



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Análisis y predicción de datos
obtenidos de un AGV
Documentación Técnica**



Presentado por Gonzalo Burgos de la Hera
en Universidad de Burgos — 5 de julio de 2023
Tutor: Bruno Baruque Zanón y Jesús Enrique
Sierra García

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	v
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	2
A.3. Estudio de viabilidad	8
Apéndice B Especificación de Requisitos	13
B.1. Introducción	13
B.2. Objetivos generales	13
B.3. Catálogo de requisitos	14
B.4. Especificación de requisitos	15
Apéndice C Pruebas de rendimiento de componentes	27
C.1. Estudio del sistema gestor de bases de datos	27
C.2. Estudio del modelo de predicción	37
Apéndice D Especificación de diseño	51
D.1. Introducción	51
D.2. Diseño de datos	51
D.3. Diseño procedimental	54
D.4. Diseño arquitectónico	58
Apéndice E Documentación técnica de programación	61

E.1. Introducción	61
E.2. Estructura de directorios	61
E.3. Manual del programador	63
E.4. Compilación, instalación y ejecución del proyecto	66
E.5. Pruebas del sistema	67
Apéndice F Documentación de usuario	71
F.1. Introducción	71
F.2. Requisitos de usuarios	71
F.3. Instalación	72
F.4. Manual del usuario	72
Apéndice G Publicaciones derivadas del TFG	81
G.1. Publicación	81
Bibliografía	93

Índice de figuras

A.1. Burndown del Sprint 1	3
A.2. Burndown del Sprint 2	4
A.3. Burndown del Sprint 3	5
A.4. Burndown del Sprint 4	5
A.5. Burndown del Sprint 5	6
A.6. Burndown del Sprint 6	7
A.7. Burndown del Sprint 7	8
 B.1. Diagrama de casos de uso	16
 C.1. Procedimiento de la prueba de inserción	29
C.2. Procedimiento de la prueba de latencia	30
C.3. Interfaz Chronograf	37
C.4. Predicción de ARIMA por defecto	41
C.5. Predicción de TCN por defecto	42
C.6. Predicción de N-HiTS por defecto	42
C.7. Predicción de Transformer por defecto	43
C.8. ACF del encoder derecho	44
C.9. PACF del encoder derecho	44
C.10.ACF del encoder derecho después de diferenciar	45
C.11.PACF del encoder derecho después de diferenciar	45
C.12.Predicción de ARIMA con optimización	48
C.13.Predicción de TCN con optimización	48
C.14.Predicción de N-HiTS con optimización	49
C.15.Predicción de Transformer con optimización	49
 D.1. Diagrama de secuencia del servicio “Simulator” generando datos aleatorios	55

D.2. Diagrama de secuencia del servicio “Simulator” generando datos de un CSV	56
D.3. Diagrama de secuencia del servicio “Receiver”	57
D.4. Diagrama de secuencia del servicio “Forecaster”	58
D.5. Diagrama de paquetes	59
D.6. Diagrama de despliegue	60
E.1. Mensaje de carga del servicio “Simulator”	68
E.2. Mensaje de carga del servicio “Receiver”	68
E.3. Registro de CUDA	68
E.4. Mensaje de espera para el escalador	69
E.5. Mensaje de carga del modelo	69
E.6. Mensaje espera para el entrenamiento	69
E.7. Mensaje de entrenamiento del modelo	69
E.8. Mensaje de éxito en la predicción	69
E.9. Predicciones y datos reales	70
F.1. Página de inicio de sesión	75
F.2. Imagen de bienvenida	76
F.3. Información general de la organización	76
F.4. Información de los miembros de la organización	77
F.5. Buckets de la base de datos	77
F.6. Consulta gráfica	78
F.7. Consulta con Flux	78
F.8. Apartado de “Dashboards”	79
F.9. Tablero con las predicciones	79
F.10. Configuración del refresco automático.	80

Índice de tablas

A.1. Asignación temporal	2
A.2. Costes de hardware	8
A.3. Costes de personal	9
A.4. Otros costes	9
A.5. Otros costes	10
A.6. Licencias de las dependencias	11
B.1. CU-1 Gestión de datos.	17
B.2. CU-2 Visualización de datos.	18
B.3. CU-3 Visualización de datos.	19
B.4. CU-4 Predicción de datos.	20
B.5. CU-5 Configuración de entrenamiento.	21
B.6. CU-6 Cargar modelo desde archivo.	22
B.7. CU-7 Simulación de datos.	23
B.8. CU-8 Simulación aleatoria.	24
B.9. CU-9 Simulación desde CSV.	25
B.10.CU-10 Desactivado desde configuración.	26
C.1. Comparativa de información general	33
C.2. Soporte software	33
C.3. Soporte de la comunidad	34
C.4. Comparativa del modelo de datos	34
C.5. Comparativa de información técnica	35
C.6. Resultados de la prueba de rendimiento	36
C.7. MAE de los modelos por defecto	39
C.8. MASE de los modelos por defecto	40
C.9. DTW de los modelos por defecto	40
C.10.Tiempo de ajuste en segundos de los modelos por defecto	40

C.11.Tiempo de predicción en segundos de los modelos por defecto	40
C.12.MAE de los modelos optimizados	46
C.13.MASE de los modelos optimizados	46
C.14.DTW de los modelos optimizados	46
C.15.Tiempo de ajuste en segundos de los modelos optimizados	47
C.16.Tiempo de predicción en segundos de los modelos optimizados	47
D.1. Bucket “AGV”	53
D.2. Bucket “Predictions”	53

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En el ámbito del desarrollo de proyectos de software, es esencial contar con una planificación sólida y realista, así como evaluar la viabilidad económica y legal del proyecto. Estos aspectos son clave para garantizar el éxito a largo plazo, la rentabilidad y la conformidad con las regulaciones aplicables.

La fase de planificación se divide por tanto en los siguientes pasos:

- Planificación temporal: se organizará el trabajo en los sprints correspondientes.
- Estudio de viabilidad: se divide a su vez en otros dos pasos:
 - Viabilidad económica: se explorarán los gastos del desarrollo del proyecto y los posibles beneficios generados por el mismo.
 - Viabilidad legal: se discutirá la licencia del proyecto, la cual depende de las licencias de las dependencias utilizadas.

Para el paso de la planificación temporal, se analizará el tiempo estimado para completar cada uno de las partes del proyecto, y se distribuirán de manera uniforme en todos los sprints según las convenciones marcadas por el método SCRUM.

Para la segunda parte de la planificación, se tendrán en cuenta los gastos de hardware, software, personal y otros (como el precio del alquiler y el internet). Se tendrán también en cuenta el modelo de negocio planteado para generar beneficios.

Para ello, será necesario comprobar que las licencias de las dependencias utilizadas en el proyecto permiten utilizar el modelo de negocio propuesto, por lo que deberá realizarse un análisis de la viabilidad legal del proyecto.

A.2. Planificación temporal

Debido a la flexibilidad que ofrece la metodología ágil SCRUM [1] se decidió adoptar esta metodología desde el principio. Sin embargo, debido a esta misma flexibilidad, no se ha seguido de manera estricta en su totalidad. Por ejemplo, ya que el proyecto solo ha sido desarrollado en solitario, no ha sido necesario la realización de una reunión diaria.

Como marca esta metodología, el trabajo se divide en “sprints”, de unas dos semanas de duración, en la que se planifica lo que se va a realizar en una reunión al comienzo de dicho sprint. Al final de cada sprint, se ha realizado otra reunión (normalmente solapada con la del inicio del siguiente) para revisar el trabajo realizado.

Para el seguimiento de los sprints, inicialmente se empezó a usar la herramienta ZenHub. Sin embargo, por un problema con las licencias, se dejó de usar en mitad del proyecto. Para el control de los sprints, se empezó a usar en su lugar el apartado de “Issues” de GitHub. Para ello, se crea un issue con la tarea a realizar y se asigna una estimación temporal. Dicha asignación temporal puede verse reflejada en la Tabla A.1

Story points	Estimación
1	30 minutos
2	1 hora
3	2 horas
5	3 horas
8	6 horas
13	12 horas
21	1 día
34	3 días
55	1 semana

Tabla A.1: Asignación temporal

Se muestra a continuación la planificación de cada Sprint.

Sprint 1 (2 de marzo de 2023 - 14 de marzo de 2023)

En este Sprint se comenzó el proyecto, quedando definidos los objetivos del mismo. Este Sprint se dedicó principalmente a tareas de investigación: se investigaron diferentes sistemas gestores de bases de datos, herramientas de visualización de datos, ElasticSearch como recomendación de los tutores y Docker; y a tareas de inicialización del repositorio: se importó la plantilla de LaTeX de la memoria y anexos y se desarrollaron las versiones iniciales del servicio “Simulator” y “Receiver”.

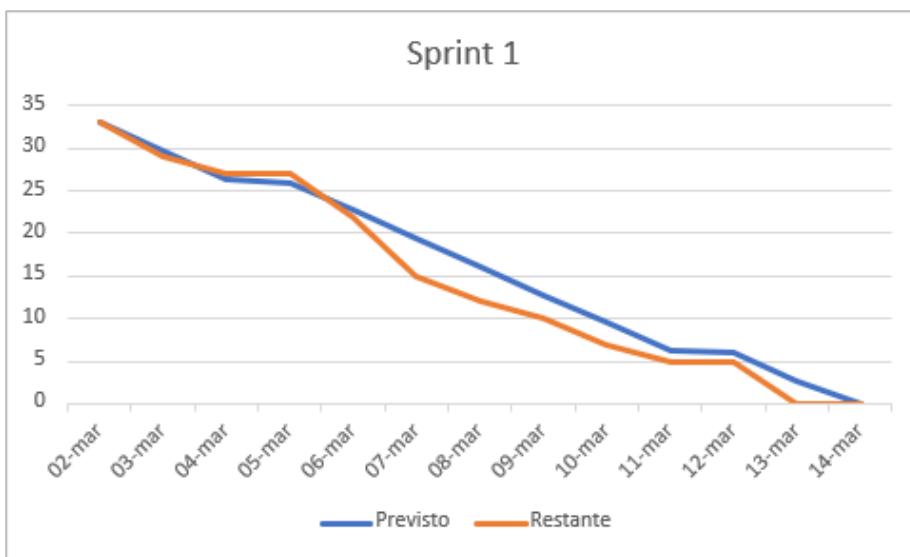


Figura A.1: Burndown del Sprint 1

Sprint 2 (15 de marzo de 2023 - 31 de marzo de 2023)

Este Sprint se dedicó a la mejora de los servicios realizados en el Sprint anterior, así como la redacción de aspectos generales de la memoria. Se integraron también dichos servicios con la herramienta docker compose para ejecutarlos todos de manera coordinada. Por último, en este Sprint se comenzó a escribir lo publicado en el SOCO.

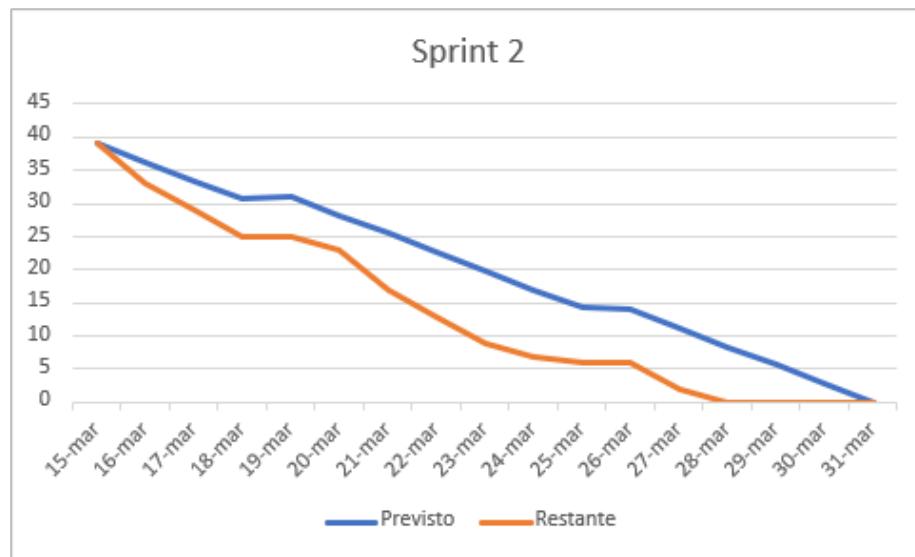


Figura A.2: Burndown del Sprint 2

Sprint 3 (31 de marzo de 2023 - 1 de mayo de 2023)

El Sprint tuvo una duración más larga y una carga de trabajo reducida debido a dos razones. En primer lugar, coincidió con las vacaciones de Semana Santa. En segundo lugar, durante este Sprint, mi enfoque principal estuvo en la redacción del artículo entregado al congreso SOCO 2023, lo que limitó los avances directos en este proyecto.

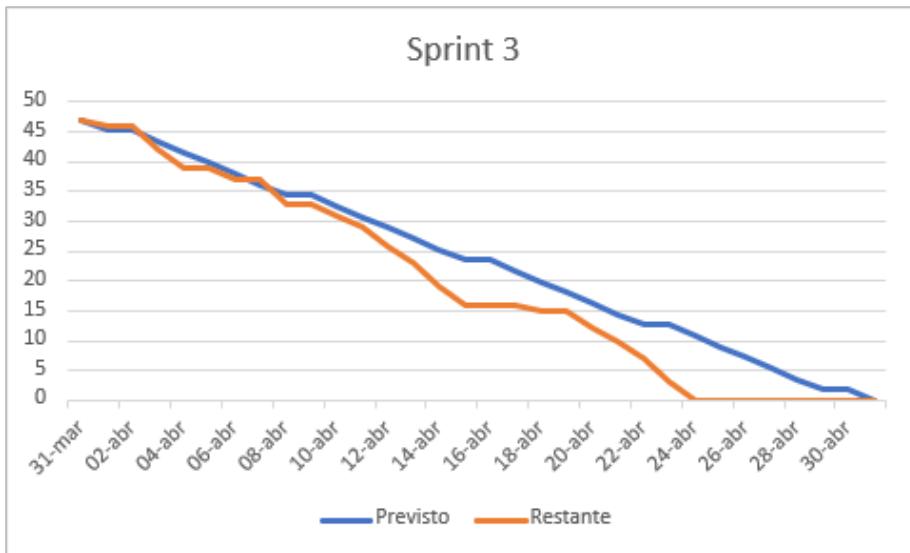


Figura A.3: Burndown del Sprint 3

Sprint 4 (1 de mayo de 2023 - 15 de mayo de 2023)

Al igual que en el anterior Sprint, este se centró también en la redacción del trabajo presentado en el congreso SOCO 2023, por lo que este Sprint se centró en dicha tarea y en realizar avances en la memoria.

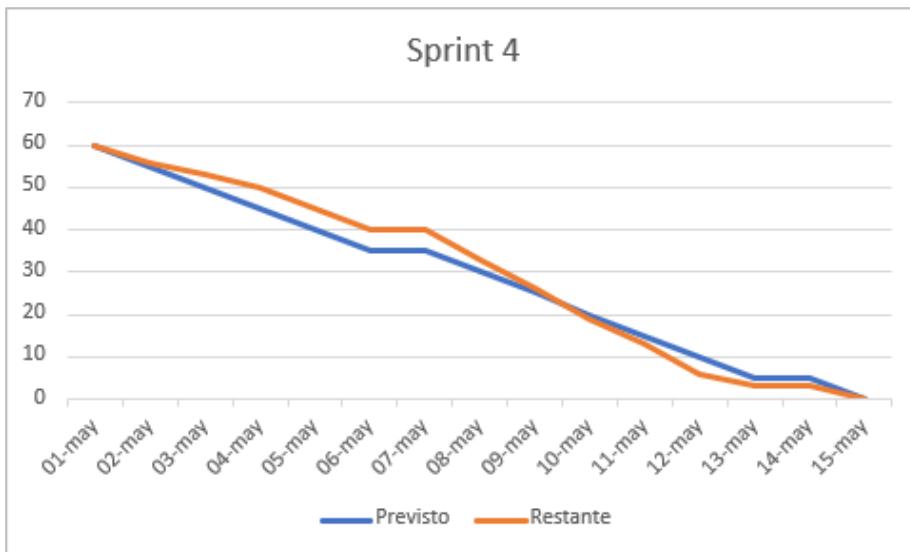


Figura A.4: Burndown del Sprint 4

Sprint 5 (15 de mayo de 2023 - 31 de mayo de 2023)

El trabajo de este Sprint se centró en tres apartados: integrar el artículo anteriormente mencionado en la memoria y anexos del proyecto, investigar herramientas de aprendizaje automático e investigar y optimizar los modelos escogidos en dicha investigación.

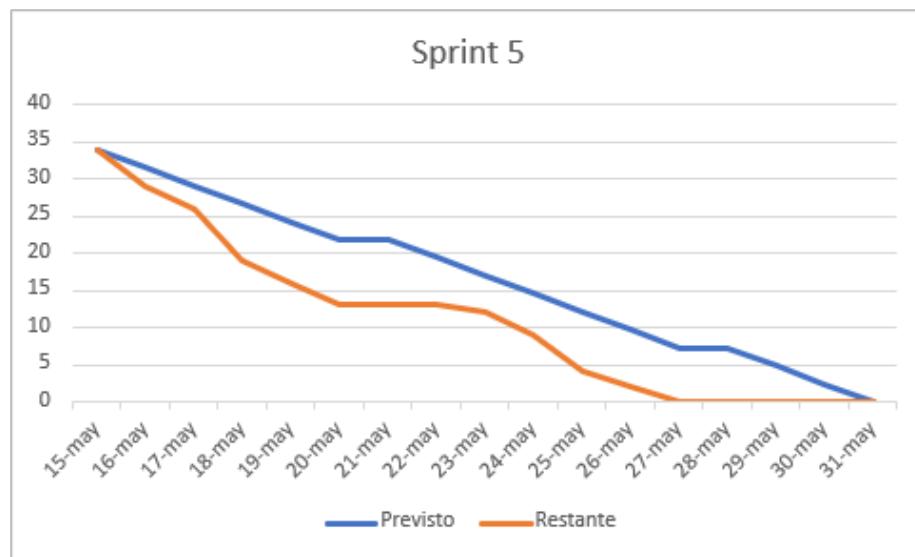


Figura A.5: Burndown del Sprint 5

Sprint 6 (31 de mayo de 2023 - 15 de junio de 2023)

En este Sprint, se desarrolló el servicio “Forecaster” y se mejoró el simulador para que pudiese leer datos desde un archivo CSV. Se continuó con la redacción de la memoria, en especial las partes relacionadas con el sistema gestor de bases de datos, así como mejorar el README del proyecto.

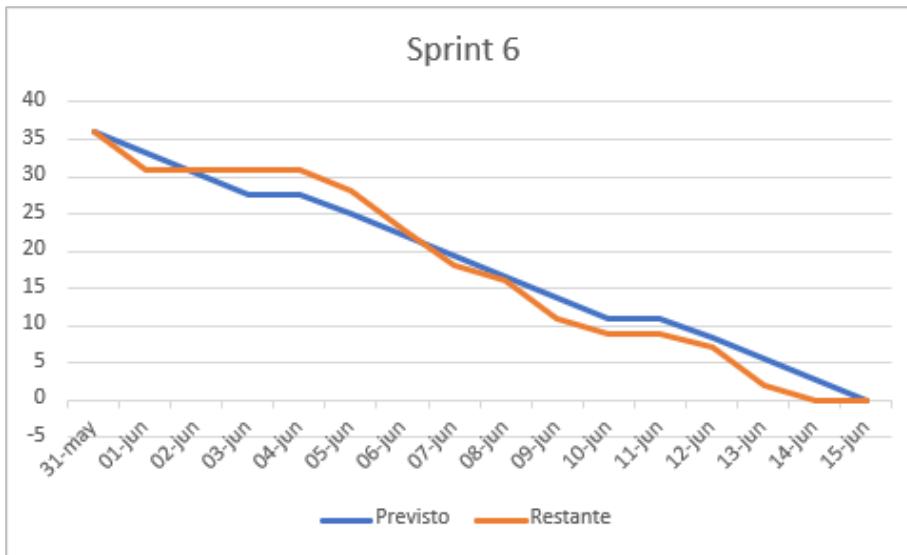


Figura A.6: Burndown del Sprint 6

Sprint 7 (15 de junio de 2023 - 4 de julio de 2023)

Como último Sprint, el trabajo realizado en este Sprint se centró en la mejora del código y de la funcionalidad de los servicios, así como terminar la memoria y anexos. Se integró Code Climate en el repositorio para monitorizar la mantenibilidad del código y Pylint para comprobar que se siguen las convenciones estilísticas. Por último, se añadió la opción de usar GPU o CPU para entrenar el modelo de predicción.

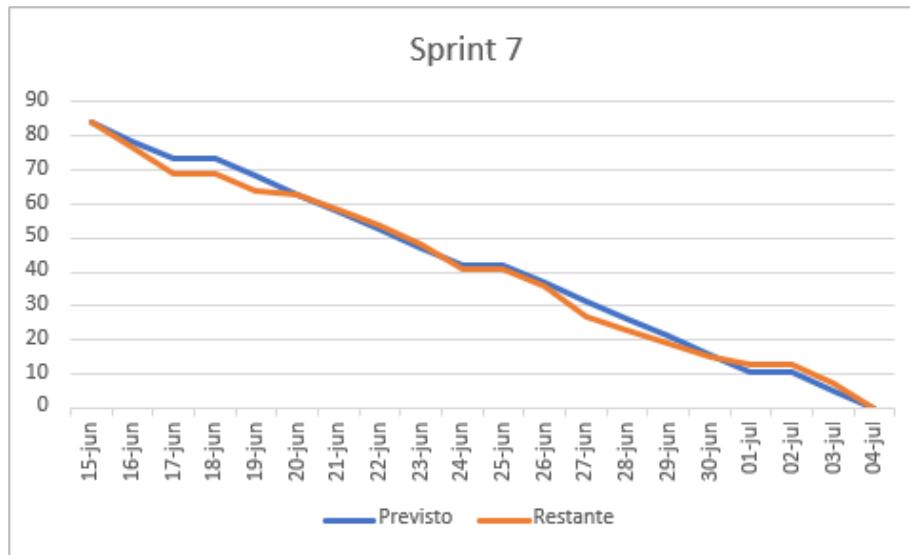


Figura A.7: Burndown del Sprint 7

A.3. Estudio de viabilidad

Viabilidad económica

En este apartado, se analizan los costes y beneficios que podría haber incurrido con el desarrollo de este proyecto en el caso de que se hubiese desarrollado en un entorno profesional.

