CEA Comité Español de Automática

XLVI JORNADAS AUTOMÁTICA CARTAGENA 2025

UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Modelo de tiempo real de un caso de uso de movilidad inteligente sobre hardware MPSoC heterogéneo

Gomez, Iosu[a,*], Diaz de Cerio, Unai[a], Rivas, Juan M.[b], Gutiérrez, J. Javier[b]

[a] *Inteligencia Conectada y Distribuida, Ikerlan Centro de Investigación Tecnológico, Basque Research and Technology Alliance (BRTA), Arrasate-Mondragón, España.*

[b] *Grupo de Ingeniería Software y Tiempo Real, Universidad de Cantabria, Santander, España.*

To cite this article: Gomez, I., Díaz de Cerio, U., Rivas, J. M., Gutiérrez, J. J.. 2025. Real-time model of a smart mobility use case on heterogeneous MPSoC hardware. Jornadas de Automática, 46. https://doi.org/10.17979/ja-cea.2025.46.12173

Resumen

La tendencia actual en aplicaciones industriales evoluciona hacia plataformas heterogéneas que integran múltiples procesadores y aceleradores especializados dentro de un único MPSoC (*MultiProcessor System on Chip*). Muchas de estas aplicaciones presentan requisitos temporales que las actividades deben cumplir mediante la adecuada gestión de aspectos como la concurrencia, la sincronización y el despliegue de esas actividades en los procesadores o elementos de cómputo disponibles en un entorno normalmente distribuido. En este trabajo, se presenta el modelo de tiempo real de una aplicación del ámbito de la movilidad inteligente que ejecuta en un banco de pruebas basado en procesadores MPSoC. El objetivo es proporcionar un modelo detallado y completo de la aplicación que sirva como caso de uso para la investigación y el desarrollo de nuevas técnicas de planificación, análisis y optimización de sistemas de tiempo real basados en plataformas heterogéneas.

Palabras clave: Tiempo real, Modelado, Planificación, Movilidad inteligente, Aplicaciones

Real-time model of a smart mobility use case on heterogeneous MPSoC hardware

Abstract

The current trend in industrial applications is evolving towards heterogeneous platforms that integrate multiple processors and specialized accelerators within a single MPSoC (MultiProcessor System on Chip). Many of these applications present temporal requirements that activities must meet by properly managing aspects such as concurrency, synchronization and deployment of these activities on the available processors or computing elements in a typically distributed environment. In this paper, the real-time model of an application in the field of smart mobility running on a testbed based on MPSoC processors is presented. The objective is to provide a detailed and complete model of the application to serve as a use case for the research and development of new techniques for scheduling, analysis and optimization of real-time systems based on heterogeneous platforms.

Keywords: Real-time, Modelling, Scheduling, Smart mobility, Applications

1. Introducción

En el desarrollo de nuevas técnicas de análisis de planificabilidad para sistemas de tiempo real, es importante disponer de casos de uso que permitan no sólo validar la aplicabilidad e identificar las ventajas y limitaciones de estas técnicas, sino también comparar los resultados con enfoques anteriores o futuros. En muchos casos, las técnicas desarrolladas sólo se han aplicado a pruebas sintéticas que, si bien permiten su validación y evaluación, son difíciles o incluso imposibles de reproducir posteriormente. Por ello, aplicaciones reales o casos de uso, como los propuestos en los retos industriales de determinadas conferencias, sirven como herramienta eficaz para este fin, al tiempo que cumplen el objetivo de plantear nuevos retos.

*Autor para correspondencia: iosu.gomez@ikerlan.es

Alineado con el tipo de aplicación propuesta en uno de los retos aludidos (Andreozzi et al., 2022), este trabajo propone el modelado de tiempo real de una aplicación de movilidad inteligente que se ejecuta sobre un sistema distribuido en el que los nodos de procesamiento están compuestos por dos tarjetas Jetson NX Orin y dos Unex OBU 301E. El objetivo es hacer pública una descripción detallada de un caso de uso a través de un modelo parametrizado que incluye:

1. El conjunto de tareas junto con sus tiempos de ejecución (concurrencia).
2. Las interacciones entre tareas (incluyendo los mensajes intercambiados a través de las redes) así como el uso de recursos compartidos (sincronización).
3. El mapeado de tareas a nodos y de mensajes a redes (despliegue).

