so initiating the cascade with the device that generates the larger gain is the best option.

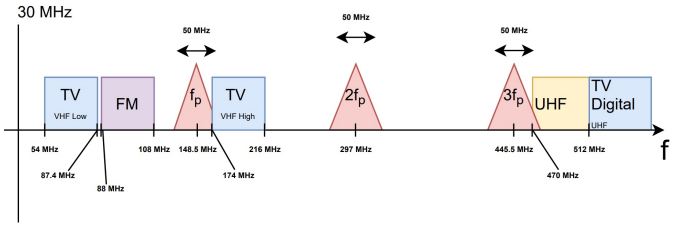


Figure 9. Radio spectrum bands close to the frequencies of interest.

3) Minimize the distortion of the signal. To sample a spectrum containing 50 MHz bandwidth of information, the passing band should be at least four times larger than the maximum frequency of interest - 25 MHz - given that there is an anti-aliasing filter inside the SDR.
4) A directional antenna or beam antenna with selective frequency centered around the passing band of the RF cascade devices.

B. Circuit Design

Given the scope of the project, designing the RF cascade was based on the available market's off-the-shelf RF devices. Having in consideration the availability of each required item on the market, some trade-offs were expected, the items listed below and shown in Figure 10 were selected:

1) Mini-Circuit LNA ZX60-P103LN+ [12].
   - Minimum gain of 23 dB to 18 dB between 50 MHz and 500 MHz, respectively.
   - NF = 1.2 dB to 0.4 dB between 50 MHz and 500 MHz, respectively.
   - $SWR(f) < 2$ dB between 50 MHz and 500 MHz.
   - Requires an external PSU DC 5V@95mA.
2) Mini-Circuit HPF SHP-250+ [13].
3) Mini-Circuit LPF SLP-450+ [14].
4) Ettus Research USRP B200mini SDR [15].
   - FPGA Xilinx Spartan-6 XC6SLX75 [16].
   - Analog Devices AD9364 RFIC direct-conversion transceiver [17].
   - Full duplex, SISO (1 Tx, 1 Rx).
   - Frequency range 70 MHz to 6 GHz.
   - Maximum sampling rate 56 MHz.
   - Maximum Input Power 0 dBm.

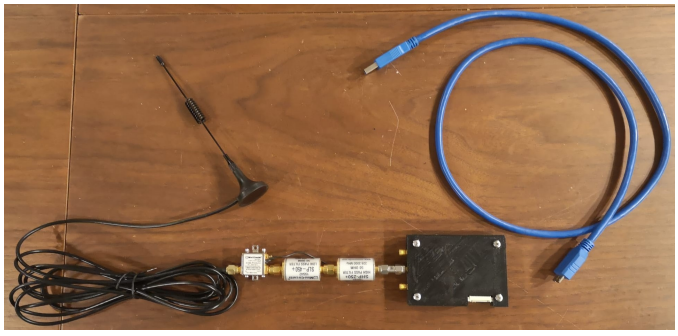


Figure 10. Hardware configuration: Antenna, LNA ZX60-P103LN+ (with 5V source), HPF SHP-250+, LPF SLP-450+ and USRP B200mini.

The full system was tested in a laboratory environment and performed as expected in the frequencies of interest, delivering the desired noise figure and standing wave ratio values. The antenna, though lacking the directivity is was intended to have, was still able to receive the signals with a high enough power to visualize the monitor's video. This will be shown in the real time examples shown in section VII.

VI. Software Architecture

The architecture is based on the following GNU Radio features:

- the flow-graph and its scheduler to run the CPU's different processes, blocks or tasks of gr-tempest
- the Message Passing Interface (MPI) as an asynchronous inter-processes communication tool
- the flow-graph's circular buffers as a single ended input-output bound synchronous data-stream between each of the signal processing blocks

The use of MPI allows to send information from one block to the other. Moreover, this procedure is not constrained to a down-stream information flow, as opposed to the common data-stream. This makes it essential to notify up-stream blocks about updates of parameters calculated during run time.

