Tipo de artículo: Artículo original

Visualizador digital de un radar de seguimiento empleando Raspberry Pi 4

Digital display of a tracking radar using Raspberry Pi 4

Lisvan Guevara Trujillo[1*], **https://orcid.org/0000-0002-1830-2045**
Leandro Zambrano Méndez[2], **https://orcid.org/0000-0002-8243-2813**
Wenny Hojas-Mazo[2], **https://orcid.org/0000-0002-8298-3439**
Margarita André Ampuero[2], **https://orcid.org/0000-0001-5088-6039**

[1] Centro de Investigación y Desarrollo Naval. Calle Estrada Palma No. 13, Casa Blanca, Regla, La Habana, Cuba. Correo electrónico: lisvan94trujillo@gmail.com
[2] Departamento de Inteligencia Artificial e Infraestructura de Sistemas Informáticos, Facultad de Informática, Universidad Tecnológica de La Habana "José Antonio Echeverría". Calle 114 No. 11901, Marianao, La Habana, Cuba. Correo electrónico: {lzambrano, whojas, mayi}@ceis.cujae.edu.cu

*** Autor para correspondencia: lisvan94trujillo@gmail.com**

Resumen

Un visualizador de radar es un dispositivo electrónico que se utiliza para presentar una imagen gráfica continua, que permita comprender de forma fácil la posición relativa de los objetos detectados. Los visualizadores modernos realizan la representación a partir de la obtención de señales digitales. La mayoría de los visualizadores de radares de seguimiento con que se cuenta están sustentados en tecnología analógica. Estos son altos consumidores de energía eléctrica, poseen gran volumen y peso, sus piezas están obsoletas y existe incapacidad de adquirir los repuestos en el mercado internacional. Estos visualizadores están sometidos a largos períodos de mantenimiento, al presentar inestabilidad durante su funcionamiento. El objetivo de este artículo es diseñar e implementar un visualizador digital de la información de señales de radar que cumpla con el período de actualización de un radar de seguimiento. En la solución propuesta se emplea el ordenador de placa reducida Raspberry Pi 4 y el software fue elaborado en el entorno de desarrollado multiplataforma Qt Creator. Además, se empleó la computación paralela para lograr la recepción, el procesamiento y la representación de los datos cumpliendo con el período de actualización de la información del radar de seguimiento. Esto propició disminuir el tiempo de ejecución del software y cumplir con el período de actualización del radar, a diferencia de la versión secuencial donde no se cumplió con la restricción temporal. Como resultado se obtuvo una sustitución tecnológica del sistema visualizador analógico que posee bajo consumo eléctrico, costo y tamaño.

Palabras clave: visualizador de radar; radar de seguimiento; software multiplataforma; computación paralela; Raspberry Pi 4.

Abstract

A radar display is an electronic device used to present a continuous graphic image, which allows easy understanding of the relative position of detected objects. Modern displays perform the representation by obtaining digital signals. Most of the available tracking radar displays are based on analog technology. They are high consumers of electrical energy, have a large volume and weight, their parts are obsolete and it is impossible to acquire spare parts from the international market. These displays are subject to long maintenance periods, as they are unstable during operation. The objective of this work is to design and implement a digital display of radar signal information that complies with the update period of a tracking radar. The proposed solution uses the Raspberry Pi 4 single board computer and the software was developed in the Qt Creator cross-platform development environment. In addition, parallel computing was used to achieve the reception, processing and representation of the data, complying with the update period of the tracking radar information. This allowed to reduce the software execution time and to comply with the radar update period, unlike the sequential version where the time restriction was

not met. As a result, a technological replacement of the analog display system, which has low power consumption, cost and size, was obtained.

Keywords: *radar display; tracking radar; cross-platform software; parallel computing; Raspberry Pi 4*

Recibido: 12/07/2022
Aceptado: 28/08/2022
En línea: 01/09/2022

Introducción

Los radares son elementos fundamentales en la seguridad y la defensa, por su papel en la obtención de los datos básicos que permiten el funcionamiento de los sistemas de control de tráfico terrestre, aéreo y marítimo. Según su finalidad los radares se clasifican en radar de búsqueda y de seguimiento. Un radar de búsqueda es utilizado principalmente para la detección inicial de objetivos en un volumen de interés (IEEE, 2017). El radar de seguimiento tiene como función principal el seguimiento de objetivos, que se logra manteniendo el haz sobre el objetivo (IEEE, 2017). Una de las diferencias entre estos tipos de radar es la frecuencia de actualización de la información. Los radares de exploración poseen una tasa de actualización del orden de 1 - 0.1 Hz y los de seguimiento de 100 – 10 Hz (Butler, 1998). Conceptualmente, un sistema de radar consta de cinco componentes: un generador, un receptor, un amplificador, un procesador y un visualizador (Acosta *et al.*, 2006; Acosta *et al.*, 2007). Los componentes empleados para la transmisión y recepción en radiofrecuencia se ofrecen habitualmente como dispositivos COTS (*commercial off the shelf*), mientras que el procesamiento de señal radar, es el encargado de brindar valor agregado (Gómez, 2017). El visualizador es un dispositivo electrónico que se utiliza para representar en un formato adecuado la información procesada por el analizador (Kavyashree *et al.*, 2017). Para mostrar la información, pueden emplearse una variedad de formatos. Entre los tipos más comunes de indicaciones se encuentra la Indicación de Posición en el Plano los cuales suelen utilizarse en radares de vigilancia que efectúan coberturas en los 360º y la indicación tipo B que es empleada en el aterrizaje de precisión de aeronaves (Kaushik, 2014). Estas indicaciones deben presentar al observador una imagen gráfica continua, que permita comprender de forma fácil la posición relativa de los objetos detectados, lo que contribuye a la eficiencia del proceso de toma de decisión por los operadores (Kaushik, 2014; Sulistyaningsih *et al.*, 2019).

Tradicionalmente, todos los componentes de un radar han sido analógicos (Gómez, 2017). Estos son altos consumidores de energía eléctrica, poseen gran volumen y peso, sus piezas están obsoletas y existe incapacidad de adquirir los repuestos en el mercado internacional. También, están sometidos a largos períodos de mantenimiento al presentar inestabilidad durante su funcionamiento, provocado fundamentalmente por el envejecimiento de sus partes y

piezas. Sin embargo, con la evolución de los sistemas digitales se han ido transformando para usar la señal digital (Gómez, 2017). Para la implementación de un sistema de procesamiento digital de señal de radar se suelen emplear dispositivos electrónicos programables como microcontroladores, DSPs (*digital signal processing*), FPGAs (*field-programmable gate array*) y CPLDs (*Complex Programmable Logic Device*) (Gómez, 2017). Para la presentación de la información de las señales de radar se emplean medios de cómputo y monitores digitales (Jevtić and Stamatović, 2009; Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013), siendo esta una de las vías para la solución de los problemas planteados. Conociendo la inestabilidad en el funcionamiento del sistema de visualización analógico y la necesidad de obtener un sistema visualizador que cumpla con las exigencias operativas del radar de seguimiento, este trabajo propone la digitalización del visualizador.