Este modelo no pretende seguir un metamodelo concreto como los usados por algunas herramientas de análisis u optimización de sistemas de tiempo real, ni tampoco ningún estándar como MARTE (*Modeling and Analysis of Real Time and Embedded systems*) de la OMG (*Object Management Group*) (Object Management Group (OMG), 2019). El objetivo es realizar una descripción genérica en términos de funcionalidad, flujos de ejecución en respuesta a eventos externos o temporizadores, identificación de hilos para planificación, o tiempos de ejecución de las funciones concretas.

Otros aspectos, como el conjunto de requisitos temporales (plazos), la política de planificación utilizada o la asignación de tareas a núcleos específicos dentro de un nodo, quedan abiertos a posibles configuraciones y están fuera del objetivo de este trabajo. La idea es que el modelo publicado pueda ser útil para testear diferentes técnicas de tiempo real como las aplicadas en el trabajo (Altmeyer et al., 2023).

El artículo se organiza como sigue. En el Apartado 2 se describe la arquitectura del caso de uso atendiendo a su funcionalidad y elementos que lo componen. El Apartado 3 se dedica a la descripción del modelo de tiempo real del caso de uso con todos los detalles relevantes. Finalmente, en el Apartado 4 se presentan las conclusiones y la línea a seguir como trabajo futuro.

2. Descripción del caso de uso

El caso de uso utilizado se trata de una maqueta que representa una aplicación de conducción cooperativa entre un vehículo inteligente y la infraestructura donde ambos elementos colaboran para detectar obstáculos en la vía y así poder tomar decisiones de manera segura. La maqueta utiliza diferentes plataformas (hardware y software) tanto para procesamiento de datos y toma de decisiones de conducción, como para la comunicación entre nodos.

La parte de infraestructura (*Road Side Unit*, RSU) consta de una serie de sensores capaces de capturar imágenes RGB y nubes de puntos, una plataforma procesadora Jetson NX Orin (con 8 núcleos y una GPU) para la detección de objetos a partir de los datos recibidos de los sensores, y una unidad Unex para la transmisión de datos con el vehículo mediante comunicación inalámbrica V2X (*vehicle-to-everything*) (Festag, 2015). Con esto, la unidad RSU es capaz de detectar obstáculos en la vía y transmitir esta información al vehículo.

Como vehículo se utiliza la plataforma para conducción Roboracer (RoboRacer Foundation, 2025), una maqueta a escala 1/10 de un coche de carreras. Al igual que en la unidad RSU, el vehículo lleva acoplados diferentes sensores para capturar imágenes RGB y profundidad, y dispone de una plataforma Jetson NX Orin para realizar la detección de objetos y toma de decisiones de conducción, así como de una plataforma Unex para la comunicación con la infraestructura. La información sobre la detección de objetos recibida por la unidad RSU es fusionada con la correspondiente a los objetos detectados por el propio vehículo, mejorando la capacidad de percepción del vehículo y permitiendo una toma de decisiones más segura.

El software del sistema se implementa casi en su totalidad sobre un entorno basado en ROS 2 (*Robot Operative System 2*) (OpenRobotics, 2025), el cual proporciona un marco de programación flexible y modular. Las únicas funciones implementados fuera de este entorno son aquellas integradas en los dispositivos Unex, que son las encargadas de la transmisión y recepción de paquetes mediante V2X. ROS 2 se basa en una arquitectura orientada a componentes denominados nodos, los cuales se componen de funciones mínimas panificables llamados *callbacks*. La planificación y ejecución de estos *callbacks* es gestionada por una entidad denominada ejecutor. Cada instancia del ejecutor, que podría estar compuesto por uno o múltiples hilos, planifica los *callbacks* de los nodos asignados a él en orden de llegada y de manera no expulsable. En el caso de uso presentado, todas las instancias del ejecutor disponen de un solo hilo, por lo que los datos compartidos dentro de cada instancia no necesitan mecanismos adicionales de sincronización.