Costes

Costes materiales En este apartado se analizan todos los costes materiales relacionados con el proyecto, principalmente hardware. Se estima una vida útil del hardware de 4 años, habiéndose utilizado 6 meses para el desarrollo de este proyecto.

Elemento	Coste	Coste amortizado
Ordenador	1500 €	375 €
Periféricos	100 €	25 €
Total	1500 €	400 €

Tabla A.2: Costes de hardware

Costes de software Debido a que el proyecto ha sido desarrollado utilizando únicamente software de código abierto, los costes de software han sido de cero. Convendría, sin embargo, en el supuesto caso de aplicar este proyecto en un entorno industrial real, estudiar si merece la pena utilizar la versión empresarial y de pago de InfluxDB, la base de datos utilizada.

Costes de personal El desarrollo del proyecto ha sido llevado a cabo por un único programador. Se supone que ha sido a jornada completa y que se trata de un desarrollador junior.

Elemento	Coste
Salario bruto anual	18.000 €
Seguridad Social	6.000 €
Total anual	24.000 €
Total 6 meses	12.000 €

Tabla A.3: Costes de personal

Otros costes En este apartado se estudian otros costes como el alquiler, internet, luz, etc.

Elemento	Coste
Alquiler (mensual)	300 €
Internet (mensual)	50 €
Luz (mensual)	50 €
Calefacción (mensual)	40 €
Agua (mensual)	17 €
Total 6 meses	2.742 €

Tabla A.4: Otros costes

Coste total La suma de todos los costes queda reflejada en la siguiente tabla (Tabla A.5)

Elemento	Coste
Hardware	400 €
Software	0 €
Personal	7.173 €
Otros	2.742 €
Total	10.315 €

Tabla A.5: Otros costes

Beneficios

Ya que el sistema desarrollado se distribuye principalmente a empresas, el precio de adquisición del mismo será de 5.000 €. La adquisición de esta licencia incluye la capacidad de ejecutar y modificar este software. Se ofrece también a demás un servicio de soporte continuo con un precio de 500 € al mes.

Viabilidad legal

En este apartado se analizan las licencias del software utilizado, y se detallará la licencia escogida para el software del proyecto, así como de la documentación, imágenes y vídeos utilizados.

Software

Una licencia es “un contrato entre el desarrollador de un software sometido a la propiedad intelectual y a derechos de autor y el usuario, en el cual se definen con precisión los derechos y deberes de ambas partes.” [2] Se muestran en la Tabla A.6 las licencias del software utilizado.

Inicialmente se planteó utilizar la licencia GPLv3 [3], sin embargo, esta licencia obliga a que todo el código del programa y las herramientas utilizadas cumplan esta licencia: “You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged.” Esto sin embargo no es compatible con la licencia NVIDIA Deep Learning Container license [4]: “You may not use the CONTAINER in any manner that would cause it to become subject to an open source software license.”

Dependencia	Versión	Licencia
InfluxDB	2.6.1	MIT
Docker	24.0.2	Docker Subscription Service Agreement
Docker compose	2.16.0	Apache License 2.0
Nvidia-docker	2.13.0	Apache License 2.0
Darts	0.24.0	Apache License 2.0
Python UDP	0.5.10	MIT
Python	3.10	Python Software Foundation
CUDA	11.8	NVIDIA Deep Learning Container license

Tabla A.6: Licencias de las dependencias

Por ello, finalmente se decidió usar la licencia MIT [5] para todo el código del proyecto.

Documentación

Para licenciar la memoria y estos anexos se ha decidido utilizar una licencia “Creative Commons”. Para elegir la más adecuada, se ha utilizado la herramienta “License Chooser” [6]. Finalmente, se ha elegido la licencia “Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)” [7], que permite tanto el uso comercial, como compartir dicha documentación, siempre y cuando se compartan de la misma manera que en este trabajo.

Imágenes y videos

Para licenciar las imágenes y vídeos se ha utilizado la licencia “Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)”, la misma que la de la documentación.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En esta sección de los anexos se especifican los requisitos, tanto funcionales como no funcionales, que el sistema debe de cumplir.

Se incluye en esta sección también la comparativa entre sistemas gestores de bases de datos y modelos de predicción acordes a dichos requisitos.

B.2. Objetivos generales

El proyecto ha de cumplir los siguientes objetivos generales:

1. Desarrollar un sistema capaz de almacenar, monitorizar y predecir los datos enviados por los AGV.
2. Desarrollar los servicios del sistema de la forma más modular posible, de forma que simplifique la modificación de las funcionalidades del sistema.
3. Realizar un estudio de las herramientas a utilizar con el fin de obtener el sistema más eficiente posible.
4. El funcionamiento de los servicios que componen el sistema han de ser configurables a través de archivos de configuración.

B.3. Catálogo de requisitos

Requisitos funcionales

Se muestran a continuación los requisitos obtenidos a partir del análisis de los objetivos generales mencionados anteriormente.

- **RF-1 Gestión de datos:** el sistema tiene que ser capaz de almacenar los datos enviados por el AGV.
 - **RF-1.1 Visualización de datos:** el usuario debe poder ser capaz de visualizar los datos del AGV
 - **RF-1.2 Visualización de predicciones:** el usuario tiene que ser capaz de ver las predicciones realizadas.
- **RF-2 Predicción de datos:** el sistema tiene que poder generar predicciones a partir de los datos almacenados.
 - **RF-2.1 Configuración de entrenamiento:** el usuario debe ser capaz de especificar en un archivo de configuración si se quiere entrenar un nuevo modelo o cargarlo desde un archivo.
 - **RF-2.2 Cargar modelo desde archivo:** el usuario debe ser capaz de especificar en un archivo de configuración el nombre del archivo del modelo a cargar.
- **RF-3 Simulación de datos:** el sistema ha de poder simular datos en caso de no disponer de un AGV.
 - **RF-3.1 Simulación aleatoria:** el simulador tiene que ser capaz de generar datos aleatorios.
 - **RF-3.2 Simulación desde CSV:** el simulador debe poder leer los datos desde un archivo CSV con información real de un AGV.
 - **RF-3.3 Desactivado desde configuración:** el usuario debe ser capaz de activar o desactivar el simulador desde un archivo de configuración.

Requisitos no funcionales

- **RNF-1 Modularidad:** el sistema ha de desarrollarse de forma que los servicios que lo integran sean lo más modulares posible.

- **RNF-2 Escalabilidad:** el sistema ha de estar desarrollado de manera que permita ser escalable en el futuro.
- **RNF-3 Licencias:** todas las herramientas utilizadas deben ser de código abierto.
- **RNF-4 Compatibilidad de hardware:** el sistema ha de soportar aquellos sistemas con una GPU Nvidia con soporte para CUDA 11.8. Para aquellos sistemas que no cumplan este requisito, se ha de poder realizar el entrenamiento de los modelos usando la CPU.
- **RNF-5 Compatibilidad de software:** todas las herramientas software utilizadas han de tener un buen soporte para Python y GNU/Linux.
- **RNF-6 Rendimiento:** el sistema tiene que ser capaz de almacenar los datos del AGV en tiempo real, así como tener un buen rendimiento a la hora de realizar las predicciones.
- **RNF-7 Momento de inserción:** las entradas de la base de datos han de poder insertarse con el timestamp del momento en el que dichos datos fueron generados en el AGV.
- **RNF-8 Precisión temporal** El timestamp de cada entrada debe tener una precisión en el rango de los milisegundos.
- **RNF-9 Precisión:** los datos tienen que poder insertarse en cualquier momento a la base de datos.
- **RNF-10 Predicción:** el modelo de predicción ha de generar resultados aceptables 10 segundos en el futuro.

B.4. Especificación de requisitos

Diagrama de casos de uso

Se muestra a continuación el diagrama de casos de uso (Figura B.1)

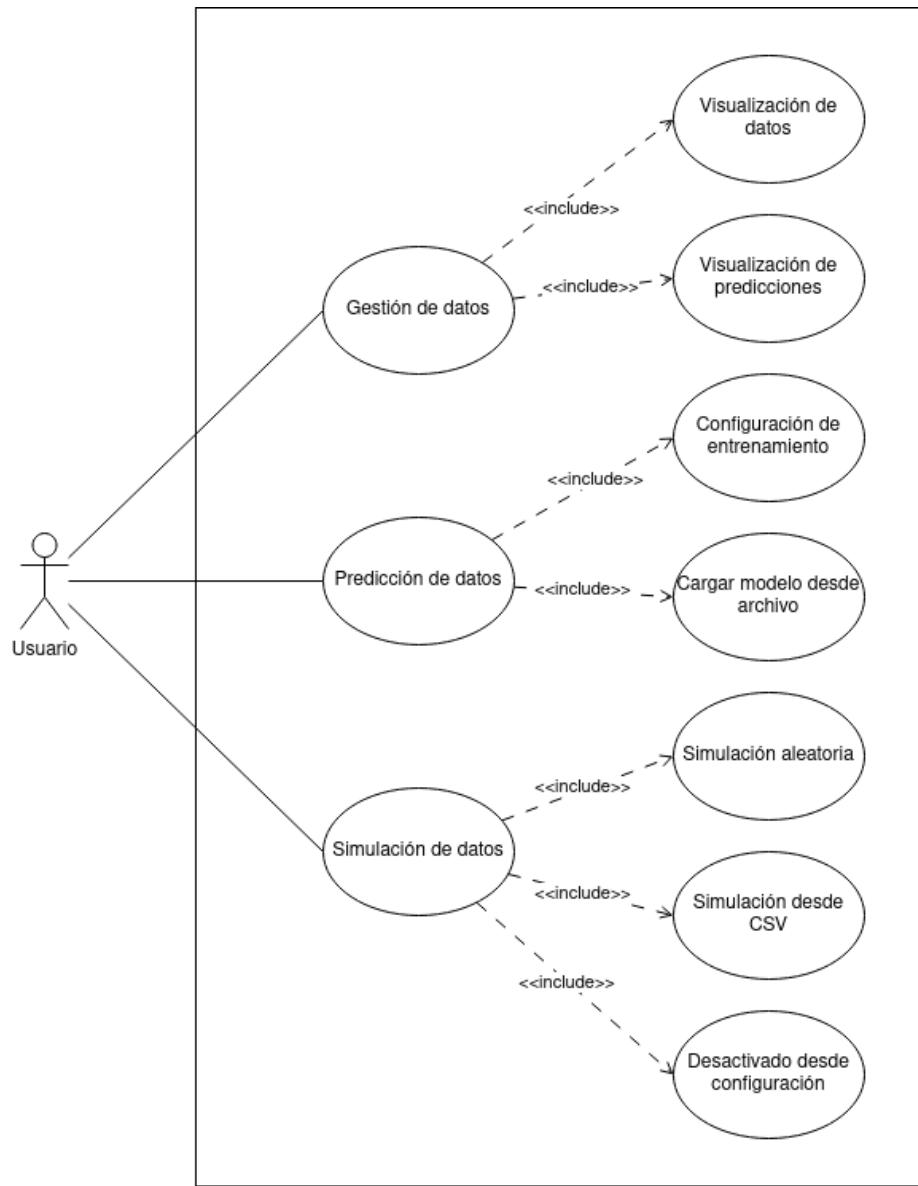


Figura B.1: Diagrama de casos de uso

Casos de uso detallados

A continuación se detallan los diferentes casos de uso:

CU-1	Gestión de datos
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-1, RF-1.1, RF-1.2
Descripción	El sistema tiene que ser capaz de almacenar los datos enviados por el AGV.
Precondición	La base de datos debe estar activa.
Acciones	<ol style="list-style-type: none"> 1. El AGV o el simulador envían datos. 2. El servicio “Receiver” envía esos datos a la base de datos.
Postcondición	Los datos se almacenan.
Excepciones	<ul style="list-style-type: none"> ■ No se puede conectar con la base de datos. ■ El servicio “Receiver” no está activo.
Importancia	Alta

Tabla B.1: CU-1 Gestión de datos.

CU-1	Visualización de datos
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-1.1
Descripción	El usuario tiene que ser capaz de visualizar los datos almacenados.
Precondición	La base de datos debe estar activa.
Acciones	<ol style="list-style-type: none"> 1. El usuario hace la consulta determinada a la base de datos.
Postcondición	Los datos se visualizan.
Excepciones	<ul style="list-style-type: none"> ■ No se puede conectar con la base de datos.
Importancia	Alta

Tabla B.2: CU-2 Visualización de datos.

CU-1	Visualización de predicciones
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-1.2
Descripción	El usuario tiene que ser capaz de visualizar las predicciones realizadas.
Precondiciones	<ol style="list-style-type: none"> 1. La base de datos debe estar activa. 2. El servicio de predicción está activo.
Acciones	<ol style="list-style-type: none"> 1. El usuario hace la consulta determinada a la base de datos.
Postcondición	Las predicciones se visualizan.
Excepciones	<ul style="list-style-type: none"> ■ No se puede conectar con la base de datos. ■ El sistema de predicción no está activo.
Importancia	Alta

Tabla B.3: CU-3 Visualización de datos.

CU-1	Predicción de datos
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-2, RF-2.1, RF-2.2
Descripción	El sistema tiene que poder generar predicciones a partir de los datos almacenados.
Precondiciones	<ol style="list-style-type: none"> 1. La base de datos debe estar activa. 2. El servicio de predicción está activo.
Acciones	<ol style="list-style-type: none"> 1. El servicio de predicciones hace una consulta con los datos necesarios. 2. El servicio de predicciones realiza la predicción. 3. El servicio inserta la predicción en la base de datos.
Postcondición	Las predicciones se almacenan en la base de datos.
Excepciones	<ul style="list-style-type: none"> ■ No se puede conectar con la base de datos. ■ El sistema de predicción no está activo.
Importancia	Alta

Tabla B.4: CU-4 Predicción de datos.

CU-1	Configuración de entrenamiento
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-2.1
Descripción	El usuario debe ser capaz de especificar en un archivo de configuración si se quiere entrenar un nuevo modelo o cargarlo desde un archivo.
Precondiciones	<ol style="list-style-type: none"> 1. El servicio de predicción está inactivo.
Acciones	<ol style="list-style-type: none"> 1. El usuario modifica la configuración.
Postcondición	El servicio de predicciones se inicia con la configuración especificada.
Excepciones	<ul style="list-style-type: none"> ■ Se introduce una configuración inválida.
Importancia	Alta

Tabla B.5: CU-5 Configuración de entrenamiento.

CU-1	Cargar modelo desde archivo
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-2.2
Descripción	El usuario debe ser capaz de especificar en un archivo de configuración el nombre del archivo del modelo a cargar.
Precondiciones	<ol style="list-style-type: none"> 1. El servicio de predicción está inactivo.
Acciones	<ol style="list-style-type: none"> 1. El usuario modifica la configuración.
Postcondición	El servicio de predicciones se inicia con la configuración especificada.
Excepciones	<ul style="list-style-type: none"> ■ Se introduce una configuración inválida.
Importancia	Alta

Tabla B.6: CU-6 Cargar modelo desde archivo.

CU-1	Simulación de datos
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-3, RF-3.1, RF-3.2, RF-3.3
Descripción	El sistema ha de poder simular datos en caso de no disponer de un AGV.
Precondiciones	<ol style="list-style-type: none"> 1. El servicio de simulación está activo en la configuración.
Acciones	<ol style="list-style-type: none"> 1. El simulador genera datos. 2. El servicio “Receiver” recibe los datos.
Postcondición	Los datos se almacenan en la base de datos.
Excepciones	<ul style="list-style-type: none"> ■ El simulador no se inicia.
Importancia	Alta

Tabla B.7: CU-7 Simulación de datos.

CU-1	Simulación aleatoria
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-3.1
Descripción	El simulador tiene que ser capaz de generar datos aleatorios.
Precondiciones	<ol style="list-style-type: none"> 1. El servicio de simulación está activo en la configuración. 2. El servicio de simulación está configurado para generar datos aleatorios.
Acciones	<ol style="list-style-type: none"> 1. El simulador genera datos. 2. El servicio “Receiver” recibe los datos.
Postcondición	Los datos se almacenan en la base de datos.
Excepciones	<ul style="list-style-type: none"> ■ El simulador no se inicia.
Importancia	Alta

Tabla B.8: CU-8 Simulación aleatoria.

CU-1	Simulación desde CSV
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-3.2
Descripción	El simulador debe poder leer los datos desde un archivo CSV con información real de un AGV.
Precondiciones	<ol style="list-style-type: none"> 1. El servicio de simulación está activo en la configuración. 2. El servicio de simulación está configurado para generar datos desde un CSV.
Acciones	<ol style="list-style-type: none"> 1. El simulador genera datos. 2. El servicio “Receiver” recibe los datos.
Postcondición	Los datos se almacenan en la base de datos.
Excepciones	<ul style="list-style-type: none"> ■ El simulador no se inicia. ■ El archivo CSV especificado no existe.
Importancia	Alta

Tabla B.9: CU-9 Simulación desde CSV.

CU-1	Desactivado desde configuración
Versión	1.0
Autor	Gonzalo Burgos de la Hera
Requisitos asociados	RF-3.3
Descripción	El usuario debe ser capaz de activar o desactivar el simulador desde un archivo de configuración.
Precondiciones	<ol style="list-style-type: none"> 1. El servicio de simulación no está iniciado. 2. El servicio de simulación está desactivado en la configuración.
Acciones	<ol style="list-style-type: none"> 1. Se inicia el simulador. 2. Inmediatamente después el simulador termina su ejecución.
Postcondición	El simulador se detiene.
Excepciones	<ul style="list-style-type: none"> ■ Se introduce una configuración inválida.
Importancia	Alta

Tabla B.10: CU-10 Desactivado desde configuración.

Apéndice C

Pruebas de rendimiento de componentes

Teniendo en cuenta los requisitos descritos en el anexo anterior, se realiza una comparación para elegir el sistema gestor de bases de datos y el modelo de predicción que mejor cumplan dichos requisitos.

C.1. Estudio del sistema gestor de bases de datos

Metodología de la comparación

Sistemas de gestión de bases de datos bajo estudio

Lo primero a tener en cuenta es el modelo de la base de datos que se va a utilizar, ya que tendrá una gran importancia a la hora de decidir qué sistema de gestión de bases de datos se va a utilizar. Los siguientes modelos [8] han sido tenidos en cuenta:

- Bases de datos relacionales: en estos sistemas, la información se almacena en relaciones. Una relación, definidas también como tablas, es una colección de tuplas, o filas. Las relaciones se definen por su nombre y un número fijo de atributos, o columnas, con tipos de datos fijos. Estos sistemas respetan las propiedades ACID (Atomicity, Consistency, Isolation y Durability), y tienen operaciones básicas definidas, como la selección, proyección y unión.

- Bases de datos de documentos: la principal característica de estas bases de datos es la organización de los datos de forma libre, sin seguir ningún esquema. Esto significa, por contrario que en las bases de datos relacionales, las entradas no poseen una estructura uniforme, las columnas pueden tener más de un valor e incluso pueden almacenar estructuras anidadas.
- Bases de datos de Clave-valor: son las bases de datos más simples y solo almacenan pares clave-valor. Normalmente no son factibles para aplicaciones complejas, pero normalmente presentan un gran rendimiento debido a su simplicidad.
- Motores de búsqueda: su uso principal es la búsqueda de datos, y son típicamente NoSQL, es decir, que no siguen un modelo relacional.
- Bases de datos de series temporales: estas bases de datos están optimizadas para almacenar series temporales [9]. Aunque son típicamente NoSQL, bases de datos de series temporales relacionales existen.

Cualquiera de estos modelos permiten el manejo de información de series temporales, como la información que recibimos de los AGV. Sin embargo, las bases de datos de series temporales están especializadas en este tipo de trabajos, por lo que las hace perfectas para nuestras necesidades.

Como selección inicial, se han escogido los cinco sistemas gestores de bases de datos de series temporales más temporales según el ranking DB-Engines [10]. Este ranking es utilizado en otros estudios, como [11]. Los sistemas gestores de bases de datos elegidos para comparar son:

- **InfluxDB 2.6.1** Un sistema gestor de bases de datos de series temporales desarrollado por InfluxData. Su uso principal es el almacenamiento y obtención de series temporales, creadas en operaciones de monitoreo de IoT, información de sensores, etc.
- **kdb+ 4.0** Una base de datos de series temporales relacional desarrollado por KX, usado principalmente en negociación bursátil de alta frecuencia, para almacenar y para procesar datos a una alta velocidad.
- **Prometheus 2.43.0** Una base de datos de series temporales usada para el monitoreo de eventos y alarmas, que utiliza un modelo HTTP.
- **Graphite 1.1.10** Una herramienta que almacena, monitoriza y grafica series temporales numéricas.

- **TimescaleDB 2.10.1** Una base de datos de código abierto, desarrollada como complemento a PostgreSQL con el fin de mejorar el rendimiento y análisis para series temporales.

Aunque este ranking solo mide la popularidad y ordena los sistemas en función de atributos sociales, es especialmente útil para soluciones de código abierto, pues esto generalmente significa que está soportada por una comunidad activa con muchos colaboradores involucrados en añadir nueva funcionalidad y corregir errores.

