XLIV Jornadas de Automática 2023

https://jautomatica.es

Particionado del software de control de un prototipo de vehículo autónomo

Ortiz, L.[a,*], Valiente, Y.[a], Balbastre, P.[a], Simó, J.[a], Crespo, A.[a]

[a] *Instituto de Automática e Informática Industrial, Universitat Politécnica de Valencia, Camino de Vera, s/n, 46022, Valencia, España.*

Ortiz, L., Valiente, Y., Balbastre, P., Simó, J., Crespo, A. 2023. Partitioning of the control software of an autonomous vehicle prototype. XLIV Jornadas de Automática, 813-818. https://doi.org/10.17979/spudc.9788497498609.813

Resumen

Este artículo presenta el diseño de una arquitectura particionada de criticidad mixta aplicada a un prototipo de vehículo autónomo en el contexto del transporte público. La movilidad inteligente, sostenible y segura es una meta aún no alcanzada, por ello este artículo propone una solución basada en un sistema ciberfísico (CPS). El CPS emula las capacidades del vehículo autónomo, incluyendo la recopilación de datos del entorno, su procesamiento en tiempo real y la toma de decisiones y de acciones mediante los actuadores de este. El principal problema que se presenta en sistemas mixtos son las posibles interferencias causadas por los diferentes niveles de criticidad de las tareas. Se propone como solución el uso del particionado, buscando aprovechar el rendimiento del multiprocesador, aumentar la seguridad y reducir costes en el proceso.

Palabras clave: Sistemas de Criticidad Mixta, Virtualización, Particionado, Control, Vehículo Autónomo.

Partitioning of the control software of an autonomous vehicle prototype

Abstract

This paper presents the design of a partitioned mixed-criticality architecture applied to an autonomous vehicle prototype in the context of public transportation. Intelligent, sustainable and safe mobility is a goal not yet achieved, therefore this paper proposes a solution based on a cyber-physical system (CPS). The CPS emulates the capabilities of the autonomous vehicle, including collecting data from the environment, processing it in real time, and making decisions and taking actions through its actuators. The main problem encountered in mixed systems is the possible interferences caused by different levels of task criticality. The use of partitioning is proposed as a solution, seeking to take advantage of multiprocessor performance, increase security and reduce process costs.

Keywords: Mixed Criticality Systems, Virtualization, Partitioning, Control, Autonomous Vehicle

1. Introducción

La industria automovilística, es un sector en constante investigación en pro a la seguridad. Todo esto se refleja en el incremento de sistemas embebidos que incorporan los nuevos vehículos, destinados principalmente a nuevos sistemas de seguridad como pueden ser el *ABS* o el *ECS*. En vehículos convencionales, algunos de estos sistemas críticos recaen sobre procesadores enfocados a únicas tareas llamados *Electronic Control Unit*s (ECUs)(Wang, 2017), los cuales tienen asociados unos sensores para la adquisición de datos y unos actuadores para dar respuesta.

Hay que destacar que se prevé un aumento en la incorporación de nuevos sistemas, ya que la Unión Europea ha puesto en marcha una estrategia denominada Visión Cero (Commission et al., 2023), para conseguir entre otros objetivos, una mejora en la conducción y una reducción de los accidentes de automóviles. Uno de los retos es incorporar en todos los vehículos los sistemas llamados *Advanced Driver-Assistance Systems* (ADAS), siendo obligatorios para su homologación(Savithry et al., 2019).

El vehículo autónomo comienza a tomar forma y se pueden ver algunos modelos por la calle en pruebas. Aunque en Europa el vehículo con las capacidades más altas homologadas hasta la

*Autor para correspondencia: luioren@ai2.upv.es

fecha es el Mercedes S o el EQS, para el mercado alemán, con capacidades autónomas de nivel 3 (SAE International, 2021). Estas nuevas cualidades, podrían producir una reducción del número de fallecidos en accidentes, ya que en la mayor parte de los accidentes, el factor humano es el responsable. La incorporación en el vehículo autónomo de sistemas de comunicación de baja latencia que permitieran la comunicación en tiempo real con los vehículos circundantes, podría resultar en un aumento de la fluidez del tráfico, al permitir una mejor coordinación entre vehículos (Hidalgo et al., 2020) o al influir en el módulo de control del semáforo (Uribe-Chavert et al., 2022).