Varios autores han abordado la implementación de visualizadores digitales cumpliendo con las exigencias de tiempo y procesamiento que impone el radar, empleándose diversas plataformas hardware y software (Acosta *et al.*, 2006; Acosta *et al.*, 2007; Jevtić and Stamatović, 2009; Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013; Manasa and Hemalatha, 2015; Montamat, 2015; Ravindra *et al.*, 2017; Tian *et al.*, 2017; Saputera *et al.*, 2018; Sulistyaningsih *et al.*, 2019). Entre las ventajas que posee la recepción de la señal digital está la posibilidad del procesamiento automático de la información, además de aumentar la cantidad de información a representar en pantalla, facilitando la comprensión y toma de decisiones a los operadores, evidenciándose mejoras respecto a la limitada representación en las antiguas pantallas de fósforo. A continuación se realiza un análisis de soluciones dadas en la implementación de visualizadores digitales de radar, prestándose atención a los materiales empleados, a elementos de diseño del software y la validación del sistema.

En (Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013) se desarrolló un visualizador digital de la situación aérea para remplazar uno antiguo basado en tubo de rayos catódicos. Por razones de rendimiento la aplicación fue escrita en C++ y la representación de la situación aérea se realizó en un hilo independiente. Para simplificar el desarrollo se utilizó el framework Qt. Los autores plantean que al sistema se le realizaron pruebas de rendimiento en los que se obtuvieron resultados satisfactorios, aunque no se explicó en qué consistieron las mismas.

Los autores de (Manasa and Hemalatha, 2015) desarrollaron un visualizador de datos de radar en tiempo real, el cual es usado para probar y analizar los algoritmos del radar y para el entrenamiento de los usuarios. Los autores emplearon una distribución de GNU/Linux para el desarrollo de este componente del sistema de radar, pues plantean que es un sistema operativo de código abierto, posee alta estabilidad y buena robustez.

En (Montamat, 2015) se desarrolló un componente del sistema de radar denominado Extractor Digital de Datos de Radar, permitiendo modernizar radares analógicos. Una de las funciones que permite este componente es la

visualización de la situación aérea. Se empleó la programación paralela mediante la creación de hilos de ejecución en diferentes puntos del código fuente, lográndose un mejor aprovechamiento de los recursos de cómputo, aunque no se realizó una comparación de la solución secuencial con respecto a la solución paralela empleada.

Por su parte en (Ravindra *et al.*, 2017) se diseñó e implementó una aplicación para la visualización de la situación aérea capaz de correr sobre varios sistemas operativos. El dibujo de los objetivos aéreos se realizó empleando la clase QPainter. Según los autores este sistema puede ser empleado en radares antiguos.

Los autores de (Tian *et al.*, 2017) presentan un sistema de procesamiento y visualización multiplataforma de la información del radar. En el sistema se auxilian de un FPGA para el procesamiento de los datos, ya que eran más de un millón de datos a analizar. Para realizar la representación se empleó un computador de bajas prestaciones. En este trabajo se propuso combinar Graphics-view con OpenGL para lograr la visualización de imágenes de radar de forma eficiente, además, de hacer uso de múltiples hilos de ejecución.

En (Saputera *et al.*, 2018; Sulistyaningsih *et al.*, 2019) se diseñó y desarrolló un visualizador de radar para monitorear el espacio aéreo de Indonesia, los autores plantean que uno de los parámetros más importantes a tener en cuenta durante el procesamiento y representación es lograr realizarlo en tiempo real.

En el análisis bibliográfico se apreció que el lenguaje de programación mayormente empleado fue el C (Saputera *et al.*, 2018; Sulistyaningsih *et al.*, 2019) y el C++ (Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013; Manasa and Hemalatha, 2015; Montamat, 2015; Ravindra *et al.*, 2017), así como el uso del Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) Qt Creator (Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013; Manasa and Hemalatha, 2015; Ravindra *et al.*, 2017; Tian *et al.*, 2017; Saputera *et al.*, 2018; Sulistyaningsih *et al.*, 2019). Además, se emplearon diferentes distribuciones de GNU/Linux (Stamatovic *et al.*, 2012; Manasa and Hemalatha, 2015; Montamat, 2015; Ravindra *et al.*, 2017; Tian *et al.*, 2017; Saputera *et al.*, 2018; Sulistyaningsih *et al.*, 2019), no siendo así en (Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013) que emplearon Windows 7, lo cual podría provocar un aumento del costo del proyecto al ser este sistema operativo privativo. La mayoría de los trabajos hacen uso de la interfaz Ethernet para la recepción de los datos (Stamatovic *et al.*, 2012; Manasa and Hemalatha, 2015; Montamat, 2015; Ravindra *et al.*, 2017; Tian *et al.*, 2017; Saputera *et al.*, 2018; Sulistyaningsih *et al.*, 2019) con el empleo del protocolo de comunicaciones UDP (en inglés, *User Datagram Protocol*) y TCP (en inglés, *Transmission Control Protocol*). En (Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013; Tian *et al.*, 2017; Saputera *et al.*, 2018; Sulistyaningsih *et al.*, 2019) se empleó como dispositivo hardware computadoras de escritorio, lo cual podrían tener un costo superior de desarrollo del proyecto, un aumento de consumo eléctrico y volumen con respecto al empleo de otras tecnologías como los ordenadores de placa reducida (SBC, por sus siglas en inglés) que fue empleado en (Montamat, 2015). En

varios trabajos emplearon la computación paralela (Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013; Montamat, 2015; Tian *et al.*, 2017) para lograr una mayor eficiencia con los recursos de cómputo. En los trabajos consultados no se fundamenta la selección de la arquitectura hardware, ya que además de tener que cumplir con requerimientos de fiabilidad, rendimiento en tiempo real y flexibilidad ante posibles cambios, se debería considerar el consumo de energía, el costo, el tamaño y peso. Además, no presentan experimentos detallados con el empleo de métricas de evaluación del rendimiento del software, para sustentar la calidad de las soluciones propuestas.