La Figura 1, muestra la arquitectura de alto nivel del caso de uso presentado. En ella se muestra la funcionalidad básica de la aplicación con el objetivo de proporcionar una visión general del sistema. Cabe destacar que, cada uno de los bloques funcionales que se muestran en la imagen, pueden contener múltiples nodos ROS 2 que se encargan de realizar las funciones específicas. Sin embargo, estos no se detallan explícitamente a este nivel para mantener la claridad del esquema. En la implementación y configuración actuales toda la funcionalidad de ROS 2 se ejecuta en un sistema operativo Linux y no se asignan prioridades, por lo que los *callbacks* se van ejecutando en los núcleos disponibles a criterio del sistema operativo según se van activando.

En primer lugar, las dos cámaras instaladas captan imágenes RGB y la profundidad del entorno. Estos datos son procesados por las plataformas Jetson NX Orin, que realizan la detección de objetos. La información sobre obstáculos detectada por la unidad RSU se envía al vehículo a través de V2X y se fusiona con los objetos detectados localmente por el vehículo. Cuando el sistema detecta un objeto con el que existe riesgo de colisión, se activa el nodo AEB (frenado automático de emergencia). Este nodo toma el control del vehículo, sustituyendo las órdenes de conducción dadas por el operador a través del *gamepad*, evitando así la colisión mediante el envío de órdenes de frenado a los controladores del motor.

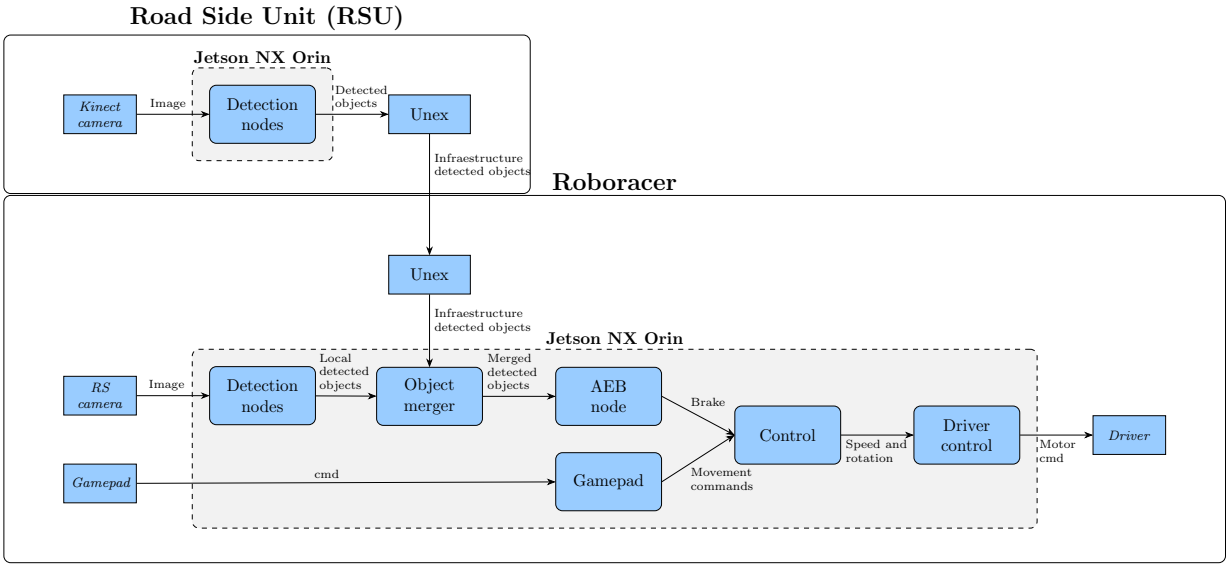

Figura 1: Arquitectura general del caso de uso.