Procedimiento de la comparación

La comparación y el filtrado de los modelos seleccionados ha sido realizado en tres pasos secuenciales:

1. Información general, soporte software y apoyo de la comunidad.
2. Modelo de datos y especificaciones técnicas.
3. Prueba de rendimiento.

En los primeros dos pasos, los sistemas comparados que no cumplen los requisitos especificados en secciones anteriores han sido descartados. Después, con los sistemas restantes, se ha realizado una prueba de rendimiento con el fin de tomar la decisión final. A su vez, esta prueba se divide en otras dos pruebas.

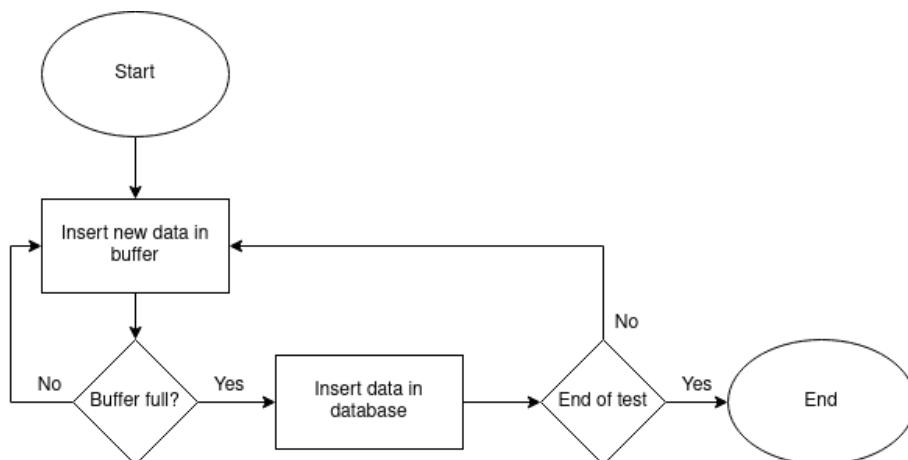


Figura C.1: Procedimiento de la prueba de inserción

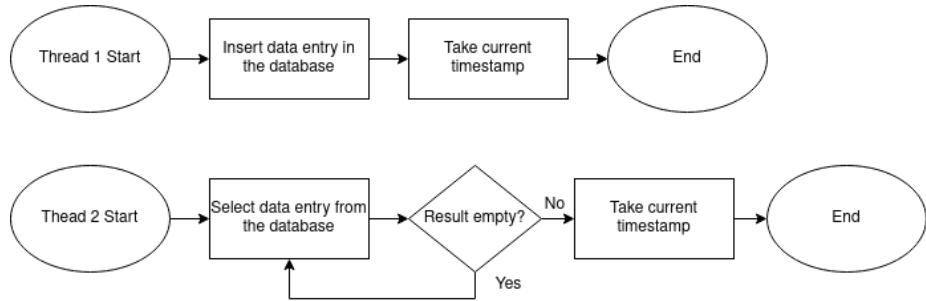


Figura C.2: Procedimiento de la prueba de latencia

El primer test, (Figura C.1) mide el uso de CPU y RAM del sistema cuando se insertan datos de forma masiva, así como el tiempo tomado y el número de inserciones por segundo realizadas. En total, 300.000 entradas son enviadas, formadas por los siguientes campos: timestamp, id del vehículo, batería, velocidad, posición en la coordenada x, posición en la coordenada y, temperatura y voltaje. Para realizar la prueba de manera más realista, los datos se insertan en la base de datos en tandas de 5.000. Para ello se almacenan primero en un buffer controlado por el servicio “Receiver”, ya que de esta manera se obtiene un mejor rendimiento que si se insertasen de uno en uno. Para medir el uso de CPU y de RAM de la misma forma con todos los diferentes gestores, las medidas son tomadas de lo que reporta el estado del contenedor de Docker en el que se ejecutan dichos sistemas.

En el segundo test (Figura C.2) se mide la latencia de inserción. Esto es, el tiempo que tarda en estar disponible una entrada después de su inserción. Para esto, un script de Python, formado por dos hilos, ha sido creado. El primero de esos hilos realiza la inserción en la base de datos y el otro intenta obtener dicha entrada en un bucle. En el momento en el que dicha entrada se obtiene, se anota dicho tiempo y se resta del momento en el que se insertó la entrada. Este test se realiza 200 veces y se hace una media con los resultados.

Cada test se realiza cinco veces para reducir la variabilidad, tomando la media de dichas ejecuciones como valor final.

Métricas de la comparación

A continuación se detallan las métricas utilizadas para realizar la comparación.

Las métricas de información general analizan:

- Organización: que organización o compañía es responsable del desarrollo y mantenimiento del sistema.
- Año de lanzamiento: en que año se lanzó inicialmente.
- Última versión: en que año se lanzó la última versión.
- Licencia: que tipo de licencia tiene el software: código abierto (OSS), o licencia comercial.

El indicador de rendimiento del soporte de software analiza:

- Sistema Operativo: qué sistemas operativos se soportan.
- Soporte para Python: como el sistema está implementado en Python, nos interesa que el sistema tenga buen soporte de este lenguaje.
- Lenguaje de consultas: qué lenguaje de consultas soporta el sistema.
- Plugins para aprendizaje automático: si el sistema soporta complementos que simplifiquen la predicción de nuevos datos.

La comparación del soporte de la comunidad contiene:

- Número de estrellas del repositorio de GitHub: como forma de medir su popularidad.
- Pull requests: número de pull requests enviadas en el último mes.
- Pull requests aceptadas: número de pull requests aceptadas en el último mes.
- Issues: número de issues creados en el último mes.
- Issues cerrados: número de issues cerrados en el último mes.

Estos cuatro últimos campos se usarán para comparar que comunidad es más activa.

El indicador de rendimiento del modelo de datos compara:

- Modelo de datos: qué modelo concreto implementa cada sistema.

- Esquema: un esquema puede verse como una plantilla que define como se almacena la información. Este campo compara si la organización de los datos es estricta (esquema fijo) o no (esquema libre).
- Índices secundarios: si el sistema soporta índices secundarios para un mejor rendimiento de consultas o no.
- Precisión temporal: unidad mínima de tiempo que puede tener una entrada.

El indicador de rendimiento de información técnica se compone de los siguientes campos:

- Scripts de servidor: si el sistema es capaz de ejecutar scripts en el servidor o no.
- Método de partición: si se soportan o no métodos de partición para una mayor escalabilidad.
- Replicación: qué métodos de replicación soporta.
- Consistencia: si la información escrita es consistente o no.
- Conformidad con ACID: si el sistema sigue los principios ACID o no.
- Concurrencia: si el sistema soporta accesos concurrentes o no.
- Durabilidad: si la información es persistente, incluso si falla.
- Método de inserción: si la información se introduce mediante una consulta de inserción o extrayendo los datos de un endpoint de forma periódica.

Por último, el análisis de rendimiento compara:

- Tiempo de inserción: tiempo que se tarda en hacer la prueba de inserción en segundos.
- Tasa de transferencia: número de inserciones por segundo.
- Uso de la CPU: uso de la CPU del contenedor de Docker en el que se ejecuta base de datos durante la primera prueba.
- Uso de RAM: uso de RAM del contenedor de Docker en el que se ejecuta la base de datos durante la primera prueba.

- Latencia: tiempo que tarda en ejecutarse la segunda prueba en milisegundos.

Experimentos y resultados

Sistema	Organización	Lanzamiento	Última versión	Licencia
InfluxDB	InfluxData	2013	2023	MIT
kdb+	Kx Systems	2000	2020	Comercial
Prometheus	-	2015	2023	Apache License 2.0
Graphite	-	2006	2022	Apache License 2.0
TimescaleDB	Timescale	2017	2023	Apache License 2.0

Tabla C.1: Comparativa de información general

Información general (Tabla C.1)

Solo software de código abierto será considerado en este trabajo. Aunque kdb+ tiene una versión de 32 bits, no se usará y no volverá a aparecer en las siguientes comparaciones. Por esta razón también, solo las características de la “Community Edition” de InfluxDB serán utilizadas en dichas comparaciones, y características de la “Enterprise Edition”, que no es de código abierto, no se tendrán en cuenta.

Sistema	OS	Python	Lenguaje de consultas	Plugins ML
InfluxDB	Linux OS x	Sí	Flux e InfluxQL	Loud ML
Prometheus	Linux Windows	Sí	PromQL	No
Graphite	Linux Unix	Sí	No	No
TimescaleDB	OS X Windows	Sí	SQL	No

Tabla C.2: Soporte software

Soporte software (Tabla C.2)

Todos los sistemas soportan Linux y Python, el sistema operativo y lenguaje utilizados. Solo Graphite no tiene un lenguaje de consultas definido, aunque se pueden realizar utilizando lo que llaman Funciones [12]. Únicamente InfluxDB soporta complementos para aprendizaje automático. Otros sistemas como TimescaleDB proveen documentación para realizarlo de forma externa en lenguajes como Python o R [13].

Sistema	Estrellas GitHub	Pull requests	Pull requests aceptadas	Issues	Issues cerrados
InfluxDB	25.2k	26	22	37	13
Prometheus	47.4k	75	53	42	20
Graphite	5.6k	0	0	0	0
TimescaleDB	14.7k	105	83	67	46

Tabla C.3: Soporte de la comunidad

Soporte de la comunidad (Tabla C.3)

Como muestra la tabla, todos los proyectos son muy activos a excepción de Graphite.

Sistema	Modelo	Esquema	Tipado	Índice secundario	Precisión temporal
InfluxDB	Multidimensional	Libre	Numéricos y strings	No	Nanosegundos
Prometheus	Multidimensional	Sí	Numéricos	No	Milisegundos
Graphite	Key-Value	Sí	Numéricos	No	Segundos
TimescaleDB	Relacional	Sí	Tipos SQL	Sí	Nanosegundos

Tabla C.4: Comparativa del modelo de datos

Comparación del modelo de datos (Tabla C.4)

Tanto InfluxDB como Prometheus utilizan un modelo multidimensional. Este modelo puede verse como un modelo clave-valor multidimensional: las entradas de datos están formados por un campo que describe la información almacenada (“nombre de la métrica” para Prometheus y “medida” para

Sistema	InfluxDB	Prometheus	Graphite	TimescaleDB
Scripts del servidor	No	No	No	Sí
Particionamiento	No	Sharding	No	Sí
Replicación	No	Sí	No	Sí
Consistencia	Eventual	No	No	Inmediata
ACID	No	No	No	Sí
Concurrencia	Sí	Sí	Sí	Sí
Durabilidad	Sí	Sí	Sí	Sí
Permisos de usuario	Permisos vía cuentas	No	No	Derechos estándar SQL
Método inserción	Push	Pull	Push	Push

Tabla C.5: Comparativa de información técnica

InfluxDB) y un set de pares clave-valor asociados con un timestamp. La principal diferencia es que las entradas en el modelo de InfluxDB están formados por la medida, un set de etiquetas y un set de valores, en vez de solo un set de pares clave-valor. Estas etiquetas guardan metadatos en forma de cadenas de caracteres, son opcionales y están indexados, mientras que el set de valores guardan la información, no están indexados y están asociados con un timestamp.

Graphite agrega los datos de manera automática en ventanas de un segundo o más. Este comportamiento no es el deseado para nuestras necesidades, ya que se requiere almacenar todos los datos enviados.

Comparativa información técnica (Tabla C.5)

En InfluxDB, la consistencia es eventual. Según la documentación, se prioriza el rendimiento de lectura y escritura antes que una fuerte consistencia. Se asegura, sin embargo, que la información es eventualmente consistente [14].

En bases de datos típicas, la información se inserta a través de algún tipo de consulta desde fuera. Por otro lado, Prometheus escucha a un endpoint en el que se publican los datos y se obtienen en intervalos fijos de tiempo. Esto significa que la inserción solo ocurrirá cuando Prometheus escuche a dicho endpoint, por lo que la información no se puede insertar en cualquier momento. Un método más típico existe, pero no es recomendado y no es posible especificar timestamps [15].

System	InfluxDB	TimescaleDB
Tiempo inserción (s)	24.13	1.16
Tasa inserción (I/s)	12432.66	258620.69
Uso CPU (%)	15.05	55.32
Uso RAM (MB)	219.85	373.73
Latencia (ms)	3.37	0.22

Tabla C.6: Resultados de la prueba de rendimiento

Solo InfluxDB y TimescaleDB cumplen todos los requisitos, ya que Graphite agrega los datos en ventanas de 1 segundo, haciendo imposible obtener datos de un momento concreto, y la forma de inserción de Prometheus le hace incompatible con nuestras necesidades, ya que es necesario poder insertar datos en cualquier momento. Por esto, solo estos dos sistemas se compararán en la prueba de rendimiento.

Análisis del rendimiento

La prueba se realizó con un procesador AMD Ryzen 5 3600 y 32 GB de RAM. Ya que este modelo de CPU tiene 12 hilos, el uso de la CPU puede ser tan alto como 1200 % (Uso CPU = Hilos * 100). Los resultados de esta prueba se muestran en la tabla C.6.

Como se puede observar, InfluxDB es más lento en todos las pruebas que TimescaleDB, pero este último utiliza más recursos del sistema. Si en un futuro es necesario escalar InfluxDB puede ser mejor opción, ya que un menor uso de recursos suele significar un menor coste. Sin embargo, TimescaleDB es más flexible, ya que al estar basado en PostgreSQL tiene todas sus características. Cualquiera de estos dos sistemas puede ser perfectamente usado según los requisitos marcados.

Elección final

Al final, el sistema gestor de bases de datos escogido ha sido InfluxDB. Aunque sea más lento que TimescaleDB, tiene una velocidad lo suficientemente buena, y al consumir menos recursos la hace una opción más barata en caso de que esta solución se aplique comercialmente.

Otro motivo de peso para elegir InfluxDB es que viene por defecto con una interfaz web llamada Chronograf (Figura C.3) en la que se puede

manejar la base de datos, crear gráficas, establecer alarmas, etc. Para realizar esto mismo con TimescaleDB, es necesario utilizar otra herramienta, como podría ser Grafana [16].



Figura C.3: Interfaz Chronograf

C.2. Estudio del modelo de predicción

Metodología

Modelos bajo estudio

Los modelos elegidos para la comparación son los siguientes:

- ARIMA. Modelo estadístico basado en la unión de los modelos Auto-regresivo, Integración y Media móvil.
- TCN. Modelo basado en el uso de redes neuronales convolucionales temporales.
- N-HiTS. Mejora del modelo N-BEATS, que utiliza predicciones futuras y pasadas para entrenarse.
- Transformer Model. Modelo basado en el uso del modelo transformador, normalmente usado para modelos generadores de texto como GPT.

Estos modelos han sido comparados usando las siguientes métricas:

- MAE. Error absoluto medio.
- MASE. Error absoluto medio escalado.
- DTW. Algoritmo “Dynamic Time Warping”.
- Tiempo de entrenamiento. Cuánto tarda el modelo en ajustarse.
- Tiempo de predicción. Cuánto tarda el modelo en realizar una predicción.

Tanto los modelos elegidos como las métricas utilizadas para compararlos son explicados en más detalle en el apartado 3 Conceptos Teóricos en la memoria.

Procedimiento de la comparación

La comparación de los modelos de predicción se ha realizado en dos pasos. La primera comparativa se realiza con los modelos por defecto otorgados por Darts. La segunda, se realiza después de una optimización de los hiperparámetros de cada uno de los modelos.

Para realizar la comparación, excepto en el caso de ARIMA que es un modelo estadístico que no usa redes neuronales, se entrena el modelo durante 150 épocas y se realizan varias predicciones del conjunto de validación. Para ello, se utiliza una ventana deslizante, de forma que todo este conjunto se predecirá en tramos de 200 milisegundos, 1 segundo y 10 segundos. Este procedimiento se realiza cinco veces y se calcula la media de las métricas como valor final.

Esta prueba se repite tres veces por cada modelo, de forma que al final surgen tres comparaciones diferentes;

- Univariante: el modelo toma como entrada valores anteriores de la serie temporal a predecir, y como salida se obtiene solo dicha serie temporal.
- Covariante: el modelo toma como entrada tanto valores anteriores de la serie temporal a predecir, como valores de otras series temporales relacionadas. La salida de este modelo es solo la serie temporal que se quiera predecir.
- Multivariante: el modelo toma como entrada valores anteriores de varias series temporales, prediciendo valores para todas ellas.

Preprocesado

Se han realizado dos operaciones de preprocesado sobre los datos.

1. Diferenciación: se restan los valores de una serie temporal con sus valores actuales. Esto se ha realizado con los valores “encoder_derecho” y “encoder_izquierdo” para obtener la velocidad de cada rueda a partir de la posición de los encoders.
2. Escalado: para aquellos modelos basados en redes neuronales (todos menos ARIMA), es preferible tener valores de los datos entre 0 y 1 para el entrenamiento, por lo que los datos han de ser normalizados antes de entrenar.

Resultados con los modelos por defecto

Como se ha comentado anteriormente, una primera comparativa ha sido realizada con los parámetros por defecto que Darts provee de dichos modelos. Los únicos parámetros especificados han sido la longitud de los datos de entrada y la longitud de los datos de salida en aquellos modelos que lo permitan. Se ha especificado una longitud de entrada de 60 valores (correspondiente a 12 segundos), y una longitud de salida de 10 valores (2 segundos). En los casos con covariables, se ha modificado la longitud de salida para la prueba de predicción de 10 segundos, pues, al no hacer la predicción de dichas covariables solo se puede hacer una predicción a futuro, ya que no se poseen los datos para realizar más.

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	162.909	410.160	1222.112	-	-	-	-	-	-
TCN	2219.023	1985.076	2159.769	1797.013	2492.409	1984.476	1988.296	2255.940	2445.329
N-HiTS	359.733	1102.238	2095.359	1119.604	595.256	1664.665	797.003	821.614	1594.119
Transformer	681.963	708.625	1453.811	432.977	488.078	1070.926	525.755	667.424	1234.222

Tabla C.7: MAE de los modelos por defecto

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	0.692	1.744	5.204	-	-	-	-	-	-
TCN	9.398	8.399	9.141	7.607	10.542	8.388	8.414	9.546	10.340
N-HiTS	1.522	4.668	8.850	4.734	2.523	7.035	3.377	3.476	6.735
Transformer	2.905	3.015	6.148	1.844	2.077	4.525	2.237	2.834	5.223

Tabla C.8: MASE de los modelos por defecto

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	162.909	389.584	1011.495	-	-	-	-	-	-
TCN	2219.023	1969.839	1646.481	1797.013	2485.890	1222.429	1988.296	2227.340	1946.709
N-HiTS	359.733	1083.267	1254.794	1119.604	560.890	1296.753	797.003	785.735	1303.806
Transformer	681.963	668.339	678.005	432.977	467.128	545.544	525.755	639.565	530.672

Tabla C.9: DTW de los modelos por defecto

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	0.378	0.371	0.312	-	-	-	-	-	-
TCN	30.916	30.080	30.818	37.011	36.938	37.353	29.851	28.975	28.900
N-HiTS	28.149	27.630	27.977	34.794	34.812	34.508	28.031	27.987	27.655
Transformer	88.604	90.306	89.935	101.665	102.316	96.463	98.377	88.959	93.865

Tabla C.10: Tiempo de ajuste en segundos de los modelos por defecto

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	0.043	0.036	0.037	-	-	-	-	-	-
TCN	0.013	0.013	0.015	0.018	0.013	0.014	0.017	0.013	0.014
N-HiTS	0.019	0.016	0.018	0.018	0.017	0.018	0.019	0.017	0.018
Transformer	0.020	0.021	0.021	0.020	0.020	0.021	0.021	0.021	0.021

Tabla C.11: Tiempo de predicción en segundos de los modelos por defecto

ARIMA solo soporta modelos univariantes, por lo que no puede ser probado en los ejemplos covariantes y multivariantes. Como conclusiones,

se puede decir que ARIMA genera los mejores resultados a corto plazo. También, al no ser un modelo basado en redes neuronales, no necesita del mismo entrenamiento que el resto de modelos, por lo que el tiempo de entrenamiento es sustancialmente más pequeño. Sin embargo, es el modelo más lento a la hora de predecir nuevos datos, por lo que no es el mejor en caso de que queramos incluir dichas predicciones en el software del AGV.

A continuación se ven los resultados de las predicciones realizadas. Se predicen los próximos 10 segundos utilizando covariables (excepto en el caso de ARIMA que no lo soporta). En negro se ven los valores pasados, en morado los valores predichos y en azul los valores reales para comparar.