The functionalities developed under this logic in order to implement all desired TempestSDR features or improve gr-tempest's are:

Synchronization Detection: Uses a single frame's horizontal and vertical averaged and filtered intensity values, summed up into a two dimensional array in order to detect its peaks of energy that correspond to the estimated index position of the beginning of the blanking stripes for the given frame. Using these values as a number of samples delay correction on the output-stream, it is able to reproduce a correctly aligned frame, updating in real-time to the fluctuations of the eavesdropped signal and ordinary issues of the system, like sample drops and buffer overflows that result in missing part of the video stream.

Sampling Correction: Uses the auto-correlation of the input signal in order to calculate the timing error between the SDR's current sampling period and the eavesdropped video signal's pixel rate. Analyzing the peaks found in the auto-correlation, it is possible to find the number of samples that are currently being detected as a single frame of the input data stream. Given a sample rate, the ratio comparison between the detected and expected number of samples in a frame is able to control an interpolation block down-stream, meant to correct the timing differences.

Resolution Inference: Uses the auto-correlation of the input signal in order to calculate the refresh rate of the eavesdropped video signal given by the sample number where the maximum of the signal is found. Furthermore, it is possible to obtain the vertical screen size of the spied video's resolution by calculating the distance between samples of two consecutive smaller peaks of the auto-correlation. This is due to the similarity of the video image between consecutive horizontal lines, which correspond to the correlation of the signal with itself when delayed a complete line.

Linear Equalization: Dropping the complex part of the video stream gives a much clearer image by eliminating the noise generated by the complex to magnitude block while spying a video stream as a zero-mean signal, because of the DC component of the VGA signal not being emanated. Taking the real part, and applying a linear de-emphasis filter in order to compensate for the effects of the sub-sampling of the video signal, is a valid brute-force approach to equalizing the received signal with notorious improvements to the general perception of the image quality.

Central Frequency Correction: While analyzing the results of the equalization when displaying the real part of the video stream, the central frequency error effect, produced by a shift in frequency that is given by the difference between the clock of the spied signal and the clock of the reception system, starts showing on the spy monitor. The visible effect is a sinusoidal being added to the intensity of multiple contiguous lines of pixels in the frame. A successful procedure to reduce the angular velocity of the sinusoidal is to modulate the data input stream with a sinusoidal of opposite angular velocity than the one generated by the central frequency error effect. This block uses the phase difference of the complex valued auto-correlation of the input signal, when the delay is equal to a complete single vertical line in order to estimate the angular velocity of the central frequency error effect that is being applied to the entire frame.

All of these functionalities were implemented within an OOT module that was published [18] for the purpose of this work, whilst also making use of GNU Radio's own utilities and third-party blocks that are publicly available.

VII. Results

All of the explained components were tested and evaluated under appropriate criteria to measure their results. This criteria was set in each case according to the component's alignment with the main objective of the project: to improve and extend the functionalities of gr-tempest.

The first component to be tested was the first one implemented, as it was required to begin every software test: the RF front end. The radiofrequency components were tested with a Rohde & Schwarz ZVB 8 VNA, resulting in the condition of $SWR_{in} < 2dB$ which verifies the requirement for the hardware, and band-pass limits that allow the reception of the frequencies of interest to eavesdrop the emanations of a $1920 \times 1080@60Hz$ resolution monitor.

Regarding the software, the resolution inference function was tested as a separate module from the rest of the system since it does not require the rest of the components to operate. Since the RF front end is built for a specific resolution, $1920 \times 1080@60Hz$ is the first monitor resolution to be inferred, returning the correct values for said resolution. After removing the filters, several other resolutions were also tested and resulted in satisfactory results with the correct resolution parameters every time.