Materiales y métodos

Para la solución del problema de investigación, dado por la necesidad de resolver los problemas presentes en el visualizador analógico, el sistema propuesto debe cumplir con las exigencias impuestas por el radar. Como parte del análisis del sistema se definieron las características a cumplir. Basado en la definición de requisitos realizada en (Chanchí *et al.*, 2019; Melegati *et al.*, 2019), durante el proceso de captura de requisitos se documentaron los siguientes:

- El período de actualización de la información del sistema visualizador tiene que ser menor o igual a 40 milisegundos.
- Consumo de energía del dispositivo encargado de la representación menor de 15 watts.
- Costo del dispositivo encargado de la representación menor a los 100 USD.
- Tamaño del dispositivo encargado de la representación no mayor de 10 x 10 centímetros.
- Representar la información mediante la indicación tipo B.
- Representar la situación aérea con una distancia máxima de 37500 metros.
- Utilizar la escala de grises durante la representación de la situación aérea.
- Emplear herramientas de software libre en el desarrollo y despliegue del sistema.
- Permitir la fácil migración del código fuente a diferentes sistemas operativos.

Durante el estudio del problema y de las soluciones realizadas por diversos autores para el desarrollo del visualizador de radar, los autores decidieron emplear el ordenador de placa reducida Raspberry Pi 4 modelo B de 4 GB de RAM. Esta decisión está basada en que este dispositivo posee reducidas dimensiones (85mm x 53mm), precio económico (55 USD) y bajo consumo de energía eléctrica, cumpliendo con los requisitos planteados. Esta placa posee buenas capacidades de procesamiento, tiene interfaz Ethernet para la recepción de datos y ha existido un crecimiento del uso de software y hardware libre.

Para cumplir con los requisitos del sistema se optó por emplear el sistema operativo Raspbian, la cual es una distribución de GNU/Linux. Este sistema operativo está basado en Debian y es un software libre y de código abierto. La principal diferencia de Raspbian respecto a otras distribuciones de Linux es que está optimizada para funcionar en procesadores ARM, concretamente en el de la Raspberry Pi, lo que permite un mejor rendimiento del dispositivo. Es un sistema operativo multitarea y multihilo, el cual es muy eficiente en términos de uso de recursos del sistema. Además es un sistema operativo estable, característica necesaria para aplicaciones de este tipo, donde el mantenimiento del sistema es esporádico y el tiempo de ejecución continua predomina. Es menos propenso a malware y virus.

Los autores de (Pereira *et al.*, 2017; 2021) evidenciaron que los lenguajes C y C++ poseen bajos tiempo de ejecución y consumo de energía. Se seleccionó el C++, ya que incluye muchos aspectos del lenguaje C, pero además incorpora muchas características sofisticadas tales como: programación orientada a objetos, excepciones, sobrecarga de operadores y uso plantillas (*templates*). También C++ posee control absoluto de memoria y se pueden usar todas las técnicas de manejo de punteros y conteo de referencia. Aunque su lanzamiento fue hace más de 35 años posee una comunidad que lanza actualizaciones con nuevas funcionalidades para adaptarlo a las necesidades del software y hardware actual. Es un lenguaje muy extendido, por lo que existen muchos tutoriales, libros, códigos fuentes y material disponible.

Para la elección del IDE se tuvo en cuenta que fuera una aplicación de software libre, multiplataforma y que permita programar en C++, por lo que se seleccionó Qt Creator. Esta decisión está basada, ya que está certificado su uso en sistemas embebidos al cumplir con las normas IEC 62304:2015, la EN 50128:2011, la ISO 26262:2018-6 y 2018-8. Además, los autores de (Thelin, 2007; Rischpater, 2013; Tan *et al.*, 2018) plantean que Qt es una buena opción para usar en sistemas embebidos, debido a que posee un bajo consumo de memoria, es legible y portable en diferentes plataformas. La mayoría de los trabajos consultados donde se desarrollan visualizadores digitales (Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013; Manasa and Hemalatha, 2015; Ravindra *et al.*, 2017; Tian *et al.*, 2017; Saputera *et al.*, 2018; Sulistyaningsih *et al.*, 2019) emplean el entorno de desarrollo integrado Qt Creator.

Diseño e implementación del visualizador del radar

La Figura 1 muestra la propuesta de esquema de solución del sistema visualizador de señales para un radar de seguimiento. Enmarcado en rojo el alcance de solución del trabajo.

Figura 1. Esquema de solución del sistema visualizador.

La Raspberry Pi 4 recibirá la información de la señal del radar por interfaz Ethernet desde el bloque analizador. Los datos recibidos corresponden a muestras de amplitud de la señal del radar. Cada recepción es un barrido del radar, el cual contiene 6250 muestras de 8 bits. Una exploración del espacio aéreo lo componen 128 barridos. El período de actualización de la información de cada exploración es de 40 milisegundos, constituyendo este la restricción de tiempo impuesto por el radar. Cada barrido recibido será procesado en el ordenador de placa reducida y luego representado en un monitor.

Para la recepción de datos se empleó el protocolo UDP, el cual es un protocolo simple y muy básico de la capa de transporte que no está orientado a conexión (Buñay *et al.*, 2019). Esto implica que ambos extremos del enlace transmiten datos sin mecanismos de negociación ni control entre ellos y no se produce retardo añadido al envío de paquetes a causa de la creación de una conexión (Gago, 2019; Leibovich *et al.*, 2019). El tamaño de la cabecera UDP es de tan solo 8 bytes produciendo una baja sobrecarga durante el envío de paquetes (Gago, 2019). UDP proporciona un servicio rápido y produce un retardo bajo (Beltrán *et al.*, 2017; Buñay *et al.*, 2019; Gago, 2019). Para la implementación del módulo receptor de datos se empleó la clase QUdp, disponible en Qt Creator. Este receptor vincula el puerto y la dirección IP de la interfaz de red por donde se va escuchar con *bind*() (Srivinay *et al.*, 2019). El evento *readyRead*() indica que hay nuevos datos esperando ser leídos. Como respuesta al evento *readyRead*() se desarrolló el procedimiento onReceive() para la lectura de los datos. Los valores recibidos, como resultado de la transmisión de un barrido, son almacenados en un arreglo y luego procesados.