Al igual que la industria automovilística, los sistemas embebidos siguen avanzando constantemente, enfocados actualmente en requisitos de *SWaP* (Size, Weight and Power), con el fin de proporcionar sistemas más pequeños, ligeros y con un menor consumo energético, a la vez que se incrementa la capacidad de procesamiento. Estos CPSs (Simó et al., 2021) se ven claramente beneficiados del uso de los nuevos procesadores *multicore*, que permiten incorporar nuevas tecnologías como pueden ser los sistemas particionados y las características de tiempo real. Esto es realmente conveniente, ya que tanto las nuevas exigencias legislativas, como las recientes innovaciones incorporadas en los vehículos, hace que se requieran sistemas de mayor potencia. No solo eso, sino que tienen que ser rápidos, robustos y con la capacidad de intercambiar información entre el resto de componentes.

Comprobar la robustez de los sistemas de seguridad críticos del vehículo, puede ser obligatorio para conseguir la homologación de un vehículo. La certificación es el proceso mediante el cual a fin de garantizar que un sistema es confiable, se comprueba mediante una empresa externa. El sistema deberá soportar diferentes pruebas en diversas condiciones para demostrar que su comportamiento es de acuerdo a lo esperado. Las certificaciones, normalmente van ligadas a estándares internacionales, como pueden ser en el caso de los vehículos la ISO 26262 (International Organization for Standardization, 2018) , que se centra en la seguridad de los sistemas electrónicos o la ISO 21434 (International Organization for Standardization, 2020), enfocada en la ciberseguridad.

1.1. Sistemas particionados

Los sistemas particionados surgen a partir de la necesidad de crear sistemas complejos que integren diferentes componentes funcionales. Estos sistemas, están intrínsecamente ligados con la aviónica, entendida como la unión de la electrónica y la aeronáutica. La arquitectura mayoritaria en la industria aeroespacial era la federada, que había alcanzado el límite de su viabilidad. Cada vez se requerían de más sistemas, lo que repercutía directamente en un incremento del peso de los sistemas, los costes de mantenimiento y el consumo energético. Esto se vio reflejado en un cambio en el desarrollo de los cazas de cuarta generación. Se aprovecharon los nuevos avances tanto en el campo del hardware, más potente y ligero, como en el campo del software ofreciendo la posibilidad de crear sistemas particionados, teniendo como resultado la arquitectura llamada *Integrated Modular Avionics for Space* (IMA).

La arquitectura IMA, se basa en una plataforma centralizada, la cual permite asignar recursos y funcionalidades entre los diferentes subsistemas que la conforman, permitiendo agrupar sistemas de comunicación, de navegación y de motores, bajo el mismo procesador. Esta arquitectura permite reducir los costes de desarrollo, al permitir una reusabilidad, así como los de mantenimiento. Actualmente sigue en uso en aviones y en aplicaciones espaciales la ESA adaptó el estándar IMA al sector espacial (IMA-SP)(Deredempt et al., 2012).

En la industria automotriz se encuentra el estándar llamado *AUTomotive Open System ARchitecture* (AUTOSAR), que es una arquitectura común de software, desarrollada por diferentes empresas pertenecientes a esta industria. La idea de la que surge es introducir los sistemas particionados en los automóviles, creando diferentes capas, buscando así una modularidad, una posible reutilización y la interoperabilidad del software (Jayan and Srinivasan, 2019).