3. Modelo genérico de tiempo real del caso de uso

Para poder modelar el sistema en términos de tiempo real, se ha realizado un análisis exhaustivo de los *callbacks* de ROS 2 y demás funciones involucradas activamente en la ejecución de la aplicación. Se han identificado los eventos que activan cada una de las funciones (publicación de mensajes, temporizadores periódicos o eventos externos de hardware) y se han agrupado las funciones según el hilo que las ejecuta. Esta agrupación es crítica para modelar aplicaciones de ROS 2, ya que los *callbacks* compartiendo el mismo hilo (ejecutor por tanto) se planifican de manera secuencial y no expulsable, lo que impone restricciones de concurrencia y sincronización. Las funciones identificadas se muestran en la Tabla 1, en la que también se muestra a qué hilo pertenecen y sus tiempos de ejecución de peor caso (*Worst-Case Execution Time*, WCET), de mejor caso (*Best-Case Execution Time*, BCET) y promedio (*Average Execution Time*, AET).

En la Figura 2, se representa el modelo genérico obtenido del análisis previo, en el que se reflejan las siguientes características relevantes para tiempo real:

- El conjunto de funciones que componen el sistema, que aparecen representadas como τ_n.
- Las interacciones entre funciones, que dan lugar a flujos de ejecución. La finalización de estos flujos podría tener requisitos temporales.
- Mapeo de funciones a la plataforma hardware. La asignación a núcleos específicos queda abierta a posibles configuraciones.
- Mapeo de funciones a los nodos ROS 2 y a los hilos ejecutores. Las características concretas de la planificación de estos hilos es un aspecto de configuración.

Mediante este modelo genérico se han identificado los diferentes aspectos clave del caso de uso, que se podrán completar cuando se modele haciendo uso de alguno de los metamodelos y herramientas disponibles, y que además deberán tener en cuenta otros aspectos de configuración (planificación y asignación de prioridades a hilos ejecutores, y asignación de ejecutores a núcleos por ejemplo).

En primer lugar, el sistema opera de manera que cada función puede ser disparada por uno o más eventos de entrada y puede generar uno o varios eventos de salida que a su vez pueden activar otras funciones, creando así flujos de ejecución complejos en los que pueden aparecer combinaciones de eventos. Cabe destacar que, algunas funciones como τ_5 y τ_{18}, pueden activarse mediante múltiples eventos de entrada implementando lógicas AND (para activar la función deben haberse generado todos los eventos de entrada) y OR (la función es activada por cualquiera de los eventos de entrada) respectivamente.

Un aspecto relevante del comportamiento de algunos de los nodos ROS 2 modelados, es el desacoplamiento temporal que ocurre cuando *callbacks* encargados de procesar los datos de salida de otras funciones se activan mediante temporizadores, en lugar de por la publicación directa de mensajes por ejemplo. Este desacople da lugar a flujos de eventos independientes, incluso si los *callbacks* residen dentro del mismo nodo o hilo de ejecución.

Respecto a la comunicación V2X, pese a no contemplarse explícitamente en el diagrama mostrado, requiere que se considere el tiempo de transmisión de los mensajes, lo que añade un retraso en la activación de la siguiente función en un flujo dado.

Además, se han identificado los puntos donde diferentes funciones hacen uso de recursos compartidos. Como ROS 2 planifica los *callbacks* de manera secuencial, pese a compartir recursos, estos no harán un uso concurrente de ellos. Por este motivo, en el modelo no se identifican los datos que se comparten dentro de cada ejecutor. Sin embargo, las funciones que se ejecutan sobre los dispositivos Unex, al no ejecutarse sobre ROS 2, hacen uso de recursos compartidos y por lo tanto hay que tenerlos en consideración.

Por último, se ha identificado una particularidad en el flujo encargado del procesamiento de la nube de puntos de la

