UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Análisis y predicción de datos obtenidos del funcionamiento de un AGV
Documentación Técnica

Presentado por Gonzalo Burgos de la Hera
en Universidad de Burgos — 27 de junio
de 2023
Tutor: Bruno Baruque Zanón y Jesús Enrique
Sierra García

Índice general

Índice de figuras

Índice de tablas

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.

El presente proyecto de software se enfoca en el análisis detallado de la planificación temporal, la viabilidad económica y la viabilidad legal, con el objetivo de proporcionar una base sólida para su implementación exitosa. Se realizará un exhaustivo examen de cada uno de estos aspectos, considerando tanto los recursos disponibles como las restricciones y requisitos específicos del proyecto.

En primer lugar, se llevará a cabo un análisis de la planificación temporal del proyecto. Esto implica establecer un cronograma realista y eficiente, teniendo en cuenta las tareas necesarias, los hitos clave y las dependencias entre ellas. Se identificarán los recursos requeridos y se asignarán de manera adecuada para asegurar una ejecución eficiente y oportuna del proyecto. Además, se considerarán posibles riesgos y se desarrollarán estrategias de mitigación para evitar retrasos y garantizar la finalización exitosa del software en el tiempo establecido.

En segundo lugar, se realizará un estudio exhaustivo de la viabilidad económica del proyecto de software. Esto implica evaluar los costos asociados con el desarrollo, implementación y mantenimiento del software, así como proyectar los beneficios y retornos de inversión esperados. Se analizarán los posibles ingresos generados por el software, los gastos operativos, el costo

de adquisición de herramientas y tecnologías necesarias, entre otros factores relevantes. Este análisis permitirá tomar decisiones informadas sobre la asignación de recursos financieros y evaluar la rentabilidad y sostenibilidad del proyecto.

Por último, pero no menos importante, se llevará a cabo un análisis exhaustivo de la viabilidad legal del proyecto de software. Se examinarán las leyes y regulaciones pertinentes, como la protección de datos, los derechos de propiedad intelectual y cualquier otro requisito legal aplicable. Se asegurará que el software cumpla con todas las normativas y se evitarán posibles problemas legales en el futuro. Además, se considerarán las implicaciones éticas y de privacidad para garantizar el cumplimiento de estándares éticos y legales relevantes.

En resumen, este proyecto de software se enfoca en el análisis riguroso de la planificación temporal, la viabilidad económica y la viabilidad legal. Al considerar estos aspectos de manera integral, se establecerán las bases necesarias para el desarrollo exitoso del software, asegurando su viabilidad financiera, su cumplimiento legal y su capacidad para cumplir con los objetivos establecidos.

A.2. Planificación temporal

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€
Total	1500€	375€

Tabla A.1: Costes de hardware

Costes 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 la base de datos utilizada, InfluxDB.

Costes personales 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	1200€
IRPF	
Seguridad Social	
Salario bruto	
Total 6 meses	

Tabla A.2: Costes personales

Otros costes

Coste total

Beneficios

Viabilidad legal

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 sea factible modificar 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 Inserción de datos:** los datos tienen que poder insertarse en cualquier momento a la base de datos.
  - **RF-1.2 Visualización de datos:** el usuario debe poder ser capaz de visualizar los datos del AGV y 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 Base de datos:** 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. Este timestamp a demás, debe tener una precisión en el rango de los milisegundos.
- **RNF-8 Predicción:** el modelo de predicción ha de generar resultados aceptables 10 segundos en el futuro.

B.4. Especificación de requisitos

B.5. Estudio del sistema gestor de bases de datos

Teniendo en cuenta los requisitos descritos en las secciones anteriores, se realiza una comparación para elegir el sistema gestor de bases de datos que mejor cumpla dichos requisitos.

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 [1] 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 [2]. 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 [3]. Este ranking es utilizado en otros estudios, como [4]. 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 realizados 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 cumplan 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.