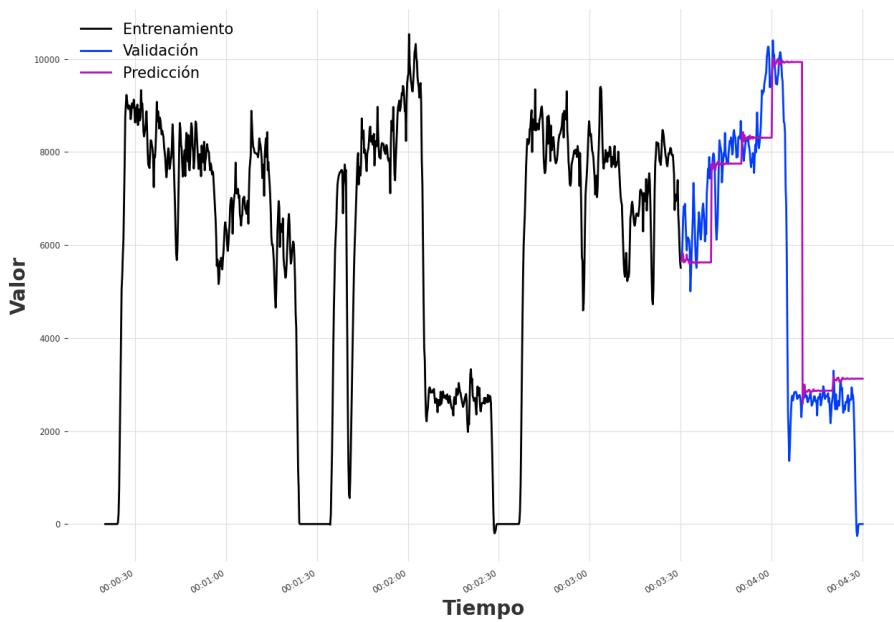


Figura C.4: Predicción de ARIMA por defecto

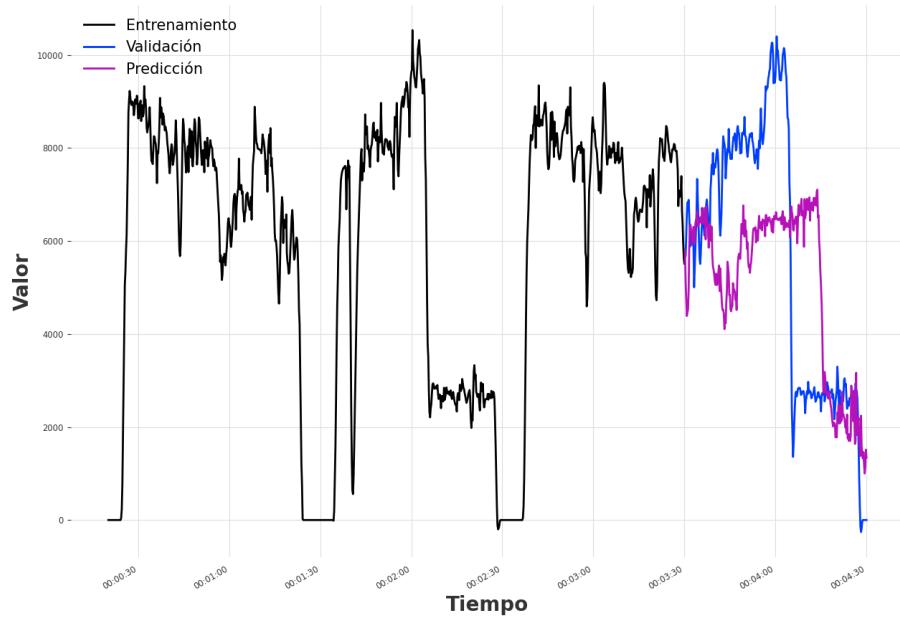


Figura C.5: Predicción de TCN por defecto

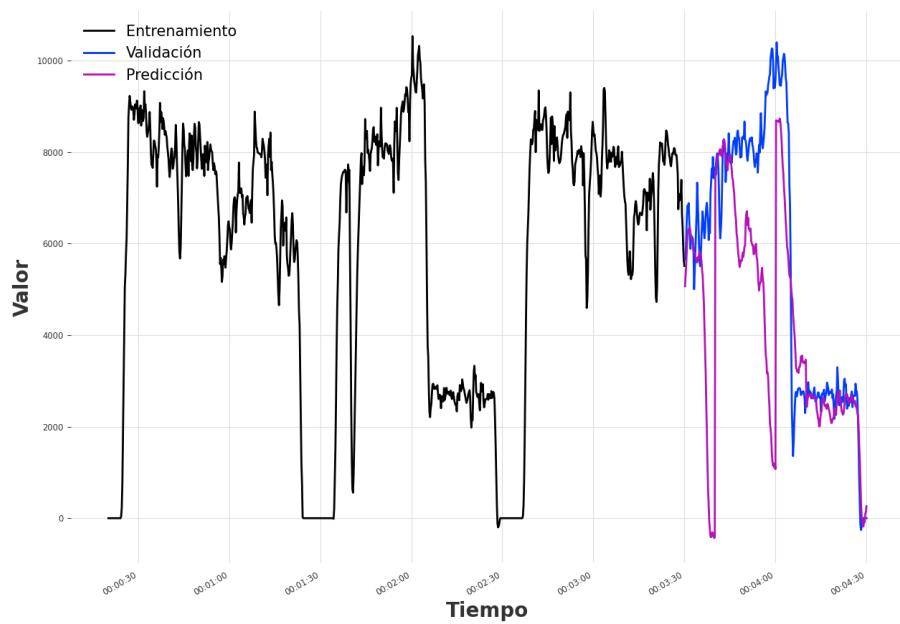


Figura C.6: Predicción de N-HiTS por defecto

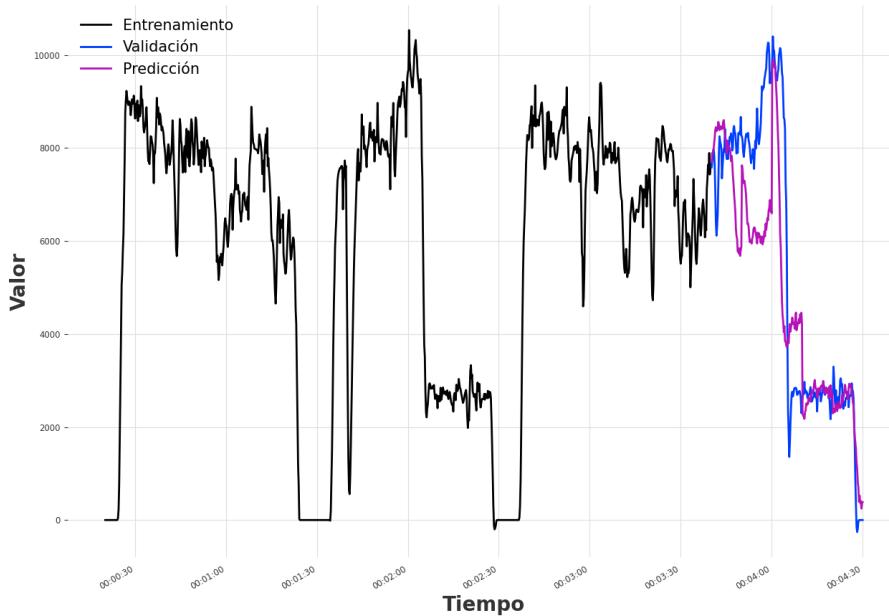


Figura C.7: Predicción de Transformer por defecto

Optimización de los hiperparámetros

Los resultados del apartado anterior son potencialmente mejorables. Para ello, es necesario encontrar los parámetros de los modelos que mejor se ajusten a nuestros datos. Esto, sin embargo, no es una tarea simple, pues existe una enorme cantidad de ellos para los modelos que utilicen redes neuronales (TCN, N-HiTS y Transformer). Para la optimización de dichos modelos se ha utilizado una herramienta llamada Optuna. Esta herramienta prueba cada modelo durante varias horas probando diferentes combinaciones de hiperparámetros de manera automática.

Para el caso de ARIMA la optimización es más simple, pues, como se ha explicado en el apartado de conceptos teóricos solo tiene tres parámetros. Además, dichos parámetros pueden optimizarse de manera analítica, utilizando los gráficos de autocorrelación y autocorrelación parcial. El primero de estos muestra como de relacionado está un valor de la serie temporal con sus anteriores, mientras que el segundo muestra la correlación con el valor inmediatamente anterior al actual.

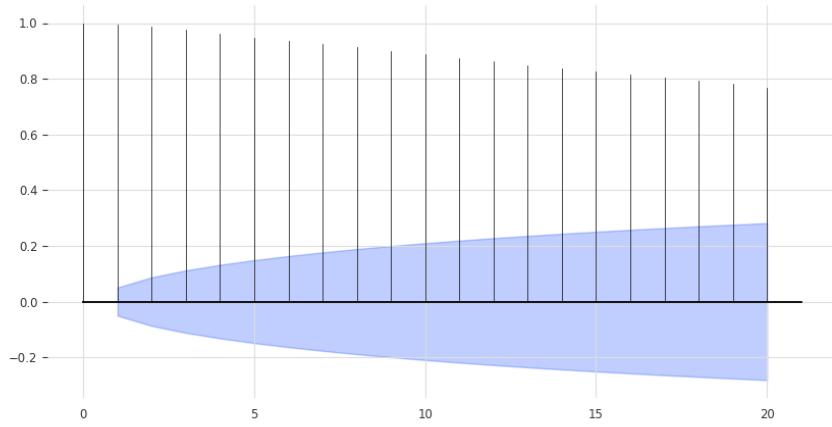


Figura C.8: ACF del encoder derecho

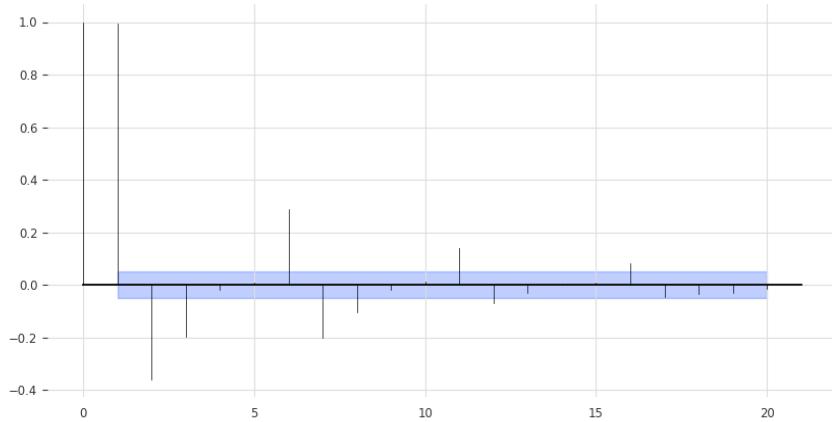


Figura C.9: PACF del encoder derecho

Como podemos ver, el valor del ACF decrementa, pero de manera muy lenta, lo que nos indica que necesitamos por lo menos un orden de diferenciación, por lo que el parámetro d será por lo menos de 1.

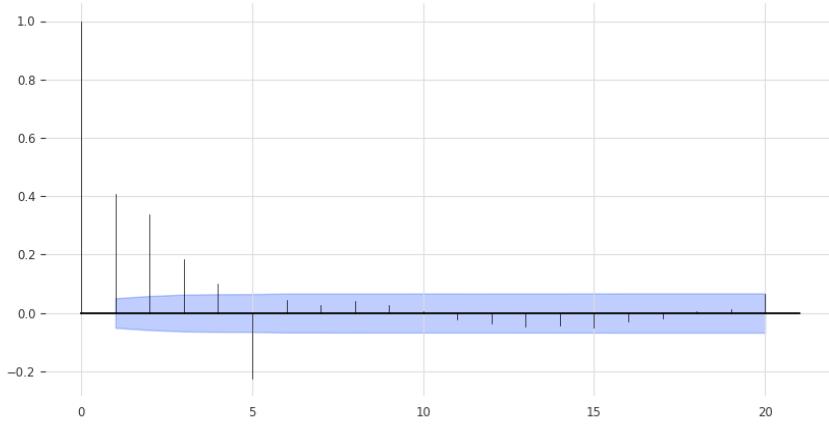


Figura C.10: ACF del encoder derecho después de diferenciar

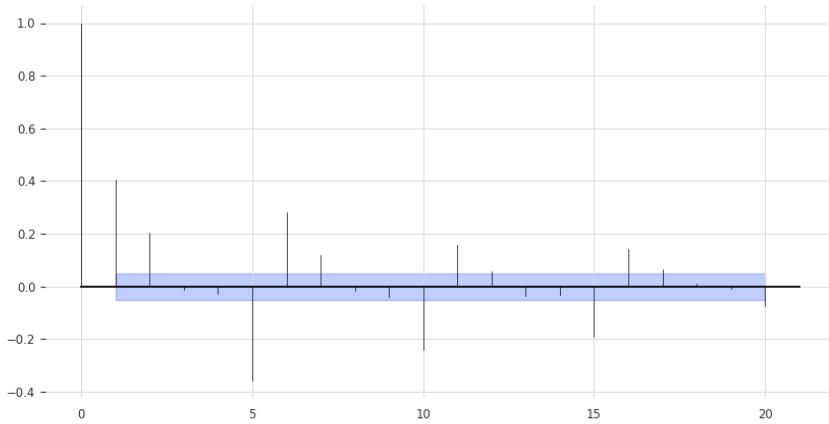


Figura C.11: PACF del encoder derecho después de diferenciar

Se puede ver también como a partir del elemento 16 no se aprecian valores significativos, por lo que el valor del parámetro p será 16. Estas observaciones nos indican a pensar que el modelo es ARIMA($p, d, 0$), ya que se cumplen dichas condiciones:

- El ACF decae de manera exponencial.

- En el PACF, hay un valor significativo a partir del valor p, pero ninguno más adelante.

Por todo esto, el modelo ARIMA que mejor se adapta a nuestro caso es ARIMA(16, 1, 0).

Resultados tras la optimización

Una vez optimizados los hiperparámetros de cada modelo, se vuelve a realizar la comparación.

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	156.587	402.922	1222.479	-	-	-	-	-	-
TCN	1584.048	1611.392	1898.327	1568.804	1596.126	1631.922	1256.715	1974.812	1787.258
N-HiTS	313.865	642.887	1429.619	349.071	396.670	1140.826	413.604	495.936	973.573
Transformer	312.591	442.846	1292.514	450.393	410.858	1119.789	683.099	604.807	1062.630

Tabla C.12: MAE de los modelos optimizados

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	0.665	1.713	5.206	-	-	-	-	-	-
TCN	6.708	6.824	8.032	6.628	6.751	6.904	5.335	8.345	7.553
N-HiTS	1.331	2.724	6.047	1.483	1.688	4.828	1.753	2.106	4.126
Transformer	1.329	1.882	5.490	1.910	1.748	4.746	2.907	2.573	4.488

Tabla C.13: MASE de los modelos optimizados

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	156.587	371.751	1059.898	-	-	-	-	-	-
TCN	1584.048	1587.879	1218.422	1568.804	1574.762	650.782	1256.715	1950.265	910.187
N-HiTS	313.865	629.109	973.335	349.071	375.828	692.528	413.604	460.923	545.182
Transformer	312.591	393.569	723.987	450.393	358.407	515.197	683.099	554.948	707.391

Tabla C.14: DTW de los modelos optimizados

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	0.875	0.823	0.687	-	-	-	-	-	-
TCN	36.238	37.601	35.455	42.710	42.582	42.831	35.664	33.898	34.281
N-HiTS	92.157	96.569	92.305	100.803	99.655	101.466	93.489	90.949	92.417
Transformer	101.097	100.450	107.295	108.456	109.764	114.103	104.710	101.225	110.707

Tabla C.15: Tiempo de ajuste en segundos de los modelos optimizados

Tiempo	Univariante			Covariante			Multivariante		
	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	0.052	0.049	0.048	-	-	-	-	-	-
TCN	0.021	0.016	0.015	0.019	0.014	0.015	0.019	0.015	0.016
N-HiTS	0.025	0.028	0.025	0.025	0.027	0.025	0.027	0.025	0.026
Transformer	0.027	0.025	0.025	0.025	0.029	0.026	0.026	0.027	0.029

Tabla C.16: Tiempo de predicción en segundos de los modelos optimizados

A continuación se ven los resultados de las predicciones realizadas con los modelos optimizados. Al igual que en la anterior prueba, se predicen los próximos 10 segundos utilizando covariables (excepto en el caso de ARIMA que no lo soporta). En negro se ven los valores pasados, en morado los valores predichos y en azul los valores reales para comparar.

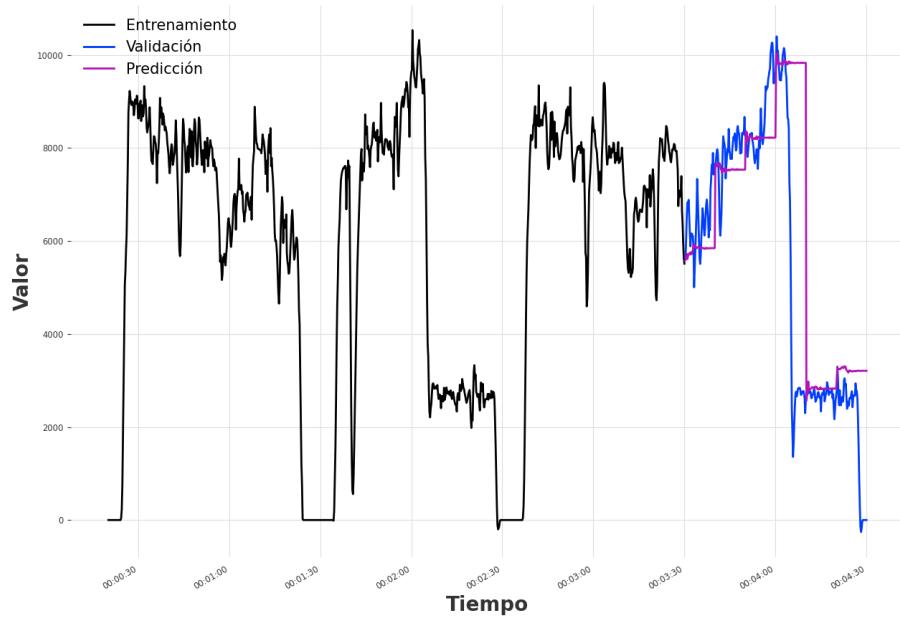


Figura C.12: Predicción de ARIMA con optimización

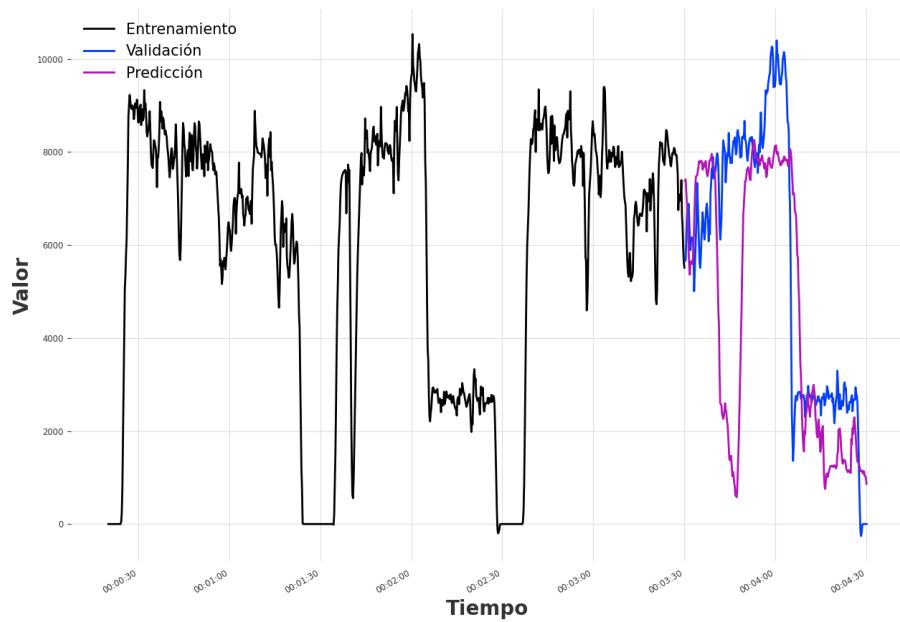


Figura C.13: Predicción de TCN con optimización

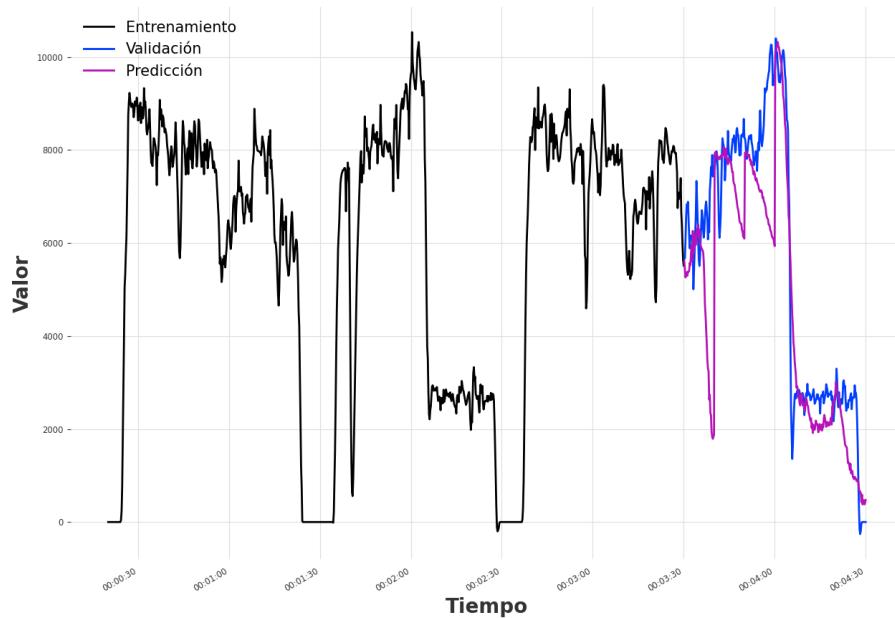


Figura C.14: Predicción de N-HiTS con optimización

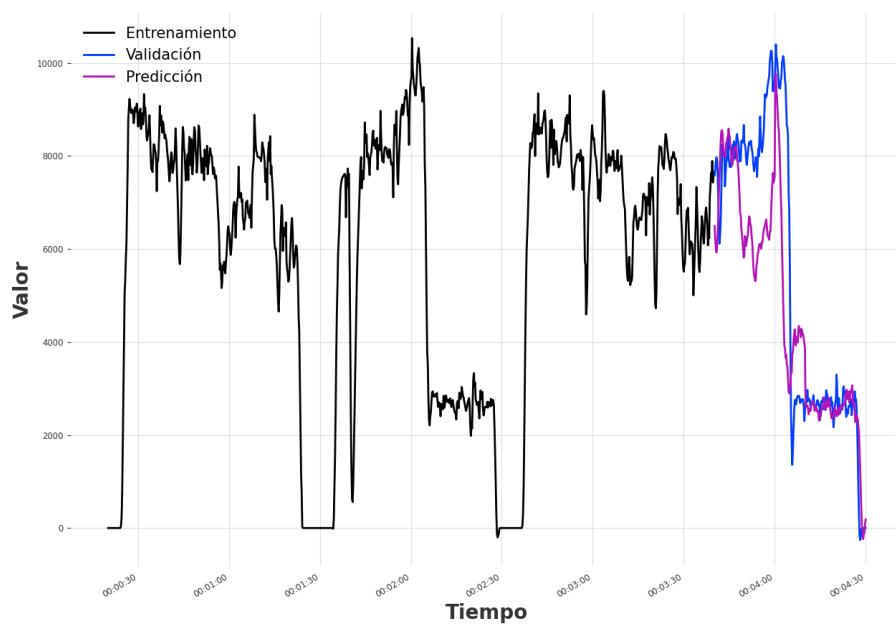


Figura C.15: Predicción de Transformer con optimización

Conclusiones

Como se puede ver, la optimización ha mejorado la mayoría de resultados, a costa de aumentar el tiempo de ajuste y el de predicción. Se sacan por tanto las siguientes conclusiones de las siguientes tablas y figuras:

1. Los mejores modelos en predicciones a 10 segundos en el futuro son el transformador por defecto y con covariables, o N-HiTS con optimizado con multivariables. Ya que trabajar en Darts con covariables es más simple que con multivariables, se escogerá el primero.
2. La cantidad de datos utilizada para el entrenamiento no es suficiente. Se habrían podido conseguir mejores resultados con un conjunto de datos más grande.
3. El tiempo de optimización en el caso del modelo transformador no ha sido suficiente.