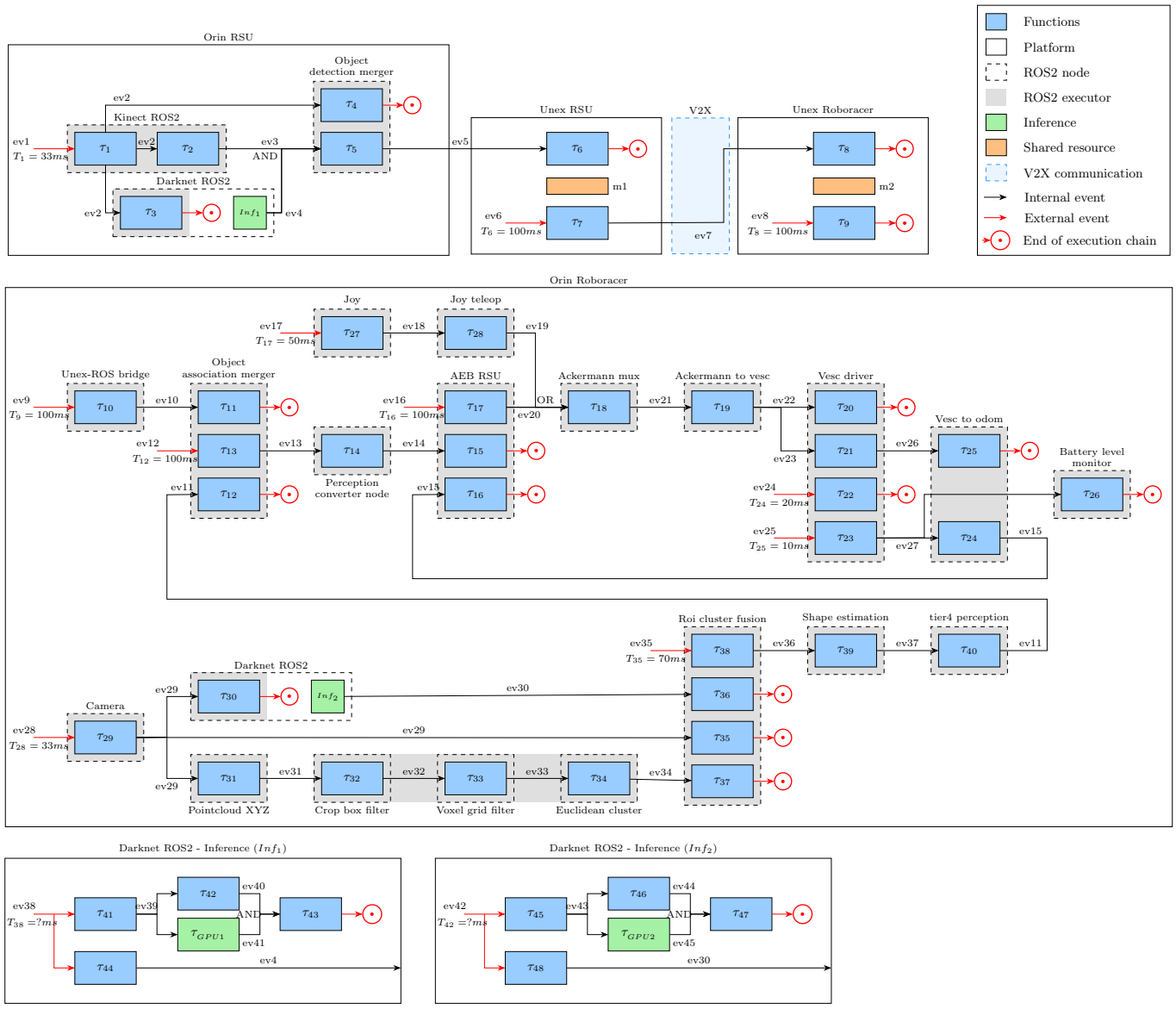

Figura 2: Modelo detallado de los flujos de ejecución del caso de uso.

cámara del Roboracer. La función τ_{29} publica los datos recibidos de los sensores a un ritmo de 30 FPS (cada 33 ms), pero el tiempo de ejecución de parte del flujo que sigue a la función τ_{31} excede con creces ese tiempo. En concreto, las tres funciones τ_{32}–τ_{34} son ejecutadas por el mismo hilo de manera secuencial e indivisible por lo que no se pueden procesar todos los datos que se producen. Por este motivo, la función τ_{37} procesa el último dato disponible. De este modo en la ejecución real, el hilo que procesa las funciones τ_{32}–τ_{34} demanda el uso del 100 % de un núcleo y marca en la práctica el ritmo de procesamiento de los datos de la cámara del Roboracer. Para aumentar la capacidad de procesamiento de estos datos, esta funcionalidad podría ser realizada en la GPU, lo que reduciría sensiblemente su tiempo de ejecución.