The process of image centering was tested first on simulated scenarios and then in recordings of real emanated signals. In both cases, the result was the same: the horizontal synchronization was done correctly, fixing the video frame at its

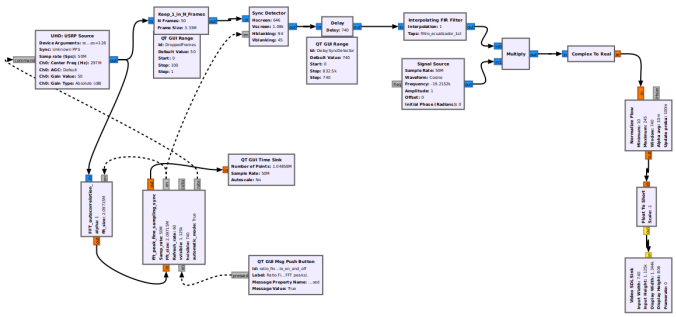


Figure 11. Full gr-tempest 2.0 flow graph.

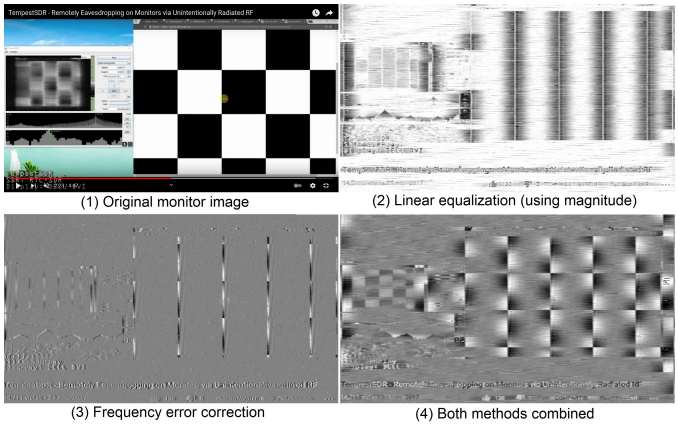


Figure 12. Comparison between equalization methods.

center, while the vertical synchronization fixed the video but usually not centered. This is due to the border detection effect: given the explained characteristics of the signal, the border detection only works for consecutive samples which only run horizontally, therefore describing only vertical lines. The lack of horizontal lines gives less information to the algorithm when estimating the location of the horizontal blankings (above and below the image), resulting in a fixed but vertically displaced frame. The solution to this issue was the inclusion of a manual vertical delay, which only needs to be used when first running the software or when there is a big shift in the energy distribution of the eavesdropped monitor, in cases like a switch between navigator tabs or minimizing a program.

The interpolation ratio calculation now uses an FFT method for estimating the signal's auto-correlation, and in combination with a peak finder algorithm running on the FFT Peaks block results in a successful substitute for one of the functionalities of the Fine Sampling Synchronization block: it calculates accurately the interpolation ratio from the complex signal in a few seconds of operation, occupying less CPU capacity and isolating the function to enable the use of a push button for the user to deactivate the calculation when the re-sampling is reached, thus requiring even less consumption during the rest of the process.

The previous functionality, together with the introduction of the sampling correction via hardware, greatly enhanced the global performance. Returning the correction value to the SDR plugins and drivers through PMT messages allows the signal to enter the system with its sampling already corrected, making it unnecessary to interpolate samples during run time and costing less CPU usage. Moreover, if the incoming signal does not need re-sampling, the size of a full frame in samples is already known by the system, enabling a 'Keep 1 in N' block to discard full frames and making for an ever better overall performance. The inclusion of this block and integration of the previous allowed to reach a working sampling rate of 50 MSps. The full system is shown in Figure 11.

The flow graph also reveals the inclusion of equalization to the online espionage, which was originally thought of as a separate, offline functionality for signal recordings.