Por todo esto, como queremos realizar predicciones a relativamente largo plazo (10 segundos), el modelo escogido finalmente será el Transformer con los hiperparámetros por defecto y con covariables. Aunque existen casos en los que este modelo no es el mejor, tiene un buen balance entre los tiempos de ajuste y predicción y resultados en el resto de métricas.

En el caso de que en un futuro quieran integrarse estas predicciones en el propio software del AGV con el fin de mejorar su rendimiento, habría que estudiar si usar ARIMA por sus buenos resultados a corto plazo, aunque habría que comprobar que los tiempos de predicción sean lo suficientemente buenos.

Apéndice D

Especificación de diseño

D.1. Introducción

En esta sección se detalla la organización y flujo de datos del proyecto, el diseño de los diferentes servicios mencionados en apartados anteriores, y un diseño más en profundidad de toda la arquitectura.

D.2. Diseño de datos

Esta sección se divide en otras dos subsecciones: la primera detallará la organización de los datos en la base de datos, y la segunda detallará el flujo de datos entre los diferentes servicios.

Base de datos

De acuerdo con lo establecido en el anexo B, se ha seleccionado InfluxDB como la base de datos de elección. InfluxDB es una base de datos no relacional orientada a series temporales, lo cual implica que sus características difieren de las bases de datos relacionales convencionales.

El elemento de mayor nivel organizativo en InfluxDB es la organización. Una organización es un entorno de trabajo en el que se definen el resto de elementos de la base de datos: usuarios, “buckets”, tareas, etc.

En InfluxDB, los datos se almacenan en lo que se conoce como “buckets”. La característica principal de estos “buckets” es que tienen un periodo de retención, es decir, el tiempo que perduran los datos almacenados. Así, por

ejemplo, un “bucket” con un periodo de retención de un minuto, los datos insertados hace más de un minuto son automáticamente eliminados.

Dentro de cada “bucket”, los datos están organizados según los siguientes elementos:

1. Time: Al ser InfluxDB una serie de datos de series temporales, este campo cobra una gran importancia. Este campo almacena el tiempo de cada entrada según el estándar RFC3339 [17].
2. Field set: El conjunto de campos, o field set, almacena pares de claves y valore: las claves guardan metadatos y son cadenas de caracteres, y los valores almacenan los datos propiamente dichos.
3. Tag set: el conjunto de etiquetas, o tag set, almacena, al igual que el field set, pares de clave-valor. El tag set, sin embargo, solo almacena metadatos, y tanto la clave como el valor almacenan cadenas de caracteres. Este elemento es opcional, aunque es muy recomendado utilizarlo, pues las etiquetas están indexadas, por lo que las consultas en este tipo de datos son mucho más rápidas.
4. Measurement: la medida, o measurement, actúa como contenedor de los campos anteriores, y sería el equivalente a una tabla en una base de datos relacional.

Con esta información en mente, la organización de la base de datos queda definida de la siguiente manera. Se utilizan dos “buckets”, uno para los datos recibidos por el AGV o por el simulador con un periodo de retención infinito (los datos no se eliminan nunca), y otro para almacenar las predicciones realizadas con un periodo de retención bajo, especificado en un archivo de configuración.

En cada uno de estos “buckets” se almacenan las series temporales siguiendo el esquema de las tablas D.1 y D.2

Elemento	Descripción	
Measurement	AGVDATA	
Tag set	Clave	Valor
	agvid	string
Field set	Clave	Valor
	encoder_derecho	int
	encoder_izquierdo	int
	in.current_l	int
	in.current_h	int
	in.i_medida_bat	int
	in.guide_error	float
	out.set_speed_right	int
	out.set_speed_left	int
	out.display	int

Tabla D.1: Bucket “AGV”

El bucket AGV contiene un tag para la ID del AGV que mande dichos datos. Esto ahora mismo no se utiliza, pues solo se puede simular y predecir un único AGV, pero se incluye por si en el futuro se decide ampliar para poder trabajar con varios vehículos.

La otra tag, “type”, se incluye también para posibles futuros desarrollos. Todos los valores enviados por los vehículos o por el simulador deben tener el valor “value”. En el futuro, en caso de que quieran almacenarse otros datos externos a los vehículos, o datos procesados de los mismos, se han de introducir con otro valor de esta tag diferente.

Elemento	Descripción	
Measurement	pred	
Field set	Clave	Valor
	encoder_derecho	int
	encoder_izquierdo	int

Tabla D.2: Bucket “Predictions”

El bucket de las predicciones es mucho más simple. Simplemente tiene los valores generados por el servicio de predicciones.

Flujo de datos

La comunicación entre servicios ocurre de tres formas diferentes:

- 5G Wifi: Los AGV envían datos al “AGV Coordinator” como cadenas de bytes a este servicio. Como este servicio no está desarrollado como parte de este proyecto, no se detallará más.
- UDP/JSON: Tanto el “AGV Coordinator” como el “Simulator” envían los datos codificados en forma de JSON a través de una conexión UDP. Estos datos son recibidos por el servicio “Receiver”, que los introducirá en la base de datos.
- Database API: El “Receiver” es el encargado de insertar los datos de los AGV o del simulador, utilizando para ello la API de la base de datos a través de consultas.

D.3. Diseño procedimental

Se muestran a continuación los diagramas de secuencia de cada servicio. En naranja se marcan los elementos pertenecientes al servicio “Simulator”, en amarillo los pertenecientes al servicio “Receiver”, en verde el servicio de la base de datos y por último en azul los elementos pertenecientes al servicio “Forecaster”.

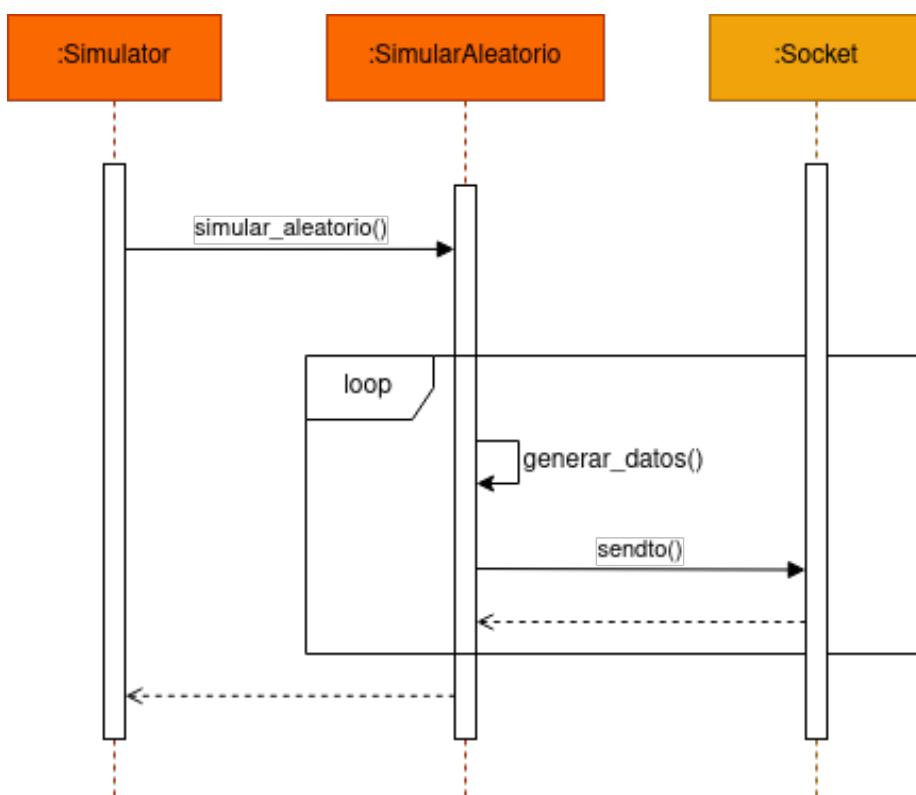


Figura D.1: Diagrama de secuencia del servicio “Simulator” generando datos aleatorios

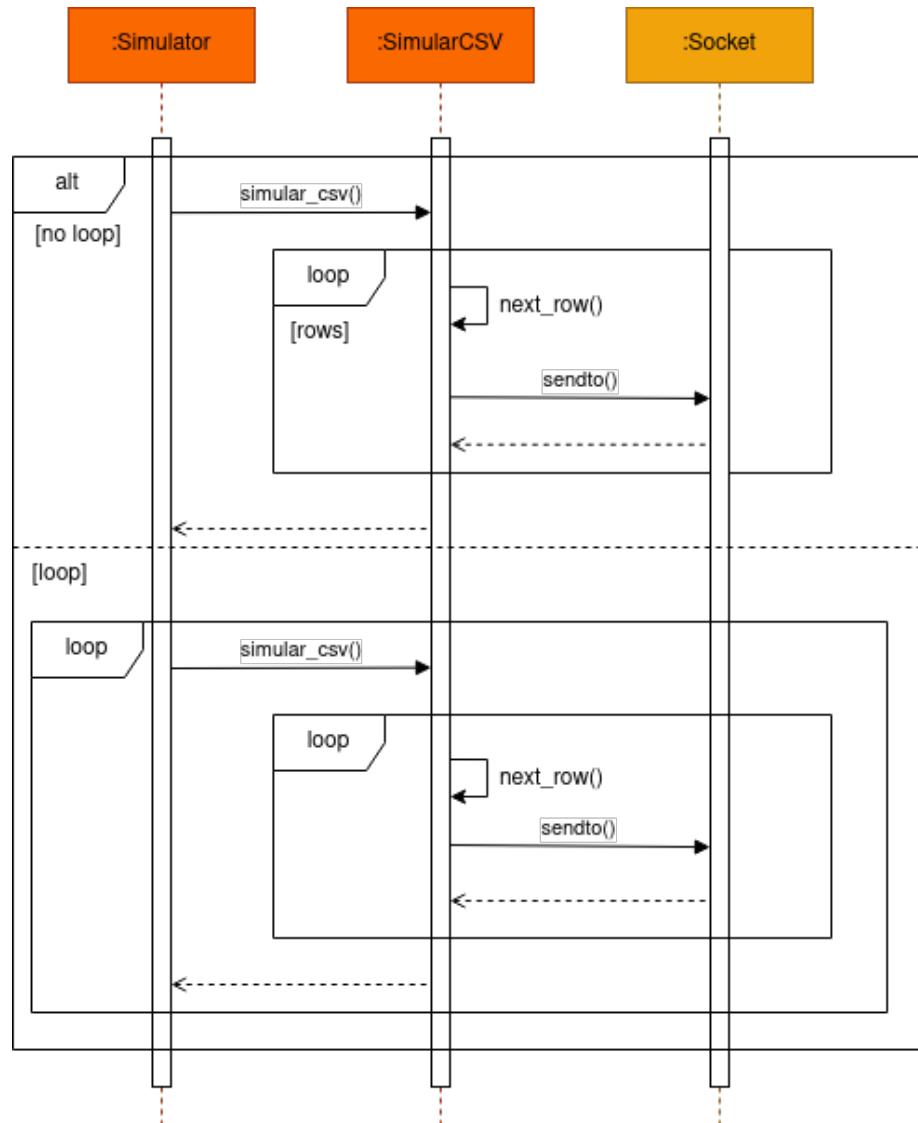


Figura D.2: Diagrama de secuencia del servicio “Simulator” generando datos de un CSV

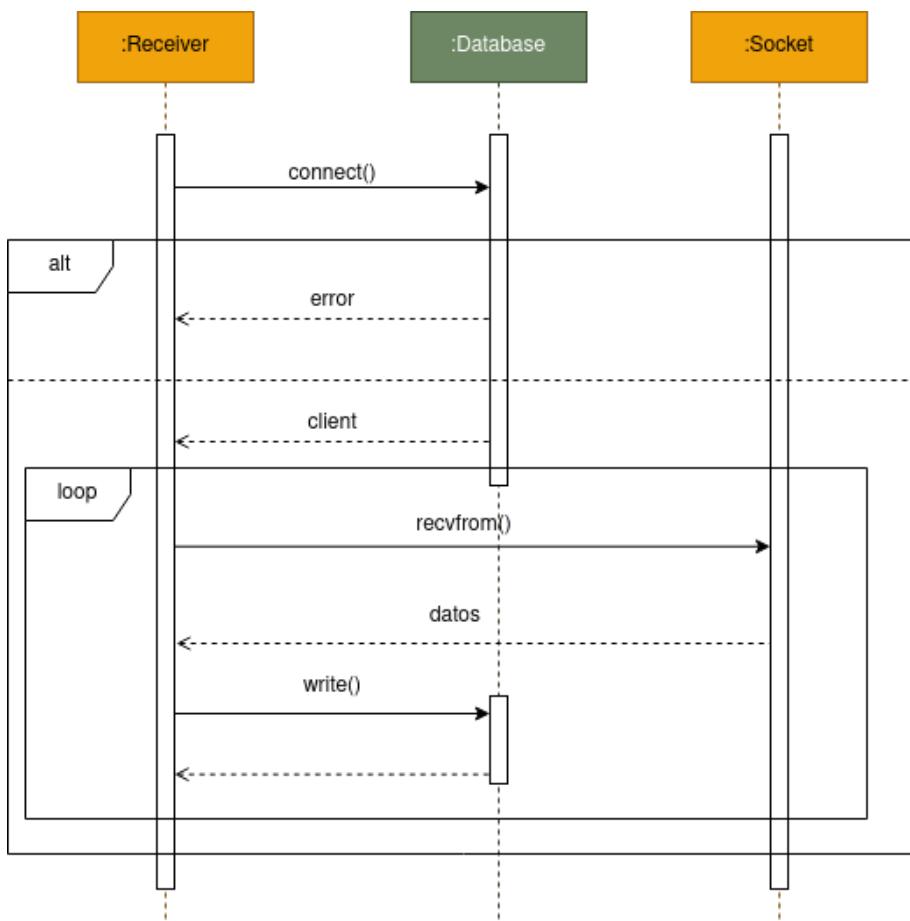


Figura D.3: Diagrama de secuencia del servicio “Receiver”

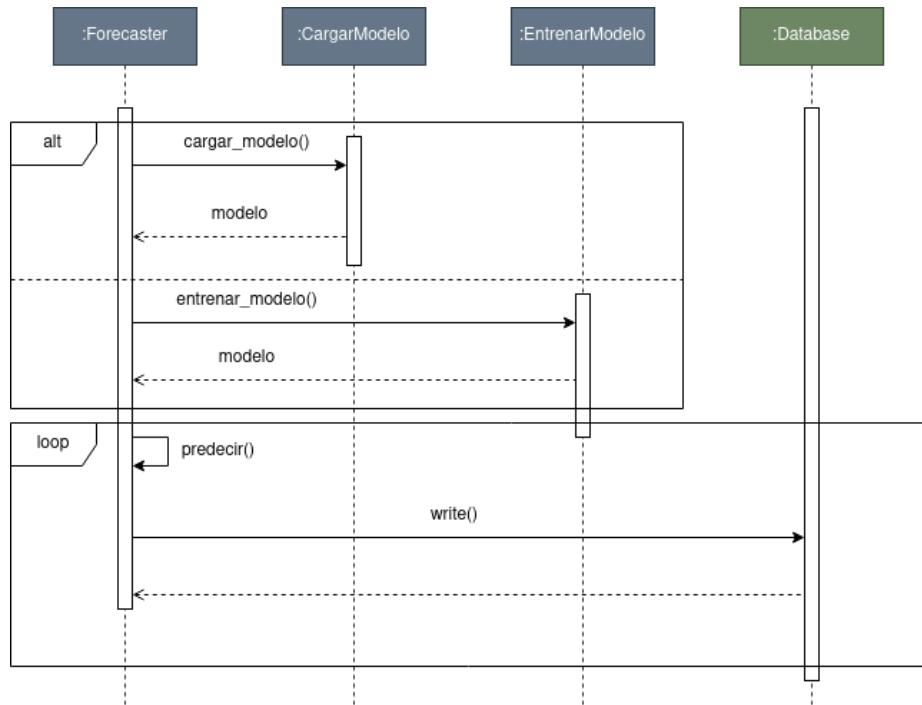


Figura D.4: Diagrama de secuencia del servicio “Forecaster”

D.4. Diseño arquitectónico

Debido a la simplicidad del código utilizado en el proyecto, se ha decidido seguir un enfoque de programación funcional, más que programación orientada a objetos. Por esto, no podrán definirse diagramas típicos de UML como puede ser un diagrama de clases.

Diseño de paquetes

El código del proyecto ha sido organizado según el servicio al que pertenecen. De esta forma se simplifica la organización, consiguiendo un proyecto con un código más fácil de leer y de mantener.

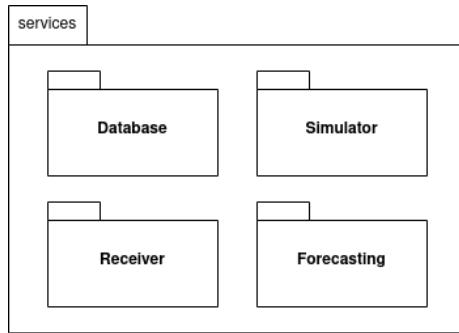


Figura D.5: Diagrama de paquetes

Como se puede ver en la Figura D.5, existen cuatro paquetes, uno por cada servicio:

- Database: contiene los scripts que se ejecutan al iniciar la base de datos.
- Simulator: contiene el código y archivos de configuración relacionados con el simulador.
- Receiver: contiene el código y archivos de configuración del servicio “Receiver”.
- Forecasting: contiene el código y archivos de configuración del servicio de predicción.

Diseño de despliegue

Un diagrama de despliegue muestra la arquitectura de ejecución de sistemas que representan la asignación de artefactos de software a objetivos de despliegue [18]. La Figura D.6 muestra el diagrama de despliegue del sistema.

Cada nodo normalmente representa un dispositivo hardware o un entorno de ejecución de software, y las asociaciones representan las comunicaciones entre nodos.

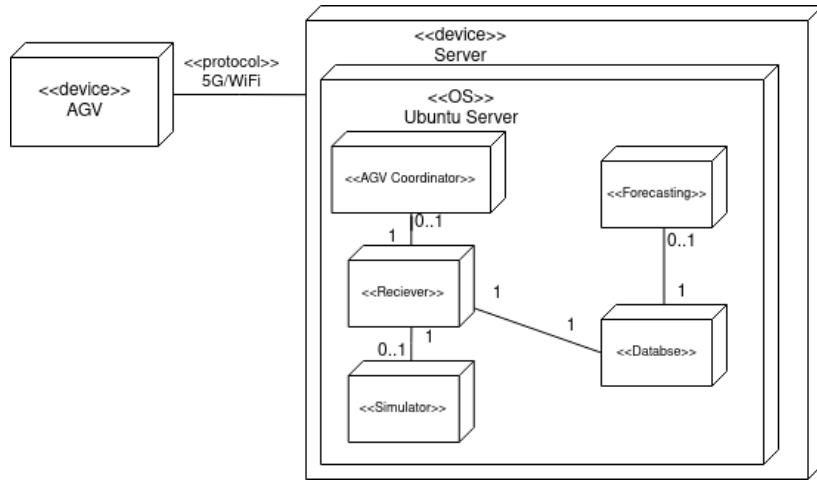


Figura D.6: Diagrama de despliegue

En este caso, el diagrama de despliegue se ha modelado de forma que todos los servicios del sistema se ejecutan de manera independiente sobre el mismo hardware y en el mismo sistema operativo. Sin embargo, esto no tiene por qué ser así. Ya que son servicios independientes, podrían ejecutarse en sistemas operativos diferentes, e incluso en dispositivos hardware diferentes.

Esto puede tener sentido desde un punto de vista del rendimiento, ya que podrían tenerse varios servidores, de forma que, por ejemplo, uno esté optimizado para alojar la base de datos, mientras que otro podría estar preparado con hardware especial para realizar las predicciones de la manera más rápida posible.

Apéndice E

Documentación técnica de programación

E.1. Introducción

En este apartado del anexo se detallará la organización de los repositorios del proyecto, así como un manual de programador, una guía de instalación y ejecución del mismo y las pruebas realizadas al sistema.