3.1. *Modelo del uso de la GPU*

En este apartado se han descrito previamente las actividades del sistema en el nivel ROS 2, es decir, todas las actividades ejecutadas como *callbacks* que conforman el sistema. Sin embargo, hay ciertas actividades que hacen uso de la GPU para acelerar la ejecución de su carga de trabajo. Para modelar este uso de la GPU, se ha realizado un análisis específico con el objetivo de identificar qué nodos hacen uso de este recurso.

En la aplicación analizada, solamente el nodo *Darknet ROS2* (Bjelonic, 2016–2018), que tiene una réplica tanto en el vehículo como en la infraestructura, hace uso de la GPU. Este nodo integra el *framework Darknet* que realiza la inferencia para la detección de objetos utilizando la red neuronal YOLO. El nodo recibe las imágenes mediante *topics* de ROS 2, las transfiere al *framework Darknet* para que realice la inferencia, y finalmente publica los resultados (objetos detectados) en otro *topic* para que puedan ser utilizados por los siguientes nodos en el flujo de ejecución de la aplicación. Con el objetivo de acelerar la ejecución de la inferencia, ésta se realiza descargando la mayor parte del procesamiento en la GPU.

En el modelo detallado de la Figura 2, también se muestra una versión simplificada del comportamiento del nodo *Darknet ROS2* extraído mediante el uso de trazas de ejecución. Es-

Tabla 1: Tiempos de ejecución de las funciones (en ms) y asignación de hilos.

Función	Hilo	WCET	AET	BCET	Función	Hilo	WCET	AET	BCET
τ_1	1	11.454	2.312	0.028	τ_{25}	15	0.172	0.049	0.023
τ_2	1	12.521	6.525	1.635	τ_{26}	16	0.284	0.037	0.005
τ_3	2	3.393	0.509	0.221	τ_{27}	17	1.64	0.152	0.102
τ_4	3	3.392	2.147	0.520	τ_{28}	18	0.028	0.001	0.001
τ_5	3	66.32	44.84	40.126	τ_{29}	19	1.071	0.36	0.038
τ_6	4	3.693	0.864	0.637	τ_{30}	20	5.63	1.657	0.560
m_1	-	1.775	0.29	0.176	τ_{31}	21	11.85	9.28	0.024
τ_7	5	14.558	8.5486	5.139	τ_{32}	22	160.25	100.86	98.94
τ_8	6	7.628	3.702	3.401	τ_{33}	22	393.49	350.62	342.37
m_2	-	0.006	0.004	0.003	τ_{34}	22	7.92	5.78	5.10
τ_9	7	4.453	2.196	1.917	τ_{35}	23	0.144	0.054	0.035
τ_{10}	8	20.14	15.99	0.137	τ_{36}	23	9.46	0.112	0.015
τ_{11}	9	0.302	0.070	0.054	τ_{37}	23	5.54	0.807	0.063
τ_{12}	9	0.112	0.070	0.016	τ_{38}	23	6.29	2.167	1.942
τ_{13}	9	5.09	0.285	0.074	τ_{39}	24	2.29	0.137	0.065
τ_{14}	10	0.016	0.002	0.001	τ_{40}	25	0.189	0.137	0.065
τ_{15}	11	11.51	0.232	0.159	τ_{41}	26	0.71	0.69	0.58
τ_{16}	11	0.123	0.015	0.001	τ_{42}	27	13.7	13.28	12.8
τ_{17}	11	1.00	0.197	0.005	τ_{43}	28	3.6	2.99	2.6
τ_{18}	12	0.388	0.109	0.046	τ_{44}	29	0.948	0.224	0.1
τ_{19}	13	0.384	0.134	0.086	τ_{45}	30	0.71	0.69	0.58
τ_{20}	14	4.52	0.405	0.259	τ_{46}	31	13.7	13.28	12.8
τ_{21}	14	5.32	0.272	0.182	τ_{47}	32	3.6	2.99	2.6
τ_{22}	14	12.61	0.180	0.029	τ_{48}	33	0.948	0.224	0.1
τ_{23}	14	0.013	0.002	0.001	τ_{GPU1}	-	52.4	28.94	23.3
τ_{24}	15	0.662	0.293	0.003	τ_{GPU2}	-	52.4	28.94	23.3

te nodo emplea internamente múltiples hilos de ejecución para coordinar la adquisición de imágenes, el proceso de inferencia y la comunicación con la GPU. Así pues, aunque la aproximación que se representa no captura fielmente la complejidad de dicho nodo, sí que sirve para completar un modelo para el análisis en el que se identifica la funcionalidad medible y que podría ser ejecutada por uno o varios hilos. La elaboración de un modelo más preciso, que incluya todas las interacciones entre hilos y con las diferentes actividades de la GPU (copias de datos y ejecución), se deja como trabajo futuro.