Para lograr la representación de la situación aérea en la indicación tipo B se hizo uso de la clase QPainter de Qt Creator. QPainter permite dibujar diferentes formas geométricas como puntos, líneas, rectángulos, imágenes, textos, etc (Ravindra *et al.*, 2017). Para la representación en el indicador de tipo B, además de emplearse QPainter, se empleó la clase QImage. QImage está diseñada y optimizada para operaciones de entrada/salida, y para acceso y manipulación directa a nivel de pixel. Una imagen a color es un arreglo de tamaño M x N x 3, donde M x N define las dimensiones de los planos, mientras que el 3 corresponde a cada componente del formato RGB (Mesa *et al.*, 2018). La función *setPixel*() permite acceder a un pixel específico de la imagen y asignarle el color deseado. Para la

asignación del color de cada pixel se empleó el formato RGB y al ponerse los valores de las tres componentes (rojo, verde y azul) con el mismo valor, se obtiene un color en escala de grises (Mesa *et al.*, 2018). El proceso de pintado de cada barrido en la imagen consiste en acceder a cada uno de los 625 valores de 8 bits y asignarle el mismo valor a las tres componentes del formato RGB para generar el color. Al conocerse el número del barrido a representar, se le asigna el color generado al píxel correspondiente de la imagen. La Figura 2 muestra el pseudocódigo del método encargado de la representación de los barridos.

```
Procedimiento PintarImagen (arrayValores, noBarrido)
    X_Imag ← noBarrido * 2
    Para J ← 0 Hasta 624 Con Paso 1 Hacer
        intensidad ← arrayValores_(J)
        colorPixel ← Rgb(intensidad, intensidad, intensidad)
        Y_Imag ← 624 − J
        imagen.setPixel(X_Imag, Y_Imag, colorPixel)
        imagen.setPixel(X_Imag + 1 , Y_Imag, colorPixel)
    Fin Para
Fin Procedimiento
```

Figura 2. Pseudocódigo del algoritmo de representación de los barridos.

Una vez implementado el software fue ejecutado en el ordenador de placa reducida con que se cuenta. En la Figura 3 se puede apreciar el sistema visualizador en funcionamiento. El sistema está compuesto por el ordenador de placa reducida, el cual está conectado al monitor vía HDMI. Además, se aprecia la conexión del cable de red a la Raspberry Pi 4 y del cable de alimentación.

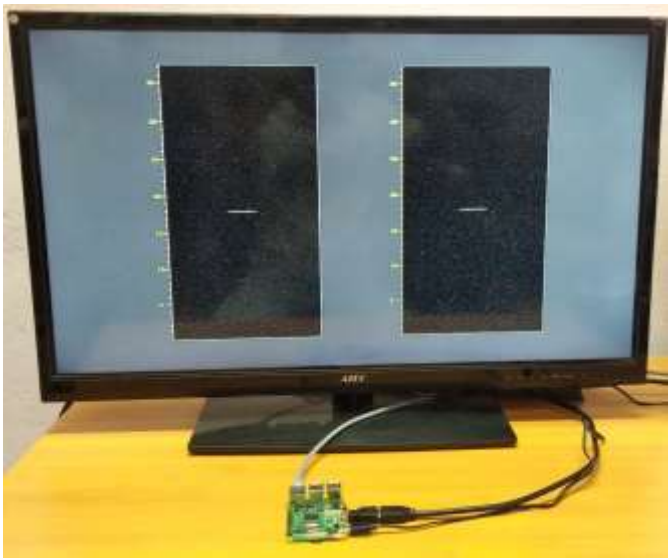

Figura 3. Sistema visualizador de un radar de seguimiento.

Considerando la propiedad de calidad de un sistema/software eficiencia de desempeño, la cual está relacionada con el rendimiento de un dispositivo en función de su comportamiento temporal, entre otras definidas en la Norma ISO/IEC 25023:2016, se midió el tiempo de ejecución del software. En esta medición se detectó que no se cumplió con el período de actualización del radar. En (Piñero *et al.*, 2021) se plantea que la eficiencia del desempeño puede ser afectado por la velocidad de ejecución del procesador, el ancho de banda de la red, la carga de la red, entre otros. Como el ancho de banda empleado es de 1 Gigabit/s, siendo este el máximo soportado por los ordenadores de placa reducida empleados y suficiente para garantizar la recepción de los datos, entonces los esfuerzos fueron concentrados en la velocidad de ejecución del software.

Teniendo en cuenta que varios trabajos emplearon la computación paralela (Stamatovic *et al.*, 2012; Stamatović *et al.*, 2013; Montamat, 2015; Tian *et al.*, 2017), con el objetivo de disminuir el tiempo de ejecución mediante la aceleración de los algoritmos, razón por la cual fue tenido en cuenta para la solución. En (Mena *et al.*, 2017) se plantea que uno de los pasos a seguir para el diseño de un algoritmo paralelo es la identificación de las partes del algoritmo que son más costosas y que puedan ejecutarse de forma concurrente. En (Piñero *et al.*, 2021) se plantea como buena práctica asociada a la eficiencia del desempeño el uso de herramientas dedicadas a las pruebas, la monitorización y el análisis de registros de la eficiencia del desempeño en el software. Para la identificación de las funciones más costosas temporalmente se empleó la herramienta de *profiling* Gprof, la cual es una herramienta de análisis dinámico que extrae información en tiempo de ejecución (Janjusic and Kartsaklis, 2015; Cho, 2018). Dentro de la información recolectada por esta aplicación se encuentra el por ciento de tiempo de ejecución de las funciones, el tiempo de ejecución de cada función, el número de veces que se ejecuta cada función y el tiempo consumido por llamada de cada función (Cho, 2018; Singhal *et al.*, 2019). Para hacer uso de esta herramienta en compiladores **gcc** es necesario compilar el código con la opción **–pg** para que el ejecutable incluya información con el análisis estadístico del rendimiento.

Para la obtención del fichero con el análisis de rendimiento se accedió a través del terminal a la dirección donde se ubica el ejecutable del sistema visualizador mediante el comando cd. Luego se ejecutó el siguiente comando:

gprof Visualizador > log_file

Siendo **Visualizador** el nombre del ejecutable y **log_file** el nombre del fichero que se creará en la misma ubicación del ejecutable, el cual contiene el análisis estadístico del rendimiento generado por **gprof**. En la Figura 4 se muestra el por ciento de tiempo consumido por las funciones del software.

Figura 4. Tiempo consumido por las funciones del software.

La Figura 4 muestra que las funciones más costosas temporalmente son en primer lugar, la encargada de la recepción de datos (representando aproximadamente un 57% del tiempo total), en segundo lugar, la encargada del procesamiento de los datos (aproximadamente un 22%) y por último, la de representación con aproximadamente un 18% del tiempo total. A partir de estos resultados que devolvió la herramienta de *profiling* se hace necesario realizar un diseño que explote las características de la Raspberry Pi 4.