Algunos de los aspectos clave de AUTOSAR son:

- Componentes básicos de software (BSW): es una capa estándar de software, que proporciona los componentes básicos que gestionan las funciones básicas del automóvil y que son independientes del hardware.
- Metamodelo AUTOSAR: es un modelo *UML* que define una estructura de datos común y un conjunto de interfaces, para el intercambio de información entre las diferentes empresas que participen de un desarrollo.
- Metodología de desarrollo: proporciona una guía de desarrollo, que permite el diseño, la configuración y la integración de los diferentes componentes de software.
- Protocolos de comunicación: AUTOSAR define un protocolo de comunicación estándar BCP, que permite la comunicación entre diferentes ECUs.

En el contexto de la seguridad funcional hay una clara relación entre el AUTOSAR y la ISO 26262. AUTOSAR proporciona una serie de especificaciones y pautas para el desarrollo de software robusto, seguro y confiable en el marco de la ISO. Estas especificaciones incluyen la gestión de la seguridad en sus distintos grados y la implementación de mecanismos para cumplir con los requisitos de Automotive Safety Integrity Level (ASIL). Al ofrecer AUTOSAR una arquitectura modular, permite la implementación de seguridad en diferentes componentes de software de manera independiente. La clasificación de la criticidad de las aplicaciones se realiza en función de los riesgos asociados a dichas aplicaciones. La clasificación ASIL se divide en 4 niveles de mayor a menor criticidad D, C, B, A.

A continuación (ver tabla 1), se pueden observar algunos elementos de un automóvil clasificados de acuerdo con ASIL:

Tabla 1: Clasificación de sistemas por criticidad

Sistema	Clasificación ASIL
Sistema de control autónomo	D
Sistema de control de estabilidad (ESC)	C o D
Sistema de frenado electrónico	B o C
Sistema de dirección asistida eléctrica	A o B

1.1.1. Temporal and Spatial Partitioning (TSP)

En los sistemas de tiempo real críticos disponemos de herramientas que nos permiten reforzar la confianza en el sistema.

El aislamiento ante fallos es altamente relevante en sistemas particionados. Esto se refiere a la capacidad de evitar que un fallo se propague afectando a otras particiones. De esta forma se garantiza la seguridad y la integridad del sistema.

La separación temporal consiste en la asignación de una ventana temporal de procesamiento, con el fin de garantizar que se cumplan los requisitos temporales de las diferentes tareas como pueden ser un tiempo de respuesta, latencia o periodicidad. Este aislamiento busca evitar interferencias u otro tipo de conflictos. A las diferentes tareas de un conjunto se les asigna un nivel en función de su importancia, teniendo las de mayor importancia prioridad sobre las de menor.

La separación espacial persigue evitar que los fallos se propaguen, esto lo consigue gracias a que cada partición tiene unas direcciones de memoria asignadas, las cuales son exclusivas para ella y de las que no podrá excederse.

1.2. Sistemas de criticidad mixta (MCS)

Los sistemas de criticidad mixta son aquellos que combinan componentes de diversa criticidad dentro del mismo sistema. Los componentes de alta criticidad son aquellos cuyo fallo puede suponer unas consecuencias más extremas y que de acuerdo con el ASIL se clasificarían como D. Mientras que los componentes de baja criticidad son los que su fallo no tiene un impacto significativo clasificándose como A. Es importante tener en cuenta que en un MCS se debe considerar las interacciones entre los componentes de ambos tipos para evitar posibles fallos y así garantizar la integridad y funcionalidad del sistema.

La complejidad de gestión de un MCS, es uno de los principales problemas a la hora de ser situado en una única plataforma o mismo procesador. Esto es debido a que requieren de una rigurosa planificación y extrema coordinación entre los diferentes componentes para evitar interferencias. Además, al haber diferentes criticidades se plantean nuevos retos en cuanto a la seguridad, ya que hay que mitigar los posibles efectos del colapso de una tarea de menor criticidad. Esto se ve reflejado en los altos costes de certificación, ya que habría que certificar todas las tareas al mismo nivel (Crespo et al., 2014).