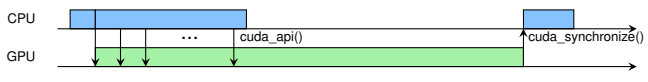

Figura 3: Caracterización del hilo que ejecuta la inferencia en la GPU.

Como se ha mencionado anteriormente, cuando una imagen capturada por la cámara es publicada en un *topic*, el *callback* asociado al nodo *Darknet ROS2* se activa para procesar dicha imagen. Sin embargo, la inferencia de la imagen no es procesada directamente por el *callback*, sino que se almacena para que otro hilo independiente realice la inferencia. Este hilo divide la inferencia de cada imagen en múltiples *Kernels* que son ejecutados en la GPU. Para evitar bloqueos, estos *Kernels* son descargados en la GPU de manera asíncrona, es decir, no se espera a que un *Kernel* complete su ejecución para enviar el siguiente. Una vez descargados todos los *Kernels* y las operaciones de copia de datos necesarias, el hilo espera a que finalice la ejecución encargada a la GPU para procesar y almacenar los resultados de la inferencia. La Figura 3 muestra la secuencia de ejecución de este hilo y su interacción con la GPU. Un tercer hilo es responsable de publicar periódicamente el último resultado de inferencia disponible.

Toda la actividad responsable de la ejecución de la inferencia y publicación de resultados se representa en la Figura 2 englobada en los recuadros Inf_1 e Inf_2, cuya funcionalidad se detalla en la parte inferior de la figura. Las funciones τ_{41} a la τ_{43}, en el caso del recuadro Inf_1, describen el flujo de ejecución del hilo que realiza la inferencia. La función τ_{41} representa la parte de ejecución en la CPU desde la activación del hilo hasta que se lanza la primera operación a la GPU. A partir de este punto la ejecución se bifurca. Por un lado, la función τ_{42} representa la ejecución restante en la CPU durante el cual se lanzan todos los *Kernels* de manera asíncrona hasta que el hilo entra en estado de espera a la finalización de todas la operaciones en la GPU. Por otro lado, τ_{GPU1} representa el tiempo de uso de la GPU desde el envío de la primera operación hasta la finalización de la última. Posteriormente, la función τ_{43} se encarga de procesar los resultados de la inferencia, y se activa cuando tanto la anterior ejecución en la CPU co-

mo las operaciones en la GPU hayan finalizado. Finalmente, τ_{44}, es activada cada vez que una nueva imagen debe ser procesada para publicar el resultado de la imagen anterior. Estas actividades se ejecutan constantemente al máximo ritmo que puedan, por lo que el periodo de activación de ambos hilos vendrá determinado por los tiempos en los que las actividades del flujo se puedan completar. Este periodo queda sin especificar, ya que depende del modelo concreto de tiempo real que se vaya a utilizar. El detalle del recuadro Inf_2 representa idéntico comportamiento.

4. Conclusiones y trabajo futuro

En este trabajo se ha presentado un modelo parametrizado de un caso de uso dentro del ámbito de la movilidad inteligente desplegada sobre una plataforma heterogénea y distribuida. A partir del análisis detallado de la funcionalidad de la aplicación, en su gran mayoría componentes que funcionan sobre el entorno de programación ROS 2, se ha construido un modelo genérico en el que se detallan los aspectos relativos a la concurrencia, sincronización y el despliegue del sistema.

En el caso de uso presentado se muestran los detalles suficientes para poder aplicar técnicas de análisis y optimización de tiempo real, una vez que se hayan establecido los aspectos de configuración relativos a la planificación, despliegue en núcleos concretos, o incluso diferentes tipos de requisitos temporales. Así, el caso de uso modelado puede constituir una referencia para la comparación de técnicas actuales o futuras.