En la versión secuencial del software, para realizarse el procesamiento de los datos y la representación, es necesario primero concluir con la recepción de los datos de un barrido. Utilizando el paralelismo se puede lograr que una vez recibidos los datos correspondientes al primer barrido, pueda realizarse de forma secuencial el procesamiento y representación de estos datos y a la misma vez, la recepción de los datos del segundo barrido, ya que la recepción consume más tiempo que el procesamiento y representación de manera secuencial. El paralelismo que se alcanza puede ser comprendido a través de la Figura 5 en la que se aprecia como varias etapas pueden estar activas al mismo tiempo en la versión paralela.

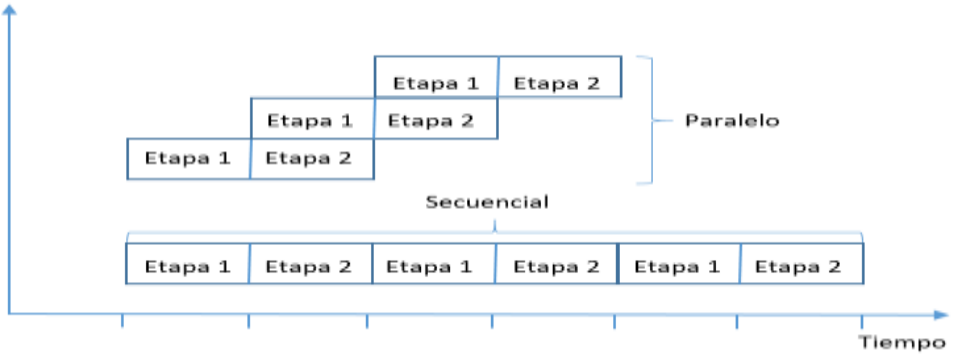

Figura 5. Comportamiento en el tiempo del algoritmo secuencial y paralelo.

En la etapa 1 ocurre la recepción de los datos y en la etapa 2 el procesamiento y representación. Para el desarrollo de la versión paralela del software se hizo uso del paradigma de memoria compartida. Este paradigma consiste en que múltiples núcleos pueden acceder a la misma memoria. Las principales ventajas que posee el uso de este paradigma es la facilidad de programar y rapidez al compartir los datos entre los hilos de ejecución (Mena *et al.*, 2017), siendo estas las principales razones por las que se seleccionó este paradigma. Para el desarrollo de la versión paralela del software se empleó como base la versión secuencial del mismo. El paralelismo empleado fue el de flujo. Este tipo de paralelismo corresponde al principio de trabajo en cadena, ya que se dispone de un flujo de datos sobre los cuales se deben efectuar un conjunto de operaciones (Castro and Leiss, 2004). Este tipo de paralelismo implica un procesamiento de tipo encauzamiento, pues la recepción de datos es continua durante el funcionamiento del sistema. Para la implementación, Qt Creator proporciona clases para hilos que facilitan sacar las tareas de larga duración del hilo principal y provee a cada hilo de una cola de mensajes, lo que permite enviar señales a *slots* en otros hilos. Esto nos proporciona una forma sencilla de pasar datos entre los hilos de la aplicación. Para garantizar que el objeto receptor de datos se ejecute en otro hilo se hizo uso de la clase QThread. Esta clase proporciona una manera independiente de la plataforma para administrar hilos. Para cambiar el objeto receptor de datos del hilo principal se utilizó el método *moveToThread*(), al cual se le pasó como parámetro el objeto de la clase QThread.

Resultados y discusión

Una vez implementadas las versiones del software para realizar el procesamiento y representación de la información de la señal de un radar de seguimiento, se hace necesario verificar si se cumple con la restricción temporal. No completar las operaciones en 40 milisegundos puede resultar en una pérdida grave de la calidad proporcionada por el sistema. Con el objetivo de determinar la configuración óptima del sistema visualizador y determinar los factores que influyen de manera significativa en el tiempo de ejecución, se empleó el diseño de experimentos. Se define como rendimiento del proceso, el tiempo en milisegundos que emplea el sistema en la representación de una exploración del radar.

Los factores controlables juegan un rol primordial en la caracterización de un proceso. Los factores controlables identificados a continuación son descritos:

- Versión del software: se refiere a la forma de identificación de un software como resultado de cambios en el desarrollo del mismo. Fueron seleccionados como nivel bajo de la versión del software el secuencial y el nivel alto la paralela. En estas dos versiones desarrolladas, la única diferencia entre ellas es que en la versión paralela se empleó un hilo de ejecución independiente para la recepción de datos.

- Prioridad del proceso: el sistema operativo le asigna una prioridad a cada uno de los procesos. Los procesos con mayor prioridad se ejecutan antes que los procesos con menor prioridad. Fueron seleccionados dos niveles uno por debajo de lo normal (DN) como nivel bajo y encima de lo normal (EN) como nivel alto.

Los factores no controlables de un experimento son responsables de la variabilidad en el desempeño del proceso. A continuación se describe el identificado:

- Procesos ejecutados por el sistema operativo: en el momento en que se ejecuta el programa hay procesos ejecutándose en el sistema operativo, que consumen recursos arbitrariamente y no pueden ser detenidos.

Se empleó como unidad experimental la Raspberry Pi 4 Modelo B con el sistema operativo Raspbian con el panel de escritorio LXPanel 0.10.0.

Para realizar el diseño de experimentos se decidió hacer un diseño factorial completo, ya que se desea incluir todas las combinaciones posibles de los niveles de los factores. Al tener el factor controlable versión del software 2 niveles y prioridad del proceso 2 niveles, se realizaron 4 combinaciones y cada combinación será replicada 5 veces para disminuir el error experimental, por lo que en total se realizaron 20 corridas. Al utilizarse el mismo número de repeticiones en cada tratamiento, provoca que el diseño sea balanceado. Para que el ambiente en el que se aplican los tratamientos sea lo más uniforme posible, se realizará en orden aleatorio.

Los tratamientos correspondientes al diseño señalado con anterioridad, fueron generados en el software Minitab 19. La Tabla 1 muestra los valores de rendimiento en milisegundos de las 20 ejecuciones realizadas como resultado de los cambios intencionados en los factores del proceso.

Tabla 1. Tratamientos generados por el software Minitab 19 y tiempo de ejecución.