For the evaluation of the equalization components, there are results for the two separate methods explained -linear equalization and frequency error correction- and the two of them combined. These effects are compared in Figure 12. The first picture shows the original screen in the spied monitor. The following show in order: only lineal equalization taking samples' magnitude, only frequency error correction, and the both methods combined.

With this comparison, the effects of both methods come to light. On the one hand, frequency error correction makes the hole image to take a grey tone due to the information loss that was taking place because of keeping only the samples' magnitude. Also, the vertical lines show white and black strips, that narrow down for bigger frequency errors - correcting the frequency makes these line more uniform.

On the other hand, linear equalization make the vertical lines thicker. This effect is caused by the enhancement of the lower frequencies, contributing to the energy distribution of the lines. This consequence helps with to reconstruct the picture, as the thickening behaves similarly to a shadow effect, making it easier for the viewer to identify the shapes.

The spy monitor shown in Figure 13 shows how the mix of this effects improve the definition of the letters in the spied image, allowing to identify words that were otherwise lost information.

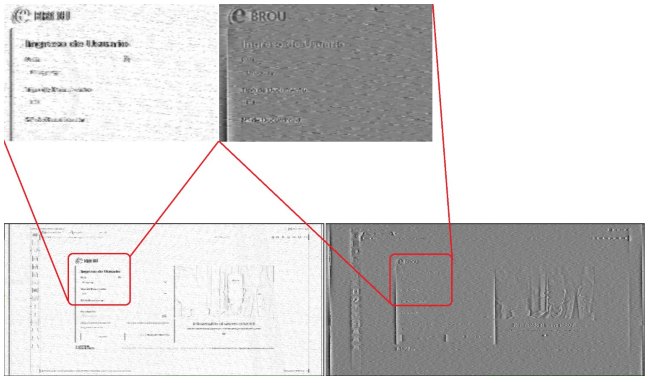


Figure 13. Detail comparison with and without equalization.

Finally, Figure 14 compares an original spied monitor image with its corresponding eavesdropped result. The screenshot was taken during an online operation using the final flow graph shown, and fully corrects the frequency error.

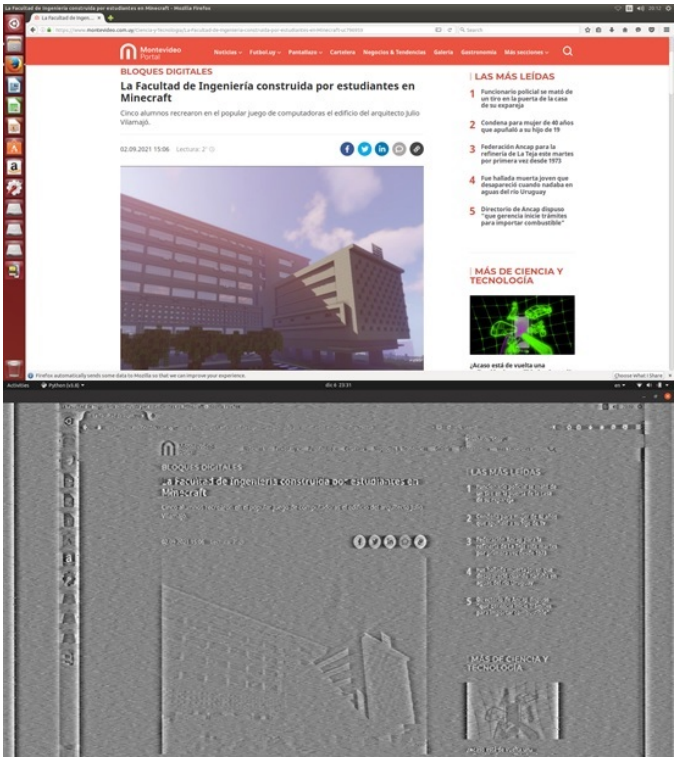


Figure 14. Full system espionage result. Spied monitor (up) vs spy monitor (down).