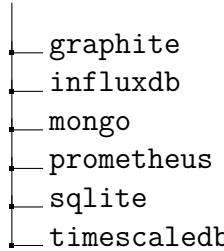
E.2. Estructura de directorios

El repositorio se encuentra alojado en https://github.com/gbd1004/TFG_AGV. La organización de los directorios de dicho repositorio queda definida de la siguiente manera:

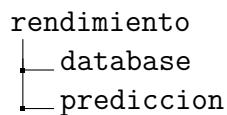
```
memoria
  └── build
  └── img
  └── tex
```

El directorio memoria guarda todos los archivos relacionados con la memoria y estos anexos. En el subdirectorio build se encuentran los archivos pdf finales, en img se encuentran las imágenes utilizadas y en tex se guardan los archivos de LaTeX de cada uno de las secciones de la memoria y los anexos.

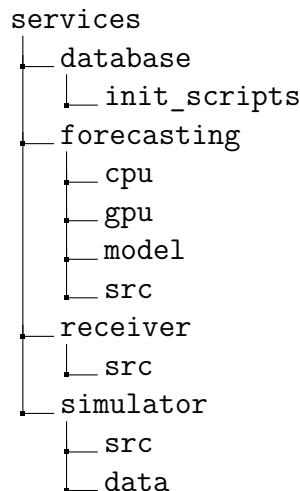
```
prototipos
  └── datos
  └── predicción
```



En el directorio prototipos se guardan todos los prototipos creados durante la realización del proyecto. Es código realizado de forma rápida con el fin de adecuarme al uso e investigar la funcionalidad básica de cada una de las diferentes herramientas consideradas.



En el directorio de rendimiento se guarda todo el código de las pruebas de rendimiento realizadas para la comparación de las bases de datos y modelos de predicción. En el subdirectorio de predicción se encuentra también el código utilizado para la optimización de los hiperparámetros de los modelos comparados, así como los resultados de cada modelo.



En el directorio de servicios se encuentra todo el código y archivos de configuración de cada uno de los servicios desarrollados en este proyecto. El código se encuentra en el directorio “src” de dentro de cada uno de los subdirectorios de cada servicio.

Dentro de la carpeta “database” se guarda una plantilla con las tablas utilizadas en la visualización de los datos. A su vez, dentro de esta carpeta se encuentra la carpeta “init_scripts”, en la que se encuentran todos los scripts que se ejecutarán en la base de datos en el momento de su inicialización.

En la carpeta “data”, dentro del directorio del simulador, se guardan los archivos CSV utilizados para leer desde este servicio.

En los subdirectorios “cpu” y “gpu” dentro del servicio de predicción se encuentran los Dockerfiles de las respectivas versiones de este servicio.

E.3. Manual del programador

En esta sección se detallarán todas las dependencias necesarias para posibles futuros desarrollos, así como los pasos necesarios para realizar dicha tarea. Este manual está pensado para el desarrollo sobre GNU/Linux, más específicamente a cualquier distribución derivada de Debian como puede ser Ubuntu o Mint.

Entorno de desarrollo

Como todos los servicios se ejecutan de manera virtualizada dentro de contenedores de Docker, técnicamente las únicas herramientas esenciales son las siguientes:

- Git: para clonar el repositorio.
- Docker: para crear los contenedores de cada servicio.
- Docker compose: para ejecutar todos los contenedores creados.
- Nvidia-docker: en caso de realizar el entrenamiento del modelo usando GPU.

Desarrollar de esta manera no es nada práctico, por lo que las dependencias necesarias para trabajar adecuadamente son, a demás de las anteriores:

- Python 3.10
- InfluxDB 2.6.1
- Darts

Git

Git es necesario para poder interactuar con el repositorio, pues nos permite clonarlo, crear nuevas ramas, subir cambios, etc. Git puede ser encontrado en [19].

Docker

Docker [20] es una herramienta que permite ejecutar procesos en contenedores. Un contenedor es un proceso aislado del resto del sistema, similar a una máquina virtual. Para ello, se crea lo que se conoce como imagen, que es una especie de plantilla del contenedor. Esta imagen contiene el sistema de ficheros del contenedor y todas las dependencias necesarias para ejecutar los procesos de este.

Gracias a esto, conseguimos que el sistema sea portable, pudiendo ser ejecutado en cualquier sistema que soporte Docker. Gracias también a Docker se simplifica mucho el despliegue del sistema, pues todas las dependencias se instalan en el sistema de ficheros especificado en la imagen.

Para definir estas imágenes se utiliza un archivo llamado “Dockerfile”. Por ejemplo, este es el Dockerfile del servicio simulador:

```
FROM python:3

COPY ./requirements.txt .
RUN pip install -r requirements.txt

WORKDIR /simulator
COPY . .

EXPOSE 5005/udp

CMD ["python", "-u", "src/main.py"]
```

En este Dockerfile se especifica una imagen que se utilizará como base. Estas imágenes se extraen del repositorio DockerHub. Después se instalan las dependencias necesarias, se expone un puerto para poder comunicarse con otros contenedores y por último se ejecuta el proceso correspondiente.

Docker compose

Docker-compose [21] es una herramienta diseñada para definir y ejecutar aplicaciones conformadas por varios contenedores de Docker. Los contenedores y redes virtuales de dicha aplicación se definen en un archivo YAML, que se utilizará para crear, ejecutar o detener todos los contenedores con un simple comando.

Nvidia-docker

Esta herramienta [22] desarrollada por Nvidia permite a los contenedores de Docker utilizar la GPU del sistema. Gracias a esto se puede acelerar el entrenamiento de los modelos de manera sustancial.

Python

Python es uno de los lenguajes de programación más utilizados de forma general, y el más utilizado para aplicaciones de ciencia de datos y aprendizaje automático. Al ser un lenguaje interpretado no es necesario compilarlo para su ejecución, por lo que el proceso de despliegue del código en los contenedores de Docker se simplifica bastante. Python puede ser descargado en [23].

InfluxDB

InfluxDB es un sistema gestor de bases de datos para series temporales. Si bien se puede instalar de forma local, es muy recomendable por comodidad ejecutarlo en un contenedor de Docker a partir de la imagen oficial [24].

Contribuciones

Estructura del repositorio

El repositorio está compuesto de dos ramas principales:

- Main: la rama principal, con la última versión estable.
- Develop: la rama sobre la que se desarrolla.

Proceso de contribución

El proceso para contribuir y añadir nuevas características o arreglar problemas es el siguiente:

1. Crear una nueva rama desde la rama “develop”. Esta rama deberá seguir la siguiente convención de nombres:
 - Nueva característica: en el caso de que la contribución sea una nueva característica, la rama deberá llamarse “feat/nombreCaracteristica”.

- Corrección de error: si la contribución es el arreglo de un error, deberá llamarse “fix/numeroError”. Los números de los errores estarán definidos en el apartado Issues del repositorio de GitHub.
 - Corrección de estilo: si la contribución es un cambio estilístico o que no afecte al funcionamiento, la rama deberá llamarse “style/nombreServicio”, utilizando el nombre del servicio al que afecta.
2. Refactorizar hasta que pase la prueba de estilo de código.
 3. Crear la pull reques para incorporar la rama creada en develop.

Reporte de errores

Para reportar un error, se seguirá la siguiente convención:

- Título: el nombre del error queda definido como “[BUG] Título”.
- Descripción: deberá añadirse una descripción breve del error, en la que deberán incluirse trazas del programa y pasos para reproducir el error.
- Etiqueta: deberá asignarse la etiqueta de “bug” en el apartado “Labels” a la hora de crear el error en GitHub.

Sugerencias de nuevas características

Las sugerencias se publicarán en el apartado de “Issues” del repositorio y deberán seguir el siguiente formato:

- Título: el título deberá seguir el siguiente formato: “[FEAT] Título”.
- Descripción: deberá contener una descripción que diga a qué servicios afecta la nueva característica y qué ventajas supone su implementación.

E.4. Compilación, instalación y ejecución del proyecto

El primer paso para la instalación del proyecto es clonar el repositorio de GitHub mediante el siguiente comando:

```
$ git clone https://github.com/gbd1004/TFG_AGV.git
```

Una vez clonado el repositorio, se procederá a la ejecución del mismo. Para la ejecución del sistema y de todos los servicios, utilizando la GPU para el servicio de predicción, se ejecuta el siguiente comando:

```
$ docker compose --profile gpu up --build
```

En el caso de que se quiera ejecutar el servicio de predicción de forma que utilice la CPU, se ejecuta el siguiente comando:

```
$ docker compose --profile cpu up --build
```

Se puede ejecutar también el sistema sin ejecutar el servicio de predicción. Para ello basta con no especificar un perfil concreto en el comando:

```
$ docker compose up --build
```

En el caso de que quiera ejecutarse de forma desacoplada al terminal actual y sin ver los registros, se añade la opción “-d” al comando anterior.

Para detener los servicios se ha de ejecutar el siguiente comando:

```
$ docker compose down
```

Para crear las imágenes de los contenedores de los servicios sin ejecutarlos, se introduce el siguiente comando:

```
$ docker compose build
```

E.5. Pruebas del sistema

En este proyecto, se han realizado pruebas de forma manual debido a la complejidad y naturaleza única del código que se ha desarrollado. El sistema presenta características altamente personalizadas y requiere interacciones específicas que no son fáciles de simular en pruebas automáticas. Además, la configuración del entorno y las dependencias externas hacen que las pruebas automatizadas sean complicadas de implementar de manera efectiva.

Pruebas realizadas

Las siguientes pruebas han sido realizadas:

- Predicciones con GPU entrenando un nuevo modelo.

- Predicciones con GPU usando un modelo ya entrenado.
- Predicciones con CPU usando un nuevo modelo.
- Predicciones con CPU usando un modelo ya entrenado

Para ello se inician todos los servicios como se ha visto en el apartado anterior de esta sección de los anexos.

Mensajes de registro

En todas las pruebas deben verse los mensajes de que los servicios “Simulator” (Figura E.1) y “Receiver” (Figura E.2) han cargado correctamente y los valores mostrados concuerdan con lo especificado en los archivos de configuración.

```
services-simulator-1 | INFO:root:Simulando desde CSV
services-simulator-1 | INFO:root:Ejecución en bucle
```

Figura E.1: Mensaje de carga del servicio “Simulator”

```
services-receiver-1 | INFO:root:Conexión a la base de datos realizada con éxito. Reintentos: 0
```

Figura E.2: Mensaje de carga del servicio “Receiver”

En el caso de utilizar la versión con GPU, en los registros del sistema deberá aparecer el mensaje de que CUDA ha cargado correctamente (Figura E.3). En el caso de utilizar CPU, este mensaje no aparece en el registro.

```
services-forecasting_gpu-1 | =====
services-forecasting_gpu-1 | = CUDA =
services-forecasting_gpu-1 | =====
services-forecasting_gpu-1 | CUDA Version 11.8.0
services-forecasting_gpu-1 | Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
services-forecasting_gpu-1 | This container image and its contents are governed by the NVIDIA Deep Learning Container License.
services-forecasting_gpu-1 | By pulling and using the container, you accept the terms and conditions of this license:
services-forecasting_gpu-1 | https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license
services-forecasting_gpu-1 | A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your convenience.
services-forecasting_gpu-1 |
```

Figura E.3: Registro de CUDA

Para los casos en los que se cargue el modelo de uno ya entrenado, se deben ver los siguientes mensajes en el registro (Figuras E.4 y E.5)

```
services-forecasting_gpu-1 | INFO:root:Esperando 60 segundos para datos para scaler
```

Figura E.4: Mensaje de espera para el escalador

```
services-forecasting_gpu-1 | GPU available: True (cuda), used: True
services-forecasting_gpu-1 | TPU available: False, using: 0 TPU cores
services-forecasting_gpu-1 | IPU available: False, using: 0 IPUs
services-forecasting_gpu-1 | HPU available: False, using: 0 HPUs
services-forecasting_gpu-1 | You are using a CUDA device ('NVIDIA GeForce RTX 3080') that has Tensor Cores. To properly utilize them, you should set `torch.set_float32_matmul_precision('medium' | 'high')` which will trade-off precision for performance. For more details, read https://pytorch.org/docs/stable/generated/torch.set_float32_matmul_precision.html#torch.set_float32_matmul_precision
services-forecasting_gpu-1 | LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
```

Figura E.5: Mensaje de carga del modelo

En los casos en los que se entrene un nuevo modelo, aparecen los siguientes registros (Figuras E.7 y E.6).

```
services-forecasting_gpu-1 | INFO:root:Esperando 290s para datos para entrenamiento
```

Figura E.6: Mensaje espera para el entrenamiento

```
services-forecasting_gpu-1 | INFO:root:Iniciando entrenamiento
services-forecasting_gpu-1 | INFO:darts.models.forecasting.torch_forecasting_model:Train dataset contains 2602 samples.
services-forecasting_gpu-1 | INFO:darts.models.forecasting.torch_forecasting_model:Time series values are 32-bits; casting model to float32.
services-forecasting_gpu-1 | GPU available: True (cuda), used: True
services-forecasting_gpu-1 | TPU available: False, using: 0 TPU cores
services-forecasting_gpu-1 | IPU available: False, using: 0 IPUs
services-forecasting_gpu-1 | HPU available: False, using: 0 HPUs
services-forecasting_gpu-1 | You are using a CUDA device ('NVIDIA GeForce RTX 3080') that has Tensor Cores. To properly utilize them, you should set `torch.set_float32_matmul_precision('medium' | 'high')` which will trade-off precision for performance. For more details, read https://pytorch.org/docs/stable/generated/torch.set_float32_matmul_precision.html#torch.set_float32_matmul_precision
services-forecasting_gpu-1 | LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
services-forecasting_gpu-1 |
services-forecasting_gpu-1 |   | Name           | Type            | Params
services-forecasting_gpu-1 | -----
services-forecasting_gpu-1 | 0 | criterion     | MSELoss         | 0
services-forecasting_gpu-1 | 1 | train_metrics | MetricCollection | 0
services-forecasting_gpu-1 | 2 | val_metrics   | MetricCollection | 0
services-forecasting_gpu-1 | 3 | encoder        | Linear          | 256
services-forecasting_gpu-1 | 4 | positional_encoding | PositionalEncoding | 0
services-forecasting_gpu-1 | 5 | transformer    | Transformer     | 548 K
services-forecasting_gpu-1 | 6 | decoder         | Linear          | 3.2 K
services-forecasting_gpu-1 | -----
services-forecasting_gpu-1 | 552 K   Trainable params
services-forecasting_gpu-1 | 0       Non-trainable params
services-forecasting_gpu-1 | 552 K   Total params
services-forecasting_gpu-1 | 2.208   Total estimated model params size (MB)
```

Figura E.7: Mensaje de entrenamiento del modelo

En estos dos últimas figuras (E.5 y E.7) se puede observar también si hay una GPU disponible para su utilización, o bien se utiliza la CPU.

Por último, por cada predicción realizada con éxito, el siguiente mensaje debería verse en los registros (Figura E.8).

```
Predicting DataLoader 0: 100%|██████████| 1/1 [00:00<00:00, 3.52it/s]
services-forecasting_gpu-1 | INFO:root:Prediccion realizada correctamente
```

Figura E.8: Mensaje de éxito en la predicción

Visualización de los datos

Después de que las primeras predicciones empiecen a generarse, deberían poder verse utilizando la plantilla guardada en “services/databse/prediccioness_dashboard.json”. Para ello, se inicia sesión en la aplicación web de la base de datos, alojada en la dirección “localhost:8086”. Después, se pincha en el apartado de “Dashboards” y dentro se pincha en “Import Dashboard”, dentro de “Create dashboard”.

Una vez importado el tablero, se puede configurar el rango temporal de los datos a visualizar, así como cada cuanto se refrescan las gráficas incluidas. En estos gráficos (Figura F.9) deberían verse en azul los datos reales del AGV y en naranja las predicciones realizadas.



Figura E.9: Predicciones y datos reales

Apéndice F

Documentación de usuario

F.1. Introducción

Este manual detalla los requisitos desde el punto de vista del usuario, así como la guía de instalación y un manual del funcionamiento del proyecto.

F.2. Requisitos de usuarios

Requisitos de hardware

Como ya ha sido mencionado con anterioridad, es muy recomendable utilizar la GPU para acelerar la predicción de los datos. Para ello, se necesita una GPU con soporte para CUDA 11.8 o superior. Dicha compatibilidad puede comprobarse en [25].

El otro elemento que necesita de un buen rendimiento es la base de datos. Un hardware insuficiente puede suponer retrasos a la hora de insertar nuevos datos e incluso cuelgues del sistema, por lo que los requisitos mínimos de hardware irán dictados en gran medida por dicho servicio. Según la documentación de InfluxDB [26], para nuestras necesidades estimadas se necesita de una CPU de 2 a 4 núcleos y de 2 a 5 GB de memoria RAM. Un disco SSD también es muy recomendado.

Requisitos de software

Para la ejecución del proyecto las únicas herramientas necesarias son Docker [20] y docker compose [21]. En cuanto al sistema operativo, el

proyecto puede ejecutarse en cualquier sistema operativo que soporte estas aplicaciones.

En el caso de querer usar la versión del servicio de predicción que utiliza la GPU, es necesario también instalar nvidia-docker [22]. Esta herramienta solo funciona en Linux, por lo que este es el único sistema operativo soportado para este caso.

F.3. Instalación

El primer paso para instalar el proyecto es clonar el repositorio con el siguiente comando:

```
$ git clone https://github.com/gbd1004/TFG_AGV.git
```

Después de clonar el repositorio se navega a la carpeta “services” y se ejecuta el siguiente comando para construir las imágenes de los contenedores de cada servicio:

```
$ docker compose build
```

Una vez ejecutados estos comandos tendremos el proyecto listo para ejecutarse.

F.4. Manual del usuario

Ejecución del proyecto

Para ejecutar el proyecto usando la GPU se utiliza el siguiente comando desde la carpeta “services”:

```
$ docker compose --profile gpu up --build
```

En caso de que no quiera, o no pueda, usarse la GPU, y por ende se utilice la CPU para el servicio de predicciones, se utiliza el siguiente comando:

```
$ docker compose --profile cpu up --build
```

Por último, mencionar que si no quiere ejecutarse el servicio de predicción basta con ejecutar el mismo comando sin especificar ningún perfil.

```
$ docker compose up --build
```

La opción “–build” construye las imágenes de los contenedores en caso de que algo haya cambiado o no se haya ejecutado el paso anterior.

Opcionalmente se puede añadir también el argumento “-d” en caso de que se quiera ejecutar desacoplado del terminal actual.

Para detener la ejecución del proyecto se ejecuta el siguiente comando:

```
$ docker compose down
```

Se puede añadir también el argumento “-v” en caso de que queramos eliminar los volúmenes montados en los contenedores.

Archivos de configuración

Cada servicio cuenta con sus archivos de configuración. A continuación se detallan los elementos de cada uno de ellos.

Database

El archivo de configuración de la base de datos se encuentra en el directorio “./services/database” y se llama “influxdb_credentials.env”. Este archivo de variables de entorno se carga en el servicio de la base de datos y en todos los servicios que tengan conexión con dicha base de datos.

El contenido es el siguiente:

- DOCKER_INFLUXDB_INIT_USERNAME: Nombre de usuario utilizado para iniciar sesión en la aplicación web.
- DOCKER_INFLUXDB_INIT_PASSWORD: Contraseña del usuario del sistema.
- DOCKER_INFLUXDB_INIT_ADMIN_TOKEN: Token de seguridad del usuario administrador, y que tiene permisos de escritura en la base de datos.
- DOCKER_INFLUXDB_INIT_ORG: Nombre de la organización en la que se crean los “buckets” y usuarios de la base de datos.
- DOCKER_INFLUXDB_INIT_BUCKET: Nombre del “bucket” en el que se insertan los datos del AGV o del simulador.
- DOCKER_INFLUXDB_INIT_MODE: Modo en el que se inicia la base de datos.

De todos estos campos, el único que no es configurable es el último, pues sin este valor la base de datos no se cargará correctamente. Tampoco se recomienda cambiar el campo DOCKER_INFLUXDB_INIT_BUCKET, aunque el sistema no debería tener problemas en el caso de que se especifique otro nombre para el bucket de datos del AGV.

Forecasting

El archivo de configuración del servicio de predicción se encuentra en la carpeta “./services/forecasting”, y se llama “config.json”. En este archivo aparecen los siguientes campos:

- load_model: guarda un valor booleano. En caso de ser “true” el servicio carga el modelo a entrenar. Por contra, si el valor es “false” el modelo se entrenará en la ejecución del servicio.
- model_file: nombre del modelo a cargar. Es también el nombre que tendrá el modelo guardado en el caso de que se entrene el modelo.
- wait_time_before_train: tiempo a esperar antes de entrenar para dar tiempo a que haya datos que utilizar en la base de datos. Cuanto más alto sea este valor, más preciso será el modelo, pero también más tiempo tardará en entrenarse.
- wait_time_before_load: tiempo a esperar antes de cargar el modelo para poder ajustar el escalador de los datos. Los datos necesitan escalarse, ya que el modelo de predicciones espera valores entre 0 y 1.

Receiver

El archivo de configuración para este servicio se encuentra en el directorio “./services/receiver” y contiene solo un único campo:

- max_retries: reintentos máximos que realizara este servicio en caso de que la conexión con la base de datos falle al iniciar este proceso.

Simulator

El archivo de configuración del servicio de simulación, llamado también “config.json”, se encuentra en la ruta “./services/simulator”. Este archivo especifica los siguientes parámetros de configuración:

- simulate: guarda un valor booleano que especifica si la simulación está activada o desactivada.
- from_csv: guarda un valor booleano, que en caso de ser “true” carga los datos de la simulación desde un csv. En caso de ser “false” los datos generados serán completamente aleatorios.
- csv_file: nombre del archivo CSV a cargar en caso de que se necesite.
- loop: valor booleano que especifica si, en caso de simular desde un CSV, al acabar de recorrer dicho fichero vuelve a ejecutarse el simulador o no.