A partir de la descripción del caso de uso presentado en este trabajo, se esta intentando aplicar el modelo MAST (*Modeling and Analysis Suite for Real-Time applications*) (Harbour et al., 2001) para realizar el análisis de tiempo de respuesta sobre diferentes configuraciones de la aplicación, incluyendo el uso de GPUs, para lo cual se aplicará la metodología de modelado y análisis presentada en (Gomez et al., 2024). El software del caso de uso, basado en ROS 2, no ha sido implementado atendiendo a las características de tiempo real del sistema, por lo que en la propia aplicación de esta metodología ya se pueden identificar algunos aspectos a mejorar en la implementación, siendo los más destacables:

- Uso de la GPU en el procesado de las funciones τ_{32}–τ_{34}. Aunque la implementación actual se puede modelar y analizar, el uso de la GPU podría permitir el procesamiento de los datos de la cámara a un ritmo más cercano al que estos se van generando, mejorando la eficacia del sistema. Habría que verificar si este uso de la GPU es compatible con el uso que ya se hace de la misma en el nodo *Darknet ROS2* (podría incrementar los tiempos de ejecución de esta parte en la GPU).
- Posibilidad de ejecutar la funcionalidad asignada a un mismo ejecutor con diferentes hilos, para permitir una asignación optimizada de prioridades. Esto puede requerir el uso de mecanismos de exclusión mutua para sincronización, pero normalmente el uso de estos mecanismos tiene un impacto menor en la planificabilidad.

Es intención de los autores hacer públicos en un repositorio los modelos MAST que se vayan generando para las diferentes configuraciones que se exploren.

Agradecimientos

Este trabajo ha sido financiado parcialmente por MCIN/ AEI /10.13039/501100011033/ FEDER «Una manera de hacer Europa» en los proyectos PID2021-124502OB-C42 y PID2021-124502OB-C44 (PRESECREL).

Referencias

Altmeyer, S., André, E., Dal Zilio, S., Fejoz, L., Harbour, M. G., Graf, S., Gutiérrez, J. J., Henia, R., Le Botlan, D., Lipari, G., Medina, J., Navet, N., Quinton, S., Rivas, J. M., Sun, Y., 2023. From FMTV to WATERS: Lessons Learned from the First Verification Challenge at ECRTS. In: Papadopoulos, A. V. (Ed.), 35th Euromicro Conference on Real-Time Systems (ECRTS 2023). Vol. 262 of Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 19:1–19:18.
DOI: 10.4230/LIPIcs.ECRTS.2023.19

Andreozzi, M., Gabrielli, G., Venu, B., Travaglini, G., 7 2022. Industrial Challenge 2022: A High-Performance Real-Time Case Study on Arm. Leibniz International Proceedings in Informatics, LIPIcs 231.
DOI: 10.4230/LIPIcs.ECRTS.2022.1

Bjelonic, M., 2016–2018. YOLO ROS: Real-time object detection for ROS. https://github.com/leggedrobotics/darknet_ros.

Festag, A., 09 2015. Standards for vehicular communication — from IEEE 802.11p to 5G. e & i Elektrotechnik und Informationstechnik 132.
DOI: 10.1007/s00502-015-0343-0

Gomez, I., Díaz de Cerio, U., Parra, J., Rivas, J. M., Gutiérrez, J. J., Harbour, M. G., 2024. Using MAST for modeling and response-time analysis of real-time applications with GPUs. Journal of Systems Architecture 157, 103300.
DOI: 10.1016/j.sysarc.2024.103300

Harbour, M. G., Gutiérrez, J. J., Palencia, J. C., Drake, J. M., 6 2001. MAST: Modeling and Analysis Suite for Real Time Applications. In: Proceedings of 13th Euromicro Conference on Real-Time Systems. IEEE Computer Society Press, pp. 125–134.

Object Management Group (OMG), 2019. An OMG UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Version 1.2.

OpenRobotics, 2025. ROS2 Humble Documentation.
URL: https://docs.ros.org/en/humble/

RoboRacer Foundation, 2025. RoboRacer.
URL: https://roboracer.ai/