1.3. Hipervisores

La implementación de virtualización en una plataforma mediante el uso de hipervisores, puede solventar el problema mencionado con anterioridad. El hipervisor es una capa de software que permite virtualizar el hardware con el fin de poder crear diferentes máquinas virtuales (VM), dentro de las cuales se puede correr un Sistema Operativo (SO) o aplicaciones en *baremetal* totalmente independientes tanto del *host* como del resto de VMs. De esta manera, se pueden implementar diferentes particiones y en cada una asignar los componentes según su criticidad, pudiendo así certificar independientemente las particiones.

Se distinguen dos tipos de hipervisores:

- Hipervisores de tipo 1: este primer tipo se refiere a los hipervisores que se ejecutan directamente sobre el hardware (*baremetal*). Estos hipervisores tienen por tanto, acceso directo al hardware, con lo que tienen un mayor control de los recursos y son más rápidos.
- Hipervisores de tipo 2: en este caso requieren de un SO *host* existente, en el que se ejecutan como una aplicación. Al ser una capa de software por encima de un SO, se disponen de menos recursos para asignar a las particiones y se puede ver afectado el rendimiento por el *host*.

En este artículo se explora una arquitectura particionada de criticidad mixta aplicada destinada a vehículos. Esta arquitectura tiene como objetivo ofrecer una base escalable para el desarrollo de proyectos en el contexto de la movilidad autónoma y segura.

Este artículo se organiza de la siguiente forma:

En la sección 2 se establecen los requisitos que debe cumplir el sistema. La sección 3 presenta el diseño de la arquitectura HW/SW basado en particionado. La validación de la propuesta se hará mediante un caso de uso descrito en la sección 4. Finalmente presentaremos las conclusiones y trabajo futuro en la sección 5.

2. Requisitos

Los siguientes requisitos están relacionados con los necesarios para realizar la conducción autónoma de un vehículo siguiendo los principios de la movilidad inteligente, segura y sostenible.

Clasificaremos los requisitos en:

- Generales (GEN), relacionados con las funciones de un sistema de conducción autónoma (ver tabla 2).
- Plataforma (PLT), relacionados con el hardware y el software necesarios para realizar las funciones anteriores (ver tabla 3).

Los requisitos del prototipo son:

Tabla 2: Requisitos generales

Requisito	Descripción
GEN-010	Se proporcionará un entorno de virtualización para ejecutar múltiples particiones separadas que comparten la misma plataforma de hardware (multinúcleo).
GEN-020	Las particiones estarán aisladas espacial y temporalmente para soportar aplicaciones críticas.
GEN-030	Respaldará las aplicaciones de tiempo real críticas para la seguridad de un sistema de automoción, que deben ser deterministas y predecibles en el tiempo.
GEN-050	Proporcionará tolerancia a fallos a las particiones, lo que significa que la función que implementa el sistema es crucial para la seguridad o, aunque no sea crucial, sigue siendo crítica cuando produce resultados erróneos.
GEN-060	Será compatible con las aplicaciones de los sistemas de conducción autónoma (ADS).

Tabla 3: Requisitos de la plataforma

Requisito	Descripción
PLT-010	El procesador se basará en núcleos ARM por su eficiencia energética, baja disipación de calor, pequeño tamaño y bajo coste.
PLT-020	Deberá soportar herramientas de IA como TensorFlow, Pytorch o Keras para implantar un sistema de Inteligencia Artificial (IA) como componente esencial del vehículo autónomo.
PLT-030	Deberá incluir al menos un sistema de cámara (cámara frontal). Esta cámara se utilizará en la conducción autónoma para identificar marcas viales, señales de tráfico y semáforos.

3. Diseño de la arquitectura HW/SW

En el desarrollo de vehículos autónomos, se requiere de una arquitectura sólida que pueda gestionar tareas críticas de tiempo real, así como otras de menor criticidad. La arquitectura de un vehículo autónomo que proporciona requisitos para un sistema inteligente, seguro y sostenible (S3, *Smart, Safe and Sustainable*), es el mejor ejemplo de la integración de las tecnologías mencionadas anteriormente (Figura 1). La solución ideal para este tipo de vehículos consiste en un sistema particionado de criticidad mixta. La virtualización será una pieza clave ya que permite que las diferentes particiones puedan ejecutarse en sistemas multiprocesador, aportando un mayor rendimiento, fiabilidad y robustez.