Entorno de usuario

InfluxDB abre por defecto en el puerto 8086 una aplicación web, Chronograf, con la que podemos observar los datos almacenados. Para entrar a dicha aplicación, basta con introducir la dirección IP del ordenador en la que esté alojada la base de datos, seguido de dos puntos y el puerto. En el caso de que se aloje de forma local, se podrá acceder a Chronograf con la url “localhost:8086”. Una vez accedido, se nos mostrará una página de inicio de sesión (Figura F.1) en la que introduciremos lo establecido en el archivo de variables de entorno “influxdb_credentials.env”.

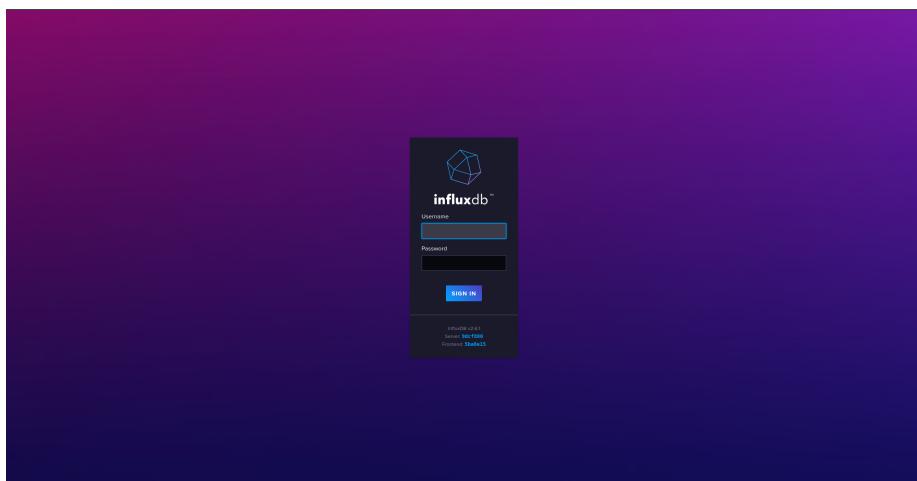


Figura F.1: Página de inicio de sesión

Una vez iniciada la sesión, se nos mostrará la siguiente ventana de bienvenida. En esta ventana, podemos ver en la parte izquierda los diferentes menús y acciones que existen.

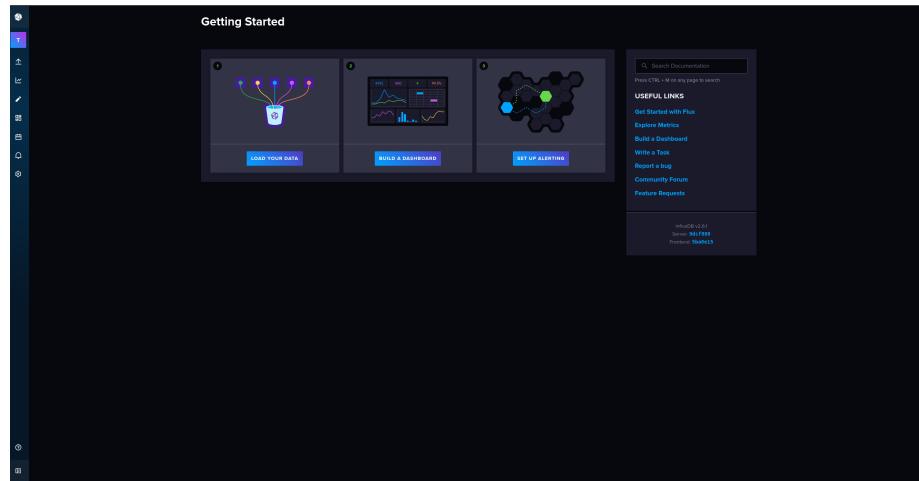


Figura F.2: Imagen de bienvenida

La primera de estas opciones muestra la información de la organización actual (Figura F.3), así como de los miembros que la forman (Figura F.4). Se puede también cambiar de organización y crear nuevas organizaciones.

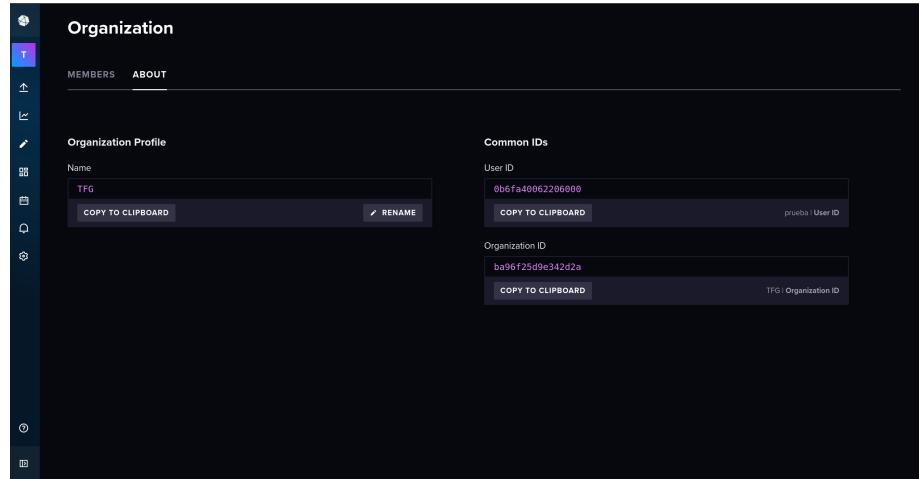


Figura F.3: Información general de la organización

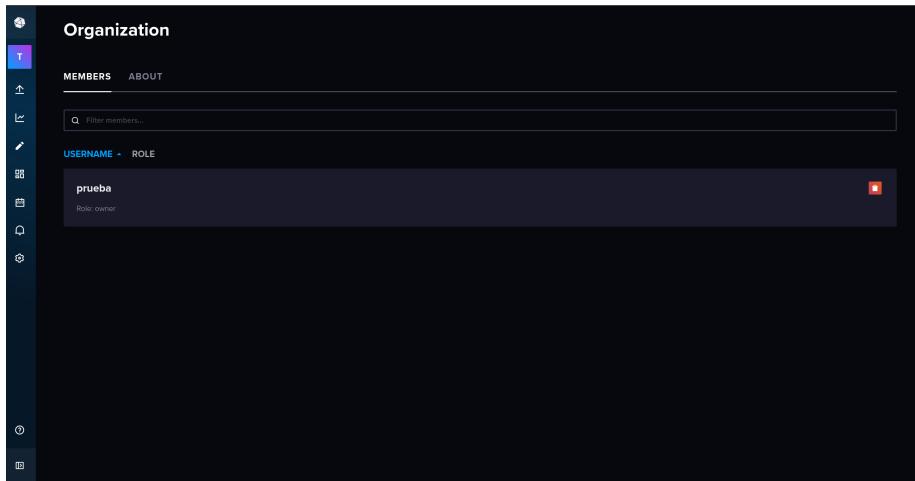


Figura F.4: Información de los miembros de la organización

El siguiente apartado, representado por una flecha hacia arriba, es el apartado “Load Data”. Desde aquí podremos realizar varias operaciones, como insertar datos, ver los “buckets” existentes, configurar Telegraf [27], etc. En nuestro caso, el único apartado interesante es el de los “buckets” (Figura F.5).

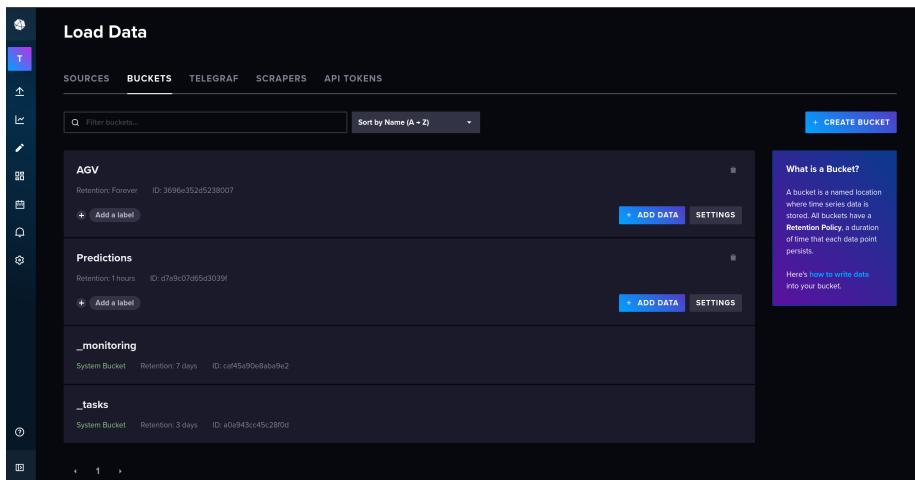


Figura F.5: Buckets de la base de datos

Al seleccionar uno de estos “buckets”, se nos redirecciona al apartado “Data Explorer”, representado en el menú lateral por un símbolo de un gráfico. Desde este apartado podremos realizar consultas a la base de datos y observar los datos recibidos de varias formas (en este caso una gráfica).

Las consultas se pueden hacer tanto de un modo gráfico (Figura F.6), como utilizando el lenguaje de consultas Flux [28] (Figura F.7)

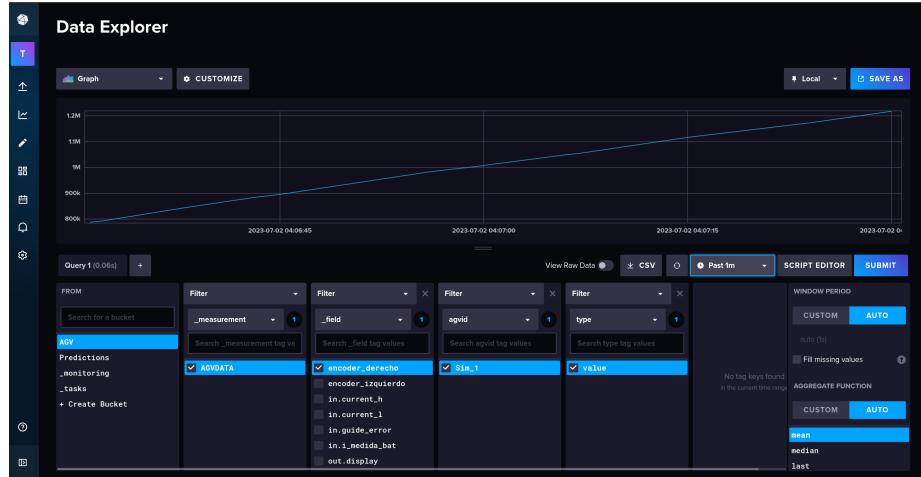


Figura F.6: Consulta gráfica

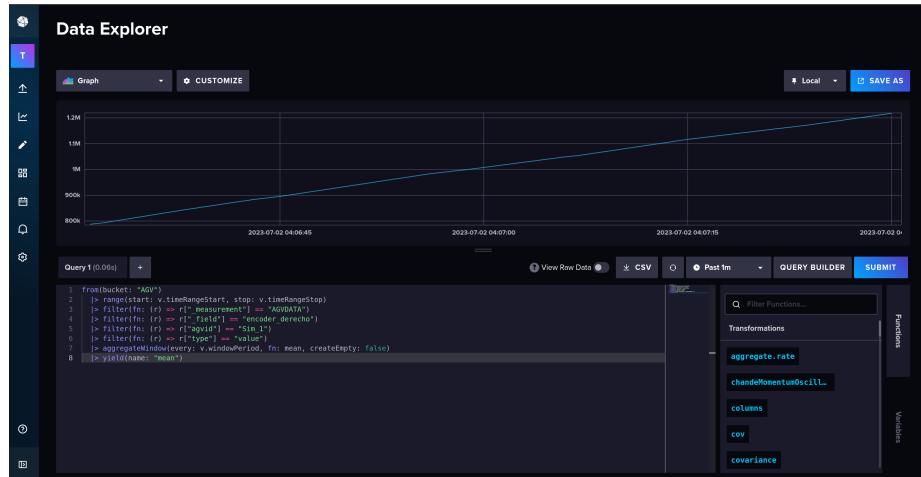


Figura F.7: Consulta con Flux

El último apartado que se explica es el de “Dashboards”, simbolizado por cuatro cuadrados pequeños. En este apartado (Figura F.8) podremos guardar consultas para luego visualizarlas en una especie de tablero. Se nos mostrarán todos los “dashboards” creados, y podremos crear nuevos, así como importar plantillas ya hechas (Se incluye la plantilla guardada en “services/database/predicciones_dashboard.json”).

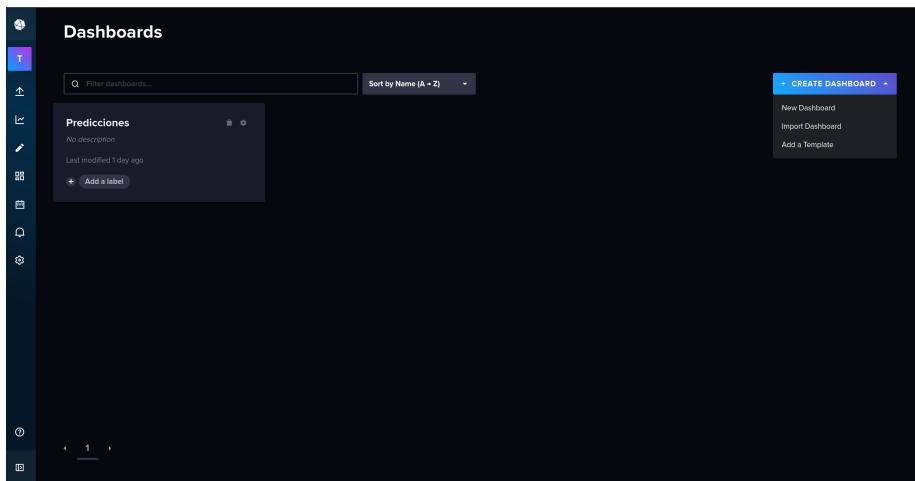


Figura F.8: Apartado de “Dashboards”

Una vez dentro de un “dashboard” (Figura F.9), se pueden ver las diferentes gráficas creadas en dicho tablero. Se puede configurar también cada cuanto quiere que se refresquen dichas gráficas y si quiere hacerse de manera automática (Figura F.10).

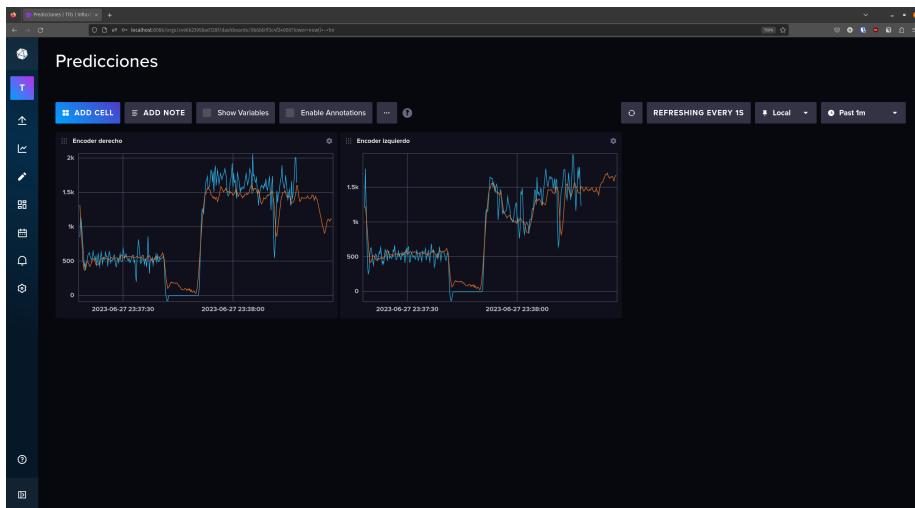


Figura F.9: Tablero con las predicciones

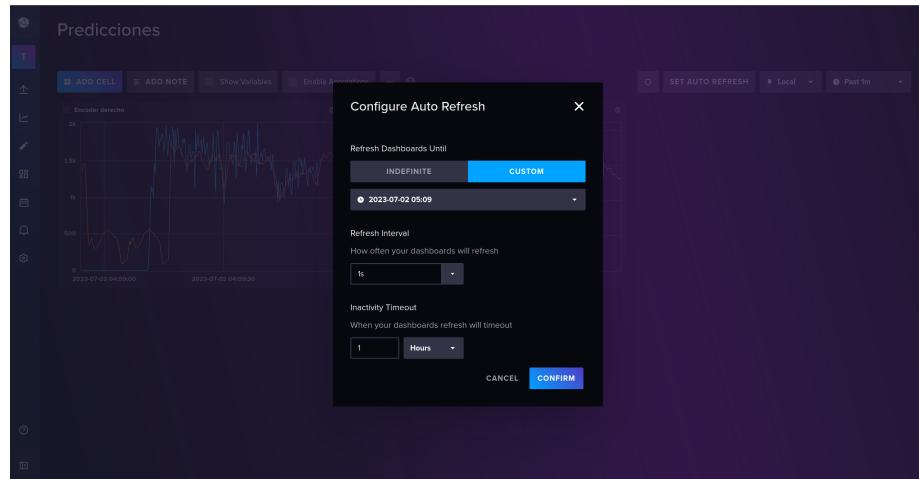


Figura F.10: Configuración del refresco automático.

En esta guía solo se han mostrado aquellas herramientas importantes para el desarrollo de este proyecto. Información más detallada sobre todas las utilidades puede verse en la documentación de Chronograf [29]

Apéndice G

Publicaciones derivadas del TFG

La metodología y comparativa de sistemas gestores de bases de datos para series temporales ha sido publicada en la conferencia SOCO 2023 [30]. Dicha publicación puede verse a continuación.

G.1. Publicación

Comparative study of open source database management systems to enable predictive maintenance of Autonomous Guided Vehicles

Gonzalo Burgos¹, J. Enrique Sierra-García², Bruno Baruque-Zanón¹

¹ Dpt. of Digitalization, University of Burgos, Burgos, Spain

² Dpt. of Electromechanical Engineering, University of Burgos, Burgos, Spain
gbd1004@alu.ubu.es, jesierra@ubu.es, bbaruque@ubu.es

Abstract. A number of open source database systems have been researched and assessed in this work for the purpose of choosing the optimum technology to implement predictive maintenance systems for industrial Autonomous Guided Vehicles. An application-driven technique has been suggested as a way to achieve it. The use case and its specifications are first outlined and listed. The top five most popular time series database systems are then contrasted based on a variety of technical metrics including software support, community support, and different technical features. From this analysis the best two options are selected (InfluxDB and TimeScale DB). The performance of these two is then further examined, taking into account performance indicators like insert time, throughput, and resource consumption. Results show that TimeScale DB provides a higher performance but demands considerably more resources.

Keywords: database management systems, time series databases, performance analysis, AGVs, industrial applications

1 Introduction

Autonomous Guided Vehicles (AGVs) are complex robotic systems, able to move around a given environment, which are used to transport heavy loads in factories or warehouses, and are designed to improve efficiency and productivity in logistics and material transportation. Due to their advantages in security, flexibility and speed, this technology is becoming increasingly common [1].

Although they may improve productivity, misalignment in their configuration or operational errors might lead to reduced throughput and in extreme cases can cause a production line shutdown. For this reason, it is necessary to extract data from running systems to analyze the performance of machines and logistics applications. This data can be used to predict future system behavior, enable preventive and predictive maintenance operations and provide feedback to design for the continuous improvement of the machines. These predictions can be achieved with the use of time series analysis algorithms that enable to anticipate future conditions of the system [2].

Some data have a low update frequency such as the temperature and the battery voltage, but others change each few milliseconds such as the electric current, the speed and position of the vehicle, errors and status, etc. All this information provided by the AGVs must be related with the time when it is generated, thus it can be grouped in time series data [3]. Any type of database can be used to store AGV data, however, since time series are involved, it is preferable to use time series databases to optimize their performance.

There are different open source database management systems to perform this task available in the market, however there is some variability in performance: some are faster than others, some are more efficient, others consume more resources, etc., so it is advisable to properly select the technology for each application in the planning or design phase. While the field has seen the exploration of similar concepts in additional literature [4], [5], our primary aim is to undertake an exhaustive examination of the database system that directly aligns with the requirements of our particular use case, which is the implementation of the monitoring platform for an AGV fleet.

In this work, several database systems have been studied and evaluated to select the best technology to deploy predictive maintenance systems for industrial AGVs. To do this, an application-driven methodology has been proposed. First, the use case and its requirements are described and identified. Then the top five most popular time series database systems are compared considering the software support, the community support and different technical indicators. From this comparison the two best tools are selected. These two are then subjected to a further performance analysis, considering performance metrics such as the insert time, the throughput, and the consumed resources.

The remaining part of the paper is structured as follows. Section 2 describes the industry 4.0 use case. The methodology of the study is explained in Section 3. The results are discussed in Section 4. The paper ends with the conclusions and future work.

2 Use case

The architecture of the system under study is depicted in Figure 1. The vehicles send their data through a 5G/WiFi connection, encoded as a byte string. Then, the "AGV Coordinator" software node, receives and decodes the messages, and transform them to the JSON format to later forward them, properly ordered, to the "Receiver" through the UDP protocol. This "Receiver" then redirects this messages to the database system that will store them. In case there are no AGV available and data needs to be generated for testing purposes, the "Receiver" may receive information from the "Simulator" software node, that will simulate the data flow sent from a fleet of AGV.

Time plays an important role in this messages. For example, if one of those messages provides information about the battery level, it is important to know when that level was recorded. Some of those variables are sent every 10 to 20 milliseconds, so a high time precision is needed.