VIII. Conclusions

The topic of monitor eavesdropping through involuntary electromagnetic emanations was taken on in this work, with the intention of establishing, designing and implementing improvements to the original gr-tempest system based on TempestSDR and other bibliography, contributing to the studies of TEMPEST.

The objectives set for this project were accomplished successfully, modifying the gr-tempest software to add or correct the original code based on the room for improvement that could be detected and the corresponding theoretical analysis.

A hardware front end was implemented to enable the reception of electromagnetic emanations for $1920 \times 1080@60Hz$ monitors. This was useful as a method to develop and test the software. It also enabled the more detailed study of the radiofrequency signal and its interaction witn the key component involved in the design of the system: SDR. The benefits of Software Defined Radio were seen both in theory and in practice.

Every separated component of software was designed, coded and tested properly, obtaining the desired results in each case: resolution detecting, image centering, re-sampling, performance optimizing and equalization were all implemented.

The simultaneous operative of these components was also successful, achieving the online eavesdropping with reasonable CPU consumption and 50 MSps sampling.

This more robust and complete system evidences the improvements achieved on gr-tempest, confirming the relevance of both SDR and GNU Radio. It also highlights the relevance of TEMPEST as a field of study, allowing very cheap systems to expose the security breach that exists in the most common image display methods in our digital components.

References

[1] NSA FOIA Case #51633. Approved for Release by NSA (2007), *TEMPEST: A Signal Problem.*
[2] Elsevier Science Publishers B.V, W. van Eck. (1985), *Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk?*
[3] University of Cambridge Computer Laboratory, Markus G. Kuhn (2003), *Compromising emanations: eavesdropping risks of computer displays.*
[4] Martin Marinov (2014), *Remote video eavesdropping using a software-defined radio platform.*, https://github.com/martinmarinov/TempestSDR
[5] Wikipedia, *Software Defined Radio.*, https://en.wikipedia.org/wiki/Software-defined_radio
[6] Facultad de Ingeniería, Universidad de la República. Pablo Menoni (2018), *Espionaje por Emisiones Electromagnéticas. Masters Thesis.*
[7] gr-tempest (2020), *gr-tempest by Federico Larroca*, https://github.com/git-artes/gr-tempest
[8] Facultad de Ingeniería, Universidad de la República. Pablo Belzarena and Federico La Rocca (2017), *Comunicaciones Inalámbricas. Notas del Curso.*
[9] Víctor González-Barbone and Federico 'Larroca' La Rocca (2020), *GNU Radio. "out-of-tree" Modules Development.*
[10] Wikipedia, GNU Radio., https://en.wikipedia.org/wiki/GNU_Radio
[11] VESA Standard (2013), *VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT)*, https://glenwing.github.io/docs/VESA-DMT-1.13.pdf
[12] Mini-Circuit (2021), *Mini-Circuit LNA ZX60-P103LN+*, https://www.minicircuits.com/WebStore/dashboard.html?model=ZX60-P103LN%2B
[13] Mini-Circuit (2021), *Mini-Circuit HPF SHP-250+*, https://www.minicircuits.com/pdfs/SHP-250+.pdf
[14] Mini-Circuit (2021), *Mini-Circuit LPF SLP-450+*, https://www.minicircuits.com/WebStore/dashboard.html?model=SLP-450%2B
[15] Ettus Research (2021), *Mini-Circuit USRP B200mini*, https://www.ettus.com/wp-content/uploads/2019/01/USRP_B200mini_Data_Sheet.pdf
[16] Xilinx (2021), *FPGA Xilinx Spartan-6 XC6SLX75*, https://www.xilinx.com/support/documentation/data_sheets/ds160.pdf
[17] Analog Devices (2021), *AD9364 RFIC direct-conversion transceiver*, https://www.analog.com/media/en/technical-documentation/data-sheets/AD9364.pdf
[18] gr-tempest-2.0 (2021), *gr-tempest-2.0*, https://github.com/fcarraustewart/gr-tempest