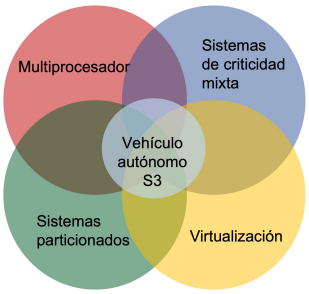

Figura 1: Integración en el vehículo S3

El primer paso en el diseño de la arquitectura es la elección del hipervisor. En este caso se ha seleccionado Kernel-based Virtual Machine (KVM) que se encargará de la virtualización asistida por hardware y de la gestión de las VM; el software QEMU emulará los dispositivos necesarios para ejecutar los SO en las VM. Tener en cuenta sus características y limitaciones es muy importante a la hora del desarrollo de la arquitectura.

La arquitectura básica HW/SW que se ha desarrollado consta de las siguientes particiones (Figura 2):

- Dos particiones dedicadas a conducción autónoma:
  - El *host* de nuestro sistema es un Linux modificado para incluir los drivers necesarios para poder trabajar con los diferentes periféricos de la plataforma. A este Linux base, se le ha añadido el parche de kernel PREEMT-RT, lo cual permite realizar cambios de contexto, mejorando la latencia. Por otro lado, el modulo de kernel KVM se ha añadido para permitir la virtualización basada en hardware. KVM aprovecha la capacidad del kernel de Linux para aislar y administrar recursos, lo que facilita la virtualización eficiente de los componentes del sistema.

    Sobre este *host* se ejecutarán las aplicaciones de mayor criticidad, en este caso la encargada del control del vehículo. Esto se debe a que en caso de fallo de esta aplicación se produciría un accidente del vehículo, debido a que realiza el control de tanto los actuadores como de los sensores.
  - La segunda partición básica, hace uso de un Linux como SO *guest*, para ejecutar la aplicación de reconocimiento de la vía para el guiado del vehículo dentro del carril. Se ha elegido Linux como SO dada la gran disponibilidad de drivers que posee, así como de herramientas adaptables para visión por computador.

    Esta funcionalidad se ha asignado a una partición dedicada ya que se ve penalizada por el manejo de imágenes, que requieren un alto tiempo de cómputo. Lo que podría provocar interferencias en la partición *host* con la aplicación de control del vehículo.
- Tantas particiones como sean necesarias para implementar funcionalidades de la carga del sistema. En la Figura 2 se representa como la partición *n* (VM n) que en este caso se ha creado para implementar el reconocimiento de señales de tráfico.

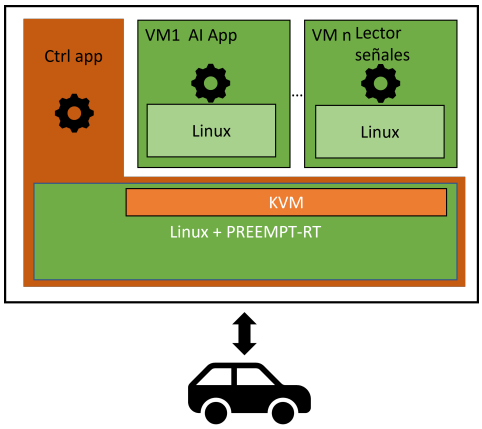

Figura 2: Arquitectura particionada propuesta

Utilizar un sistema particionado nos permite confiar en su escalabilidad para introducir nuevas particiones según requisitos de futuras implementaciones. También permite el desarrollo independiente de cada partición por equipos distintos usando SO diferentes e incluso diferentes lenguajes de programación. Debido a que las nuevas particiones serán dependientes de las características funcionales del proyecto, las características de las nuevas particiones serán de elección del desarrollador.

4. Caso de uso

La validación de la arquitectura propuesta se ha realizado implementando un prototipo de un autobús autónomo. Para ello

se ha seleccionado una plataforma de acuerdo con los requisitos. La arquitectura software que incorpora deriva de la arquitectura propuesta anteriormente.