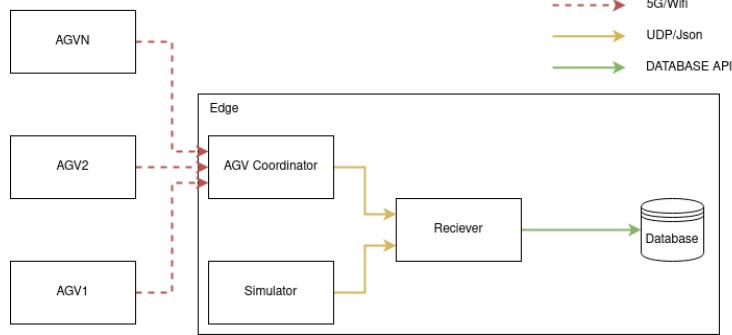


Fig. 1: AGV fleet and logging system architecture diagram

Our intention with this work is to find the database management system that best fits our use case. Table 1 contains the functional and non-functional requirements considered for our study.

Requirement Description	
FR1:	The database management system must support timestamped entries.
FR2:	Data should be able to be inserted at any moment.
FR3:	Data entries should be inserted with the timestamp corresponding to the time they were created in the AGVs.
FR4:	The database management system must support data entries with a time precision in the milliseconds range.
NFR1:	The database management system must be open source.
NFR2:	It must have good support for Linux and Python

Table 1: Functional (FR) and Non-Functional (NFR) Requirements

With the use case defined, the next section covers the methodology used to compare and choose the database management system that best suits such use case.

3 Methodology

3.1 Database management systems under study

The first thing considered is the database model to use, as this will have a key importance when deciding which database management system to use. Any of these models allow the management of time series data, like the data received from the AGVs. However, Time Series database management systems are specialized in this kind of work, making them the best suit for our needs.

For the initial screening, the most popular time series database management systems according to the DB-Engines ranking [6], have been selected. This ranking is used in previous related studies, like [4]. The time series database management systems chosen to be compared are:

1. InfluxDB (version 2.6.1) [7]. An open source time series database developed by InfluxData. Its main usage is the storage and retrieval of time series data, created in operations of monitoring IoT, sensor data, etc.
2. kdb+ (version 4.0) [8]. A relational time series database developed by KX, used mainly in high-frequency trading, to store and process data at high speed.
3. Prometheus (version 2.43.0) [9]. A time series database used for event monitoring and alerting, using an HTTP pull model.
4. Graphite (version 1.1.10) [10]. A tool that stores, monitors and graph numeric time series data.
5. TimescaleDB (version 2.10.1) [11]. An open source time series database build on top of PostgreSQL, improving the performance and analysis facilities for time series data.

Although this ranking only measures popularity and ranks the systems according to social attributes, it's really useful especially for open source solutions, as this usually means that it is supported by an active community with lots of collaborators involved in adding new functionality and correcting errors.

3.2 Comparison procedure

The comparison and filtering of the initially selected models has been performed following several sequential steps: software and community support; data model; technical specifications; and finally a performance test.

In the first two steps, the systems have been compared and those that do not meet the requirements specified in Table 1 have been discarded. Then, with the remaining systems, a performance test has been used to make the final decision.

In addition, the performance test has been separated in two complementary tests. The first test (Figure 2a) measures the CPU and RAM usage by the system when inserting data in bulk, as well as the time taken and the number of insertions per second performed. In total, 300000 entries are sent, consisting of the following fields: timestamp, vehicle id, battery, speed, x-coordinate position, y-coordinate position, temperature and voltage. To perform a more realistic simulation of the previously described AGV fleet system, the data has been sent in batches of 5000 entries that are stored in a buffer managed by the "Receiver", as this generally yields better performance than sending the entries one by one. To measure the use of CPU and RAM in the same way with all systems, the measurements have been extracted from what has been reported by the status of the Docker container in which the systems are executed [12].

In the second test (Figure 2b) the data latency is measured. That is, the time it takes for an entry to become available after insertion. For this purpose, a Python script, including two threads, has been created. One thread performs an insertion in the database and another one performs the retrieval of that same entry in a loop. At the moment the entry is obtained, the time at which it occurred is noted and subtracted from the time at which the data was inserted. This test is done 200 times and the results are averaged. Each test has been

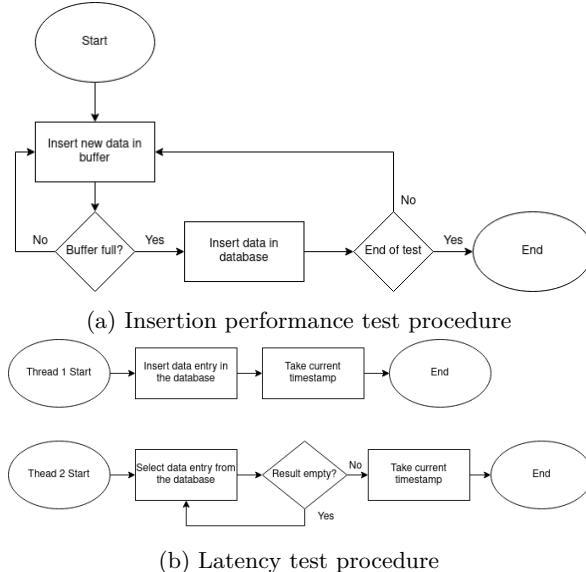


Fig. 2: Performance tests

executed 5 times to reduce variability. The results then are the average of all the executions.

3.3 Comparison perspectives

The software support aspects analyze:

- Organization: which organization or company is responsible for the development and maintenance of the system.
- Operative System: What operative systems are supported. For us, only Linux support matters.
- Python support: Our solution has been implemented in Python. For the reason, having good Python support and APIs is highly appreciated.
- Query language: which query language each database supports.
- ML Plugin: whether the system has any machine learning plugin that simplifies the forecasting of new data.

The community support indicates which system has the most active community and the characteristics compared include:

- Number of starts of the public GitHub repository: This will be used as a way of measuring its overall popularity.
- Pull requests: how many pull requests were submitted in the last month.
- Merged pull requests: of the previous pull request, how many of them have been merged.

- Issues: how many issues were submitted in the last month.
- Closed Issues: how many of those issues were closed.

The data model perspective contains a comparison of the following features:

- Data model: which data model the system implements.
- Schema: A Schema can be seen as a blueprint that defines how data is stored. This field compares whether the data organization is strict (fixed schema) or not (schema-free).
- Secondary indexes: whether the system supports secondary indexes for improved query performance.
- Time precision: the minimum time unit the timestamp of an entry can have.

The technical information perspective comprises a set of technical information fields:

- Server-side scripts: whether the system is able to execute server-side scripts.
- Partitioning method: support for partitioning methods for future scalability.
- Replication: what methods of replication the system supports, if any.
- Consistency: whether the data written is assured to be consistent or not.
- ACID compliance: whether the system follows the ACID principles (Atomicity, Consistency, Isolation, Durability)
- Concurrency: whether the system supports concurrent accesses.
- Durability: whether the data persists in the database, even if the system fails.
- Data insertion approach: whether it is done by pushing data to the database through an insert query or by pulling data from an endpoint periodically.

Lastly, the performance benchmark contains the following metrics:

- Insert time: time that takes to do the insert test, measured in seconds.
- Throughput: number of insertions per second.
- CPU usage: CPU usage of the Docker container with the database during the first test, measured as a percentage.
- RAM usage: RAM usage of the Docker container with the database during the first test, measured in Megabytes.
- Latency: time of the second test, measured in milliseconds.

4 Experiments and results

4.1 Functional comparison

Only Open Source Software has been considered for this work. Although Kdb+ has a 32 bit free version, it has not been used and does not appear in the following comparisons. For the same reason, only the features of the Community Edition of InfluxDB has not been used, and features of the Enterprise Edition, which is not open source, has not been considered.

All of the systems support Linux and Python, the operative system and language we intend to use. Only Graphite does not have a defined query language,

although data querying is possible using "Functions" [13]. Only InfluxDB supports a plugin for machine learning applications. Other systems, like TimescaleDB, provide documentation to do it externally in Python or R [14]. The Table 2 summarizes the software support of the systems compared. On the other hand, as the Table 3 shows, every project is highly active except for Graphite.

System	Organization	OS	Python	Query lang.	ML Plugins
InfluxDB	InfluxData	Linux/OS x	Yes	Flux/InfluxQL	LoudML
Prometheus	-	Linux/Windows	Yes	PromQL	No
Graphite	-	Linux/Unix	Yes	No	No
TimescaleDB	Timescale	Linux/Windows/OS x	Yes	SQL	No

Table 2: Software support

Systems	Github Stars	Pull requests	Merged pull requests	Issues	Closed Issues
InfluxDB	25.2k	26	22	37	13
Prometheus	47.4k	75	53	42	20
Graphite	5.6k	0	0	0	0
TimescaleDB	14.7k	105	83	67	46

Table 3: Community Support

Regarding the data model, both InfluxDB and Prometheus use a multidimensional mode. This model may be seen as a multidimensional key-value model: data points consist of a field that describes the data stored ("metric name" for Prometheus, and "measurement" for InfluxDB) and a set of key-value pairs associated with a timestamp. The main difference is that InfluxDB's data points consist of the measurement, a tag set and a field set, instead of only a key-value set. Tags store metadata as string values, are optional and are indexed, while fields store the actual data, are not indexed and are associated with a timestamp. Graphite automatically aggregates all data sent in windows of 1 second or more. This behaviour is not desired for our needs, as it is required to store all data sent. This comparison is shown in Table 4.

System	Model	Schema	Typing	Secondary Index	Time precision
InfluxDB	Multidimensional	Schema-free	Numeric and string	No	Nanosecond
Prometheus	Multidimensional	Yes	Numeric	No	Millisecond
Graphite	Key-Value	Yes	Numeric	No	Second
TimescaleDB	Relational	Yes	SQL types	Yes	Nanosecond

Table 4: Data model comparison

Consistency in InfluxDB is eventual, meaning that, according to the InfluxDB documentation, it prioritizes read and write performance over strong consistency. It is ensured, however, that data is eventually consistent[15].

In typical databases, the data is inserted through some type of query from the outside (the data is "pushed" to the database). Prometheus, on the contrary,

listens to an endpoint in which the data is published and "pulls" it at a fixed time interval. This means that an insertion will only happen when Prometheus polls data from the specified endpoint, thus, data may not be inserted at any time. A "Push" method is available, but it is not recommended, and it is not possible to specify timestamps.[16]. Only InfluxDB and TimescaleDB meet all the requirements, as Graphite aggregates all the data in 1-second windows, making it impossible to obtain concrete data, and the way the information is inserted in Prometheus makes it incompatible with our requirements, as it is required to be able to insert data at any time. For this reason, only these two systems have been compared in the performance tests.

System	InfluxDB	Prometheus	Graphite	TimescaleDB
Server scripts	No	No	No	Yes
Partitioning	No	Sharding	No	Yes
Replication	No	Yes	No	Yes
Consistency	Eventual	No	No	Immediate
ACID compliance	No	No	No	Yes
Concurrency	Yes	Yes	Yes	Yes
Durability	Yes	Yes	Yes	Yes
User rights	Rights via user accounts	No	No	Access rights SQL-standard
Insert method	Push	Pull	Push	Push

Table 5: Technical information

4.2 Performance evaluation

The test were performed in a computer with a AMD Ryzen 5 3600 CPU and 32 GB of RAM. Because this model of CPU has 12 threads, CPU usage may be as high as 1200% (CPU Usage = Threads * 100). The results of this performance evaluation are shown in Table 6. These results include the the system resources used in the bulk insertion of 300000 entries and the latency tests (last row). Latency is measured, only in that case, as the average time for one entry.

System	InfluxDB	TimescaleDB
Insert time (s)	24.13	1.16
Throughput (I/s)	12432.66	258620.69
CPU Usage (%)	15.05	55.32
RAM Usage (MB)	219.85	373.73
Latency (ms)	3.37	0.22

Table 6: Performance test results

According to the performance test (Table 6), InfluxDB is slower in every test than TimescaleDB, but the second uses more system resources. If scaling is necessary in the future, InfluxDB may be more convenient, as less system resource usage usually means less costs. However, TimescaleDB is more flexible, because

it extends the functionality of PostgreSQL. Thus both of these systems could be perfectly used for our use case. However, if enough computational resources are available, TimescaleDB is recommended because of its better performance and flexibility.

Here, we mainly focus on testing the performance of data insertion and availability, because at the moment we need a database that is able to store all the data sent by the fleet as fast as possible. In future work, the retrieval of that data should be measured and compared. Another thing worth to mention is that InfluxDB 3.0 version is, at the time of publication, expected to be released soon. This new version is supposed to greatly improve performance [17], so the results of this test could be outdated.

5 Conclusions and Future Work

In this study, we compared several database systems for storing data obtained from an AGV fleet in a generic industrial environment setting. Our goal was to identify which database system performed best in terms of system resources and data processing speed for managing time series data.

The results of the tests indicate that there is not a clear best system overall. While TimescaleDB is 95.19% faster, it consumes 267.57% more of CPU time, and uses 69.99% more RAM. Therefore, the decision must be taken between data availability speed and system usage when designing the final system.

Several directions for future research have been identified. One direction is to evaluate the performance of these database systems under different AGV scenarios, such as varying load capacities, different types of movements, and varying levels of complexity in layouts or AGV fleet. Another area for future research is to investigate the integration of these database systems with other emerging technologies such as machine learning and artificial intelligence. This could involve exploring how these technologies may be used to optimize data operations by AGVs, as well as how this data may be leveraged to improve decision-making in industrial operations.

Acknowledgement

This work was partially supported by the European Commission, under European Project 5G-Induce, grant number 101016941.

References

- [1] F Espinosa, C Santos, and JE Sierra-García. “Transporte multi-AGV de una carga: estado del arte y propuesta centralizada”. In: *Revista Iberoamericana de Automática e Informática industrial* 18.1 (2020), pp. 82–91.

- [2] Bruno Baruque et al. “Geothermal heat exchanger energy prediction based on time series and monitoring sensors optimization”. In: *Energy* 171 (2019), pp. 49–60. ISSN: 0360-5442. DOI: <https://doi.org/10.1016/j.energy.2018.12.207>. URL: <https://www.sciencedirect.com/science/article/pii/S0360544218325817>.
- [3] Fatoumata Dama and Christine Sinoquet. “Analysis and modeling to forecast in time series: a systematic review”. In: *CoRR* abs/2104.00164 (2021). arXiv: 2104.00164. URL: <https://arxiv.org/abs/2104.00164>.
- [4] Piotr Grzesik and Dariusz Mrozek. “Comparative Analysis of Time Series Databases in the Context of Edge Computing for Low Power Sensor Networks”. In: *Computational Science – ICCS 2020*. Ed. by Valeria V. Krzhizhanovskaya et al. Cham: Springer International Publishing, 2020, pp. 371–383. ISBN: 978-3-030-50426-7.
- [5] Alexey Struckov et al. “Evaluation of modern tools and techniques for storing time-series data”. In: *Procedia Computer Science* 156 (2019). 8th International Young Scientists Conference on Computational Science, YSC2019, 24–28 June 2019, Heraklion, Greece, pp. 19–28. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2019.08.125>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050919310439>.
- [6] DB-Engines. *DB-Engines Ranking of Time Series DBMS*. [Internet; read on 22-march-2023]. Mar. 2023. URL: <https://db-engines.com/en/ranking/time+series+dbms>.
- [7] InfluxData. *InfluxDB: Open Source Time Series Database / InfluxData*. Dec. 2021. URL: <https://www.influxdata.com/>.
- [8] KX. *KX: Make Data Driven Decisions 100X Faster At 1/10th The Cost*. Mar. 2023. URL: <https://kx.com/>.
- [9] Prometheus. *Prometheus - Monitoring system & time series database*. URL: <https://prometheus.io/>.
- [10] Graphite. URL: <https://graphiteapp.org/>.
- [11] Time-series data simplified. URL: <https://www.timescale.com/>.
- [12] Docker Inc. *Docker*. [Internet; read on 5-may-2023]. URL: <https://www.docker.com>.
- [13] Functions — Graphite 1.2.0 documentation. URL: <https://graphite.readthedocs.io/en/latest/functions.html>.
- [14] Timescale Documentation / Time-series forecasting. URL: <https://docs.timescale.com/tutorials/latest/time-series-forecast/>.
- [15] InfluxData. *InfluxDB design principles*. [Internet; read on 22-march-2023]. URL: <https://docs.influxdata.com/influxdb/cloud/reference/key-concepts/design-principles/>.
- [16] Prometheus community. *Prometheus Pushgateway*. [Internet; read on 26-april-2023]. URL: <https://github.com/prometheus/pushgateway#readme>.
- [17] InfluxData. *InfluxData Unveils Future of Time Series Analytics with InfluxDB 3.0 Product Suite*. [Internet; read on 22-march-2023]. URL: <https://www.influxdata.com/blog/influxdata-announces-influxdb-3-0/>.

Bibliografía

- [1] Ken Schwaber y Mike Beedle. *Agile software development with Scrum*. Prentice Hall PTR, 2001.
- [2] Ramón M Gómez Labrador. *Tipos de licencias de software*. 2012.
- [3] *GNU General Public License*. Inglés. Ver. 3. Free Software Foundation, 29 de jun. de 2007. URL: <http://www.gnu.org/licenses/gpl.html>.
- [4] NVIDIA Corporation. *NVIDIA Deep Learning Container License*. https://developer.download.nvidia.com/licenses/NVIDIA_Deep_Learning_Container_License.pdf. Fecha de acceso: 2023-07-03.
- [5] Massachusetts Institute of Technology. *The MIT License*. <https://opensource.org/licenses/MIT>. Fecha de acceso: 2023-07-03.
- [6] Creative Commons. *Creative Commons Licenses*. https://creativecommons.org/choose/?lang=es_ES. Fecha de acceso: 2023-07-03.
- [7] Creative Commons. *Creative Commons Attribution-ShareAlike 4.0 International License*. <https://creativecommons.org/licenses/by-sa/4.0/legalcode>. Fecha de acceso: 2023-07-03.
- [8] Andreas Meier y Michael Kaufmann. *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*. Springer Vieweg, 2019. ISBN: 3658245484.
- [9] InfluxData. *What is time series data?* [Internet; read on 22-march-2023]. URL: <https://www.influxdata.com/what-is-time-series-data/>.
- [10] DB-Engines. *DB-Engines Ranking of Time Series DBMS*. [Internet; read on 22-march-2023]. Mar. de 2023. URL: <https://db-engines.com/en/ranking/time+series+dbms>.

- [11] Piotr Grzesik y Dariusz Mrozek. “Comparative Analysis of Time Series Databases in the Context of Edge Computing for Low Power Sensor Networks”. En: *Computational Science – ICCS 2020*. Ed. por Valeria V. Krzhizhanovskaya et al. Cham: Springer International Publishing, 2020, págs. 371-383. ISBN: 978-3-030-50426-7.
- [12] Chris Davis. *Functions — Graphite 1.2.0 documentation*. URL: <https://graphite.readthedocs.io/en/latest/functions.html>.
- [13] Timescale. *Timescale Documentation / Time-series forecasting*. URL: <https://docs.timescale.com/tutorials/latest/time-series-forecast/>.
- [14] InfluxData. *InfluxDB design principles*. [Internet; read on 22-march-2023]. URL: <https://docs.influxdata.com/influxdb/cloud/reference/key-concepts/design-principles/>.
- [15] Prometheus community. *Prometheus Pushgateway*. [Internet; read on 26-april-2023]. URL: <https://github.com/prometheus/pushgateway#readme>.
- [16] Grafana Labs. *Grafana Documentation*. 2018. URL: <https://grafana.com/docs/> (visitado 25-07-2019).
- [17] Chris Newman y Graham Klyne. *Date and Time on the Internet: Timestamps*. RFC 3339. Jul. de 2002. DOI: [10.17487/RFC3339](https://doi.org/10.17487/RFC3339). URL: <https://www.rfc-editor.org/info/rfc3339>.
- [18] Jim Arlow e Ilia Neustadt. *UML 2 and the unified process: practical object-oriented analysis and design*. Pearson Education, 2005.
- [19] Git. *git*. 2023. URL: <https://git-scm.com/>.
- [20] Docker. *Docker*. 2023. URL: <https://www.docker.com/>.
- [21] Docker. *Docker Compose overview*. 2023. URL: <https://docs.docker.com/compose/>.
- [22] Nvidia. *nvidia-docker*. 2023. URL: <https://github.com/NVIDIA/nvidia-docker>.
- [23] Python Core Team. *Python: A dynamic, open source programming language*. Python version 3.10. Python Software Foundation. 2019. URL: <https://www.python.org/>.
- [24] InfluxData. *DockerHub - InfluxDB*. 2023. URL: https://hub.docker.com/_/influxdb.
- [25] Nvidia. *CUDA compatibility*. 2023. URL: <https://docs.nvidia.com/deploy/cuda-compatibility/index.html>.

- [26] InfluxData. *InfluxDB OSS guidelines*. 2023. URL: https://docs.influxdata.com/influxdb/v1.8/guides/hardware_sizing/#influxdb-oss-guidelines.
- [27] InfluxData. *Telegraf*. 2023. URL: <https://www.influxdata.com/time-series-platform/telegraf/>.
- [28] InfluxData. *Get started with Flux*. 2023. URL: <https://docs.influxdata.com/influxdb/cloud/query-data/get-started/>.
- [29] InfluxData. *Chronograf 1.10 documentation*. 2023. URL: <https://docs.influxdata.com/chronograf/v1.10/>.
- [30] Gonzalo Burgos, Enrique Sierra-García y Bruno Baruque-Zanón. “Comparative study of open source database management systems to enable predictive maintenance of Autonomous Guided Vehicles”. En: *18th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2023)*. Ed. por Pablo García Bringes et al. Cham: Springer Nature Switzerland, 2023. ISBN: Pending Publication.