Versión del software	Prioridad	Tiempo (milisegundos)
Paralelo	DN	36,8
Paralelo	DN	37,1
Paralelo	EN	34,6
Secuencial	EN	49,1
Secuencial	DN	51,8
Paralelo	EN	34,7
Paralelo	DN	37,7
Secuencial	DN	48,5
Secuencial	EN	46,6
Paralelo	EN	31,9
Paralelo	DN	34,2
Paralelo	EN	32,4
Secuencial	EN	45
Secuencial	DN	47,7
Secuencial	EN	49,1

Paralelo	EN	35,3
Secuencial	DN	50,3
Secuencial	EN	47,8
Paralelo	DN	36,9
Secuencial	DN	48,5

Luego de obtener los valores de rendimiento para los tratamientos generados, se procedió a generar un conjunto de gráficas para obtener conclusiones sobre los experimentos realizados. Para ello se empleó el software Minitab 19. El objetivo es investigar los efectos de las variables de entrada, conocido como factores, sobre la variable de respuesta. El análisis a los datos recolectados permitirá la determinación del factor más influyente para el cumplimiento del tiempo de ejecución y la determinación de la configuración de factores que optimiza los resultados.

El diagrama de Pareto permite detectar los efectos determinantes de factores e interacciones en el rendimiento del proceso o sistema (Antony, 2014). Este diagrama muestra el valor absoluto de los efectos y se traza una línea de referencia en el gráfico. Cualquier efecto que sobrepase esta línea de referencia es potencialmente importante para el rendimiento del proceso (Antony, 2014). Los efectos son organizados según su valor en orden decreciente. Además, permite visualizar las variables que tienen un mayor impacto sobre la variable de respuesta. Como se aprecia en la Figura 6 la versión del software y la prioridad influyen en el tiempo del software.

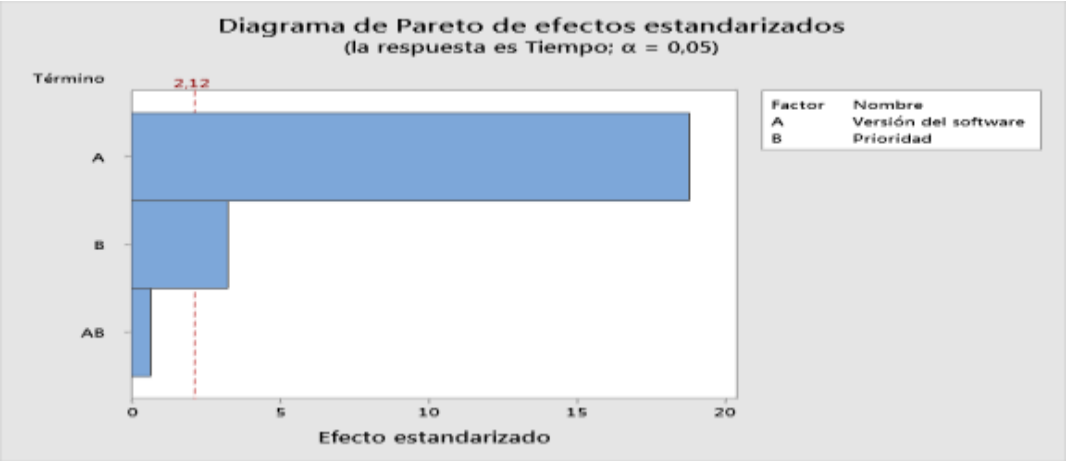

Figura 6. Diagrama de Pareto.

Puesto que el diagrama de Pareto muestra el valor absoluto de los efectos, se determinó que el efecto versión del software es el más significativo. Para analizar la magnitud del efecto de cada factor y comparar fácilmente el efecto entre distintos factores, se emplea la gráfica de efectos principales (Antony, 2014). Un efecto principal es la diferencia en la respuesta media entre los niveles de un factor. En esta gráfica se muestran las medias de tiempo utilizando los dos niveles de la versión del software y las medias de tiempo utilizando las dos prioridades. La línea horizontal

muestra la media de tiempo de procesamiento para todas las corridas. En la Figura 7 se muestra el efecto que tiene cada uno de los factores por separado en el rendimiento del problema.

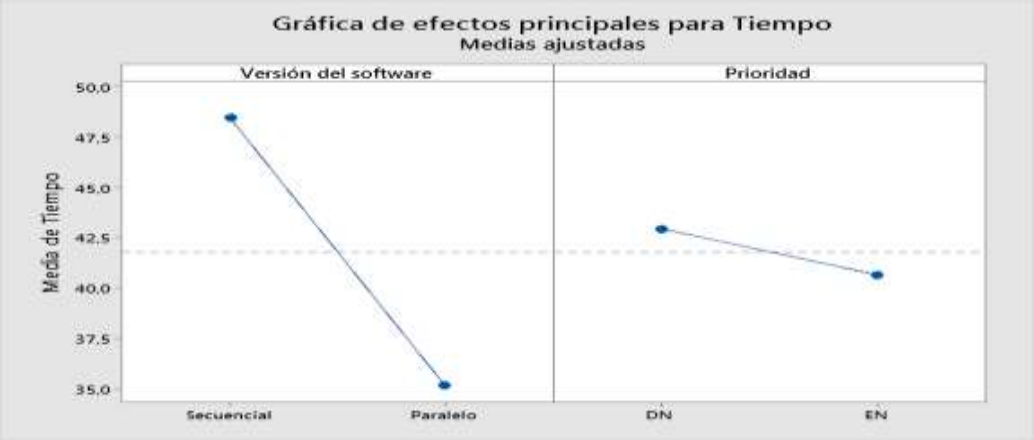

Figura 7. Gráfica de efectos principales.

El factor versión del software al poseer en la gráfica la mayor pendiente, indica que mayor es la magnitud del efecto principal. El nivel en el que se obtienen los mejores valores de rendimiento es en la versión paralela, teniendo un efecto directo sobre el rendimiento. En el caso del factor prioridad, el nivel en el que se obtiene una pequeña disminución del tiempo es por encima de lo normal (EN). Con esta gráfica se corrobora lo obtenido en el diagrama de Pareto, ambos factores influyen en el tiempo de ejecución del software y el más significativo es la versión del software.

Para analizar la interacción entre los dos factores se debe examinar la gráfica de interacción que se muestra en la Figura 8. Cada punto de la gráfica de interacción muestra el tiempo de procesamiento medio con diferentes combinaciones de los niveles de los factores.

Figura 8. Gráfica de interacción.