4.1. Plataforma

La plataforma con la que se trabaja es el kit Jetracer Pro (Figura 3), la cual se basa en un modelo Ackerman. Está impulsada por un motor de alta velocidad con tracción en las 4 ruedas, ya que cuenta con diferenciales en el eje anterior y posterior. El control de la dirección se realiza a través de un servo. Posee además, un premontaje para poder incorporar un sensor óptico para visión artificial.

En cuanto a la placa de desarrollo se disponen de varias opciones para el montaje sobre esta plataforma, como puede ser la Raspberry Pi o la Nvidia Jetson Nano, siendo esta última la seleccionada.

La Nvidia Jetson Nano, cuenta con una CPU ARM A57 de 4 núcleos, una GPU NVIDIA MAXWELL, así como encontramos dos configuraciones en cuanto a RAM, 2 o 4 GB. En función de la RAM seleccionada, se dispondrá de uno o dos puertos MIPI para la conexión de cámaras. Dispone de pines de salida con los que poder hacer uso de: GPIO, I2 C, I2 S, SPI, UART. Debido a la cantidad de particiones que se han planteado para el caso de uso, se optó por utilizar el modelo de 4GB de RAM.

El sensor óptico que incorpora es un sensor Sony, el IMX-219 que ofrece una resolución de 8MP. El modelo en cuestión que se ha seleccionado cuenta con un sensor con un FOV de 160º, y al estar la cámara en una posición no muy elevada permite tener una mejor visión del entorno del vehículo.

Figura 3: JetRacer Pro

4.2. Arquitectura implementada

La arquitectura contará con 4 componentes principales, el control autónomo del vehículo, una aplicación IA, la detección de las señales de tráfico y el control del aforo. Estos componentes se clasifican siguiendo el ASIL de la siguiente manera (ver tabla 4).

Tabla 4: Clasificación de los componentes por criticidad

Sistema	**Clasificación ASIL**
Sistema de control autónomo	D
Aplicación IA	C
Sistema de detección de señales	B o C
Sistema de control de aforo	A

La arquitectura que se ha decidido implementar (Figura 4) consiste en:

- Se parte de la arquitectura básica desarrollada en el punto anterior. En la que encontramos la partición *host* que controla el vehículo y la partición de *guest* que incorpora la aplicación de IA.
- Para el caso de uso, se ha requerido de la implementación de dos particiones de carga útil.
  - La primera partición de carga útil emplea Ubuntu 18.04 como SO *guest*. En este caso, el objetivo es la detección de señales, por lo que se requieren las imágenes que obtiene el vehículo con la cámara frontal.

    Las imágenes se toman en la partición de control y se envían a esta. La transmisión se realiza a través de un canal de comunicación segura, como es una dirección de memoria reservada. Tras la lectura de las imágenes, se procede con un primer análisis, con el objetivo de obtener la posición de señales en la imagen. Esta detección se hace mediante el uso de una versión de Yolov5 entrenada con un conjunto de imágenes llamado GTSDB (Stallkamp et al., 2012), que contiene imágenes de calles en las que las señales están marcadas con su ROI (Region Of Interest). El resultado de este proceso es la obtención de los ROI que contienen posibles señales.

    Estos serán posteriormente examinados por una red CNN que permite el reconocimiento de la señal, devolviendo una matriz de posibilidades. Esta red se ha entrenado con el conjunto de imágenes GTSRB. Finalmente, haciendo uso de una GUI se muestra la imagen con el ROI y el tipo de señal detectada.

    Ambas redes han sido entrenadas con las herramientas que proporciona Python para IA, haciendo uso del *cluster* de cómputo para IA en aplicaciones industriales del que dispone el AI2.
  - La segunda partición que incorpora el autobús tiene como finalidad garantizar la seguridad dentro del mismo, al realizar un conteo del aforo. La configuración de la VM es igual que en el caso anterior. Para la detección de las personas, se adquieren imágenes desde una cámara IP que estaría ubicada en el interior del autobús.

    El recuento de las personas tiene lugar en una aplicación. Esta realiza la obtención de las imágenes, las procesa haciendo uso de la versión ya preentrenada de Yolov5. Una vez se ha analizado, se pueden contar las personas teniendo en cuenta el número de ROIs detectados y mostrar la imagen procesada en la GUI.

    Las imágenes del interior del autobús al contener datos de carácter personal, son cifradas mediante AES para su almacenaje hasta que se proceda con su destrucción. La clave y el IV con los que se cifrarán los datos serán generados al arranque de la partición. Estas imágenes serán inmediatamente cifradas tras haber sido analizadas con Yolo.

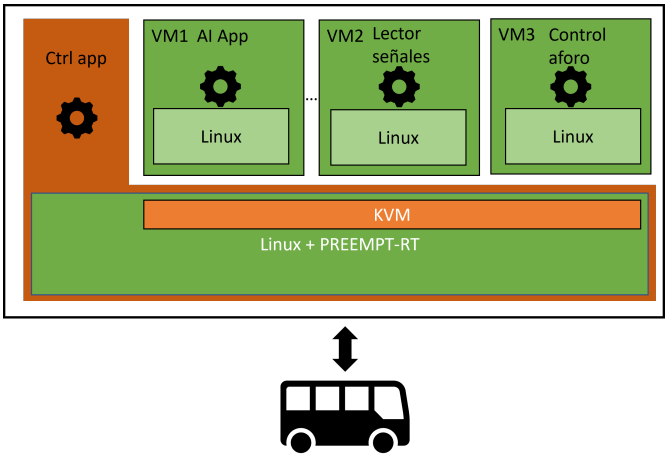

Figura 4: Arquitectura implementada en el prototipo

Las pruebas que se han realizado, han consistido en la puesta en marcha del vehículo ejecutando todas las particiones simultáneamente. En ningún caso, se han apreciado interferencias u otro fallo relacionado exclusivamente al software particionado. El principal inconveniente, ha sido utilizar KVM, que es un hipervisor de tipo 2, ya que las particiones se ejecutan como procesos del sistema anfitrión y por tanto se ejecutan de acuerdo con el planificador CFS (Completely Fair Scheduler). Este asigna el tiempo de CPU en función de los procesos en ejecución, siguiendo el *fair queuing*, lo que puede producir un mayor número de cambios de contexto, un menor rendimiento o la necesidad de ajustar el número de particiones creadas. Por otro lado, en cuanto al procesamiento de imágenes, habría que realizar una optimización de los modelos de detección, con el fin de obtener una mayor fluidez. *Aquí* se encuentra una demostración del vehículo en funcionamiento.

5. Conclusiones

En este artículo se ha propuesto una arquitectura software para vehículos autónomos, la cual ha sido comprobada mediante la puesta en marcha de un prototipo.

De acuerdo a los resultados observados durante la implementación del prototipo, se puede concluir que la arquitectura implementada es robusta y segura funcionalmente. Las diferentes particiones han podido trabajar concurrentemente, sin influir en las otras. Aunque hay aspectos en los que se puede mejorar.

Se sugiere realizar los siguientes cambios en cuanto a la implementación. El hipervisor KVM es bastante limitado en cuanto a requisitos temporales se refiere, por ello, no se puede realizar una planificación predecible, esto se podría solventar haciendo uso de otros *schedulers* como podría ser IRMOS o RESCH (Kato et al., 2009). Otra posible solución para mejorar el rendimiento del sistema, sería cambiar el hipervisor, pasando a utilizar un hipervisor de tipo 1 como podría ser XtratuM (Masmano et al., 2009), aunque en este caso requiere realizar el *port* al procesador que incorpora el prototipo. Sin embargo, la mayor diferencia se produce a la hora de controlar la planificación, ya que en XtratuM la planificación de las particiones es cíclico, obteniendo así un sistema acotado y predecible, cumpliendo así con los requisitos de un sistema con TSP.