La gráfica de interacción corrobora que con la ejecución del software secuencial se aumenta el tiempo de ejecución y con la utilización de la versión paralela y prioridad por encima de lo normal, el software se ejecutó con menor tiempo en la Raspberry Pi 4, siendo esta la configuración óptima.

Métricas de rendimiento

Las métricas de rendimiento permiten realizar un análisis cualitativo del rendimiento del algoritmo. El rendimiento de un software puede ser afectado por varios factores, entre los que se encuentran el tamaño del problema, la arquitectura hardware utilizada, el número de procesadores empleados, entre otras.

Una de las métricas más empleadas para determinar si un algoritmo es superior a otro en cuanto a su rendimiento es obteniendo el mismo resultado pero en un menor tiempo de ejecución. Para realizar las pruebas del rendimiento del software se comparan los tiempos obtenidos en las soluciones propuestas en los diferentes dispositivos.

Entre las métricas más empleadas se encuentran: la aceleración o *speed up*, y la eficiencia (Mena *et al.*, 2017), las cuales se definen a continuación.

La aceleración es una medida que captura el beneficio relativo de resolver un problema en paralelo (Goya *et al.*, 2016). Se define como la proporción del tiempo empleado secuencialmente con respecto al tiempo empleado en la versión paralela, aplicados a procesos idénticos (Mena *et al.*, 2017). Esta métrica permite conocer cuántas veces es más rápido el programa paralelo con respecto al programa secuencial. Matemáticamente está definida por la siguiente ecuación:

$$S=\frac{T_s}{T_p} \quad (1)$$

Donde S es la aceleración, T_s representa el tiempo de ejecución secuencial y T_p el tiempo de ejecución paralelo.

La eficiencia es una medida de la fracción de tiempo para la cual un elemento de procesamiento es útilmente empleado (Goya *et al.*, 2016), la cual denota qué tan bien se han utilizado los procesadores. La eficiencia de los programas paralelos decrece con el aumento del número de elementos de procesamiento para un tamaño de problema dado (Goya *et al.*, 2016). Matemáticamente está definida por la siguiente ecuación:

$$E=\frac{S}{P} \quad (2)$$

Donde E es la eficiencia, S representa la aceleración y P el número de procesadores empleados. Para realizar los cálculos de las métricas de rendimiento, fueron utilizados los valores de la configuración óptima. La Tabla 2 muestra los valores de las métricas aplicadas al software.

Tabla 2. Métricas de rendimiento aplicadas al software.

Métrica		Raspberry Pi 4
Promedio del tiempo de ejecución en milisegundos	Tiempo secuencial	47.52
	Tiempo paralelo	33.78
Aceleración		1.40
Eficiencia		0.35
Mejora		29%

La tabla corrobora la disminución del tiempo de ejecución del software a partir del empleo de la versión de programación paralela, al tener la aceleración un valor mayor que 1 y una mejora de un 29% respecto a la versión secuencial. El valor de eficiencia mostrado en la tabla indica que el aprovechamiento de los recursos de cómputo de la Raspberry Pi 4 es de un 35%.

La potencia eléctrica es la energía absorbida o emitida por un dispositivo eléctrico en un instante determinado. Cuando se trata de corriente continua la potencia eléctrica del dispositivo es el producto de la tensión V y la intensidad de corriente que lo atraviesa. Para la determinación de la potencia eléctrica consumida por la Raspberry Pi 4 se empleó una fuente de alimentación. A la fuente se le ajustó el voltaje de salida necesario para el funcionamiento del dispositivo, el cual es de 5.1 V. Luego se midió la intensidad de la corriente en amperios demandada. Estos valores fueron multiplicados para el cálculo de la potencia. Se obtuvo que la potencia consumida por este dispositivo al ejecutarse el software, sin mouse y teclado es de 2.75 W.

Conclusiones

La arquitectura hardware-software propuesta posibilita la representación digital de la información de señales cumpliendo con el período de actualización de un radar de seguimiento y aprovechar de forma eficiente los recursos de cómputos. Al emplearse software libre en la solución propuesta se logra una auténtica independencia tecnológica y el empleo de la Raspberry Pi 4 posibilitó obtener un sistema de bajo costo, tamaño y consumo eléctrico. Con la investigación realizada se obtuvo una sustitución tecnológica del sistema visualizador analógico, permitiendo realizar procesamiento digital de datos y disminuir los mantenimientos técnicos causados por el largo período de explotación y la obsolescencia de las piezas.

Conflictos de intereses

Los autores no poseen conflicto de intereses.

Contribución de los autores

1. Conceptualización: Lisvan Guevara Trujillo
2. Curación de datos: Leandro Zambrano Méndez
3. Análisis formal: Leandro Zambrano Méndez
4. Investigación: Lisvan Guevara Trujillo
5. Metodología: Wenny Hojas-Mazo
6. Administración del proyecto: Lisvan Guevara Trujillo
7. Recursos: Leandro Zambrano Méndez
8. Software: Lisvan Guevara Trujillo
9. Supervisión: Wenny Hojas-Mazo y Margarita André Ampuero
10. Validación: Leandro Zambrano Méndez
11. Visualización: Lisvan Guevara Trujillo
12. Redacción – borrador original: Lisvan Guevara Trujillo
13. Redacción – revisión y edición: Wenny Hojas-Mazo y Margarita André Ampuero

Financiamiento

La investigación no requirió fuente de financiamiento externa.

Referencias

- Acosta, N., Tosini, M. A. & Mezzanotte, M. F. Desarrollo de un Visualizador de Señales de Radar. En: XII Congreso Argentino de Ciencias de la Computación. Argentina: 2006, p. 53-64.
- Acosta, N., Tosini, M. A., Mezzanotte, M. F. & Tommasi, M. Improving radar visualization system. Journal of Computer Science & Technology, 2007, 7 (1): p. 1-7.
- Antony, J. Design of Experiments for Engineers and Scientists. Netherlands, Elsevier, 2014. 221 p.
- Beltrán, R. F., Llumiquinga, I. A., Ávalos, E. & Benalcázar, A. R. Sistema de monitoreo para una plataforma aérea usando sistemas embebidos. Revista Anales, 2017, 1 (375), p. 165-181.
- Buñay, P., Pastor, D., Paguay, P. & Moreno, S. Análisis de la Arquitectura DIFFSERV sobre redes MPLS para la provisión de QoS en aplicaciones en tiempo real (VoIP). Revista Digital Novasinergia, 2019, 2 (1): p. 33-40.
- Butler, J. M. Tracking and control in multi-function radar. Tesis Doctoral, University College London, 1998.