En el caso de uso desarrollado para comprobar la fiabilidad de la arquitectura se ha observado que la placa presentaba limitaciones. Estas limitaciones eran claras en cuanto al rendimiento del sistema, aunque también afectaban a la escalabilidad, ya que se encontraba cerca del límite viable de uso de memoria principal. Por ello sería conveniente valorar si se han enfocado correctamente la distribución de las tareas en particiones y sus requisitos mínimos, o sí por el contrario, se precisa buscar una solución hardware más potente. Por otro lado, habría que plantear otra plataforma que se adecue más a los modelos que observamos en los vehículos eléctricos, pudiendo así implementar nuevas formas de control (Hidalgo and Huerta, 2021).

Agradecimientos

Este trabajo ha sido financiado parcialmente por el proyecto PLEC2021-007609 financiado por MCIN/ AEI/ 10.13039/501100011033 y por "Unión Europea NextGenerationEU / PRTR" y por el proyecto de I+D+i PID2021-124502OB-C41, financiado por MCIN/ AEI/10.13039/501100011033.

Referencias

Commission, E., European Climate, I., Agency, E. E., 2023. EU road safety : towards "Vision Zero". Publications Office of the European Union.

Crespo, A., Alonso, A., Marcos, M., de la Puente, J. A., Balbastre, P., 2014. Mixed criticality in control systems. IFAC Proceedings Volumes 47 (3), 12261–12271, 19th IFAC World Congress.

Deredempt, M.-H., Rossignol, A., Windsor, J., De-Ferluc, R., Sanmarti, J., Thorn, J., Parisis, P., Quartier, F., Vatrinet, F., Schoofs, T., et al., 2012. Integrated modular avionics for spacecraft software architecture and requirements. DASIA 2012-DAta Systems In Aerospace 701, 3.

Hidalgo, C. E., Marcano, M., Fernández, G., Pérez, J. M., ene. 2020. Maniobras cooperativas aplicadas a vehículos automatizados en entornos virtuales y reales. Revista Iberoamericana de Automática e Informática industrial 17 (1), 56–65.

Hidalgo, H., Huerta, H., abr. 2021. Control por modos deslizantes para vehículo eléctrico con velocidad diferencial. Revista Iberoamericana de Automática e Informática industrial 18 (2), 115–124.

International Organization for Standardization, 2018. ISO 26262: Road Vehicles - Functional Safety. Tech. Rep. ISO 26262:2018, International Organization for Standardization.

International Organization for Standardization, 2020. ISO 21434: Road Vehicles - Cybersecurity Engineering. Tech. Rep. ISO 21434:2020, International Organization for Standardization.

Jayan, J., Srinivasan, G., 04 2019. Autosar based dual core partitioning for power train application of bms, 1–6.

Kato, S., Rajkumar, R. R., Ishikawa, Y., 2009. A loadable real-time scheduler suite for multicore platforms .

Masmano, M., Ripoll, I., Crespo, A., Jean-Jacques, M., 09 2009. Xtratum: a hypervisor for safety critical embedded systems.

SAE International, 2021. Standard j3016_202104 "taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles". April 30.

Savithry, J., Ortega, A. G., Pillai, A. S., Balbastre, P., Crespo, A., 2019. Design of criticality-aware scheduling for advanced driver assistance systems. In: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). pp. 1407–1410.

Simó, J., Balbastre, P., Blanes, J. F., Poza-Luján, J.-L., Guasque, A., 2021. The role of mixed criticality technology in industry 4.0. Electronics 10 (3).

Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C., 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural Networks (0), –.

Uribe-Chavert, P., Posadas-Yagüe, J.-L., Balbastre, P., Poza-Luján, J.-L., dic. 2022. Arquitectura distribuida modular para el control inteligente del tráfico. Revista Iberoamericana de Automática e Informática industrial 20 (1), 56–67.

Wang, J., 2017. Introduction to Real-Time Embedded Systems. John Wiley Sons, Ltd, Ch. 1, pp. 1–15.