- Castro, J. L. A. & Leiss, E. Introducción a la Computación Paralela. Mérida, Universidad de los Andes, 2004. 246 p.
- Chanchí, G. E. G., Gómez, M. C. A. & Campo, W. Y. M. Propuesta de un videojuego educativo para la enseñanza-aprendizaje de la clasificación de requisitos en ingeniería de software. Revista Ibérica de Sistemas e Tecnologias de Informação, 2019, (E22): p. 1-14.
- Cho, S.-Y. A Program Optimization Method for Embedded Software Developed Using Open Sources. International Journal on Advanced Science Engineering Information Technology, 2018, 8 (4): p. 1692-1697.
- Gago, D. L. Configuración y evaluación de prestaciones del protocolo Multipath TCP. Tesis de ingeniería, Universidad Carlos III de Madrid, 2019.
- Gómez, F. A. S. Metodología de codiseño hardware-software para procesamiento de señales radar en sistemas embebidos. Tesis de maestría, Pontificia Universidad Javeriana, 2017.
- Goya, A. A., Cañizares, D. G., Sánchez, R. M. & Brito, C. C. Análisis de la Escalabilidad del cálculo paralelo de medidas de similitud entre pares de genes. Revista Cubana de Ciencias Informáticas, 2016, 10 (3): p. 71-84.
- IEEE. Standard for Radar Definitions. IEEE Standards Association, New York, USA, 2017, pp. 1-54.
- Janjusic, T. & Kartsaklis, C. Glprof: A gprof inspired, callgraph-oriented per-object disseminating memory access multi-cache profiler. Procedia Computer Science, 2015, 51: p. 1363-1372.
- Jevtić, M. & Stamatović, M. Radar data processing and visualization on desktop platforms. En: 17th Telecommunications Forum (TELFOR). Belgrade: 2009, p. 1315-1318.
- Kaushik, P. Radar Displays. International Journal of Innovative Research in Technology. 2014, 1 (7): p. 472-476.
- Kavyashree, V., Madhu Chaitra, P. N., Vinutha, H. & Rajeshwari, S. A Radar Target Generator for Airborne Targets. International Journal of Advance Research and Innovative ideas in Education, 2017, 2 (15): p. 249-261.
- Leibovich, P. E., Issouribehere, F. & Barbero, J. C. Ensayo y comparación de métodos de transmisión de sincrofasores sobre redes Ethernet. En: XVIII Encontro Regional Ibero-Americano do CIGRE (ERIAC 2019). Foz do Iguaçu: 2019, p. 1-8.
- Manasa, M. & Hemalatha, M. Design of generic radar data visualizer using model view controller pattern. International Journal For Technological Research In Engineering, 2015, 2(12): p. 2990-2992.
- Melegati, J., Goldman, A., Kon, F. & Wang, X. A model of requirements engineering in software startups. Information and software technology, 2019, 109: p. 92-107.
- Mena, O. L., Cordovés, T. C., Suárez, A. R. & Pando, H. D. Algoritmo paralelo para la obtención de predicados difusos. Revista Cubana de Ciencias Informáticas, 2017, 11 (2): p. 117-133.

- Mesa, E. D. A., Vargas, H. S. T., Arango, D. A. G. & Sidek, S. B. Proceso de extracción y almacenaje de características a partir de imágenes de huellas de mordida en el desarrollo de un software para la identificación de personas mediante procesamiento digital de imágenes. Revista Espacios, 2018, 39 (11): p. 2-15.
- Montamat, I. A. Combinador de Información Primaria y Secundaria para Extractor Digital de Datos de Radar en Sistemas de Vigilancia. Tesis de Maestría, Universidad Nacional de Córdoba, 2015.
- Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P. & Saraiva, J. Energy efficiency across programming languages: how do energy, time, and memory relate? En: Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering. Vancouver: 2017, p. 256-267.
- Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P. & Saraiva, J. Ranking programming languages by energy efficiency. Science of Computer Programming, 2021, 205: p. 102609.
- Piñero, M., Marin, A., Trujillo, Y. & Buedo, D. Buenas prácticas para prevenir los riesgos de la eficiencia del desempeño en los productos de software. Revista Cubana de Ciencias Informáticas, 2021, 15 (1): p. 89-113.
- Ravindra, C., Rajkumar, S. & Sreenivasa Babu, M. Design and Implementation of Radar Console Displays for Multi Object Tracking Radar using Qt-IDE. En: 11th International Radar Symposium India. India: 2017, p. 1-4.
- Rischpater, R. Application Development with Qt Creator. Birmingham, Packt Publishing Ltd, 2013. 264 p.
- Saputera, Y. P., Sulistyaningsih, Estu, T. T. & Wahab, M. Radar Software Development for the Surveillance of Indonesian Aerospace Sovereignty. En: International Conference on Electrical Engineering and Computer Science. Pangkal: IEEE, 2018, p. 189-194.
- Singhal, S. P., Gupta, S. & Nuzzo, P. Profiling minisat based on user defined execution time--GPROF. arXiv e-prints arXiv:1909.13058, 2019: pp. 1-13.
- Srivinay, S., Spurthy, K. & Deepika, C. Navigational Display. Perspectives in Communication, Embedded-systems and Signal-processing, 2019, 2 (11): p. 264-265.
- Stamatovic, M., Jevtić, M., Kisić, U. & Tatarević, M. Design and implementation of a modern radar display for air surveillance applications. En: 20th Telecommunications Forum (TELFOR). IEEE, 2012, pp. 1520-1523.
- Stamatović, M., Jevtić, M., Kisić, U., Tatarević, M., Pajić, T. & Marković, K. Modern Air Situation Picture Display for Air Surveillance Radar Applications. Telfor Journal, 2013, 5 (1): p. 54-59.
- Sulistyaningsih, Saputera, Y. P., Wahab, M. & Maulana, Y. Y. Design of radar display of Indonesian airspace monitoring application. TELKOMNIKA, 2019, 17 (3): p. 1176-1184.

- Tan, D.-p., Chen, S.-t., Bao, G.-j. & Zhang, L.-b. An embedded lightweight GUI component library and ergonomics optimization method for industry process monitoring. Frontiers of Information Technology & Electronic Engineering, 2018, 19 (5): p. 604-625.
- Thelin, J. Foundations of Qt Development. United States of America, Apress, 2007. 528 p.
- Tian, Z., Wang, M., Zhou, M. & Qiu, F. Virtual Memory Based Radar Display and Control System. En: International Conference on Machine Learning and Intelligent Communications. Shanghai: Springer, 2017, p. 383-392.