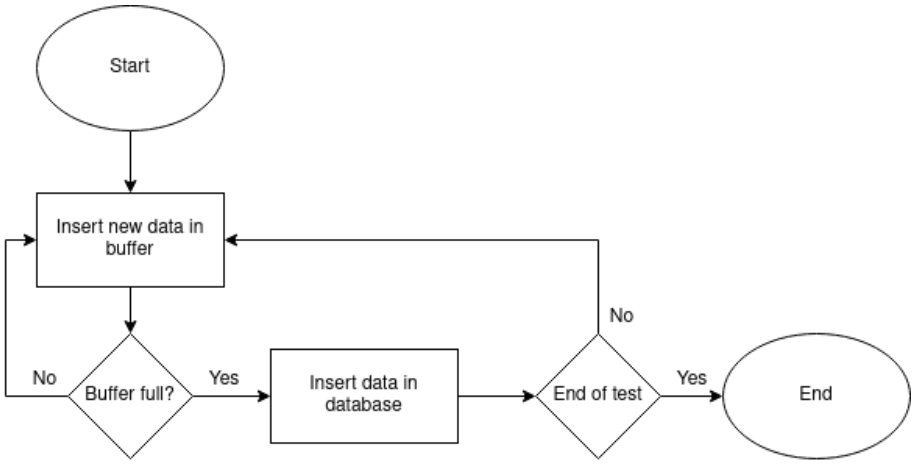

Figura B.1: Procedimiento de la prueba de inserción

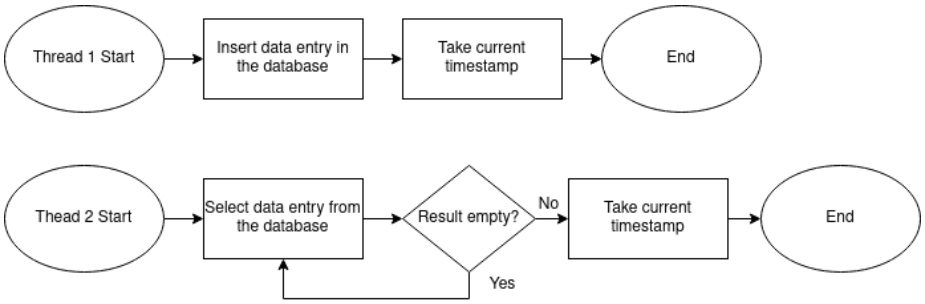

Figura B.2: Procedimiento de la prueba de latencia

El primer test, (Figura B.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 B.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 le 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: que lenguaje de consultas soporta el sistema.
- Plugins para aprendizaje automático: si el sistema soporta plugins 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: que 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: que 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

Systems	Organization	Launch year	Latest version	License
InfluxDB	InfluxData	2013	2023	OSS
db+	Kx Systems	2000	2020	Comercial
Prometheus	-	2015	2023	OSS
Graphite	-	2006	2022	OSS
TimescaleDB	Timescale	2017	2023	OSS

Tabla B.1: Comparativa de información general

Información general (Tabla B.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 "Comunity Edition" de InfluxDB serán utilizadas en dichas comparaciones, y características de la "Enterprise Edition", que no es de códigp abierto, no se tendrán en cuenta.

System	OS	Python	Query language	ML Plugins
InfluxDB	Linux OS x	Sí	Flux and InfluxQL	Loud ML
Prometheus	Linux Windows	Sí	PromQL	No
Graphite	Linux Unix	Sí	No	No
TimescaleDB	Linux OS X Windows	Sí	SQL	No

Tabla B.2: Soporte software

Soporte software (Tabla B.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 [5]. Sólo InfluxDB soporta plugins para aprendizaje automático. Otros sistemas como TimescaleDB proveen documentación para realizarlo de forma externa en lenguajes como Python o R [6].

Sistemas	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 B.3: Soporte de la comunidad

Soporte de la comunidad (Tabla B.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 B.4: Comparativa del modelo de datos

Comparación del modelo de datos (Tabla B.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 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.

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	Innmediata
ACID	No	No	No	Sí
Concurrencia	Sí	Sí	Sí	Sí
Durabilidad	Sí	Sí	Sí	Sí
Permisos de usuario	Permisos vía cuentas	No	No	Derechos estandar SQL
Método inserción	Push	Pull	Push	Push

Tabla B.5: Comparativa de información técnica

Comparativa información técnica (Tabla B.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 [7].

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 [8].

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 B.6.

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 B.6: Resultados de la prueba de rendimiento

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

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 B.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 [9]

Figura B.3: Interfaz Chronograf

B.6. Estudio del modelo de predicción

Los modelos elegidos para la comparación son los siguientes:

- ARIMA.
- TCN.
- N-HiTS.
- Transformer Model.

Estos modelos han sido comparados usando las siguientes métricas:

- MAE
- MASE
- DTW
- Tiempo de entrenamiento.
- Tiempo de 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.

Se van a hacer tres tipos de comparaciones: una comparación univariante en la que solo se prediga un valor, una comparación univariante con covariables en la que solo se prediga un valor, pero se utilicen otros como entrada y una comparación multivariante en la que se predigan todos los valores utilizados como entrada.

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 200 épocas y se realizan varias predicciones: una 200 milisegundos a futuro, otra 1 segundo a futuro y otra 10 segundos a futuro. Este procedimiento se realiza cinco veces y se calcula la media de las métricas como valor final.

Comparativa inicial

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.

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	**65.099**	**161.666**	1907.520	-	-	-	-	-	-
TCN	1171.997	1332.097	2384.317	1319.644	1309.481	2917.175	1332.920	1253.553	2375.751
N-HiTS	89.323	426.071	2051.014	527.865	731.733	1625.994	467.818	455.427	1200.268
Transformer	420.348	269.185	1856.455	234.741	178.082	779.017	189.326	364.382	**651.375**

Tabla B.7: MAE de los modelos por defecto

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	**0.301**	**0.749**	8.832	-	-	-	-	-	-
TCN	5.427	6.168	11.040	6.110	6.063	13.507	6.172	5.804	11.000
N-HiTS	0.414	1.973	9.497	2.444	3.388	7.529	2.166	2.109	5.558
Transformer	1.946	1.246	8.596	1.087	0.825	3.607	0.877	1.687	**3.016**

Tabla B.8: MASE de los modelos por defecto

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	**65.099**	**128.896**	1345.799	-	-	-	-	-	-
TCN	1171.997	1332.097	1200.933	1319.644	1309.481	2112.024	1332.920	1253.553	1190.241
N-HiTS	89.323	426.071	862.970	527.865	731.733	970.195	467.818	455.427	599.107
Transformer	420.348	252.468	593.228	234.741	133.774	**262.340**	189.326	343.909	269.327

Tabla B.9: DTW de los modelos por defecto

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	**0.346**	**0.747**	**1.110**	-	-	-	-	-	-
TCN	38.761	42.199	40.709	48.929	54.908	51.291	39.916	40.346	40.380
N-HiTS	41.019	41.613	40.470	53.496	53.334	50.245	41.693	41.385	39.363
Transformer	119.819	127.825	125.355	134.100	138.126	133.646	122.732	129.830	122.162

Tabla B.10: Tiempo de entrenamiento en segundos de los modelos por defecto

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	0.009	0.009	0.009	-	-	-	-	-	-
TCN	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
N-HiTS	0.003	0.003	0.004	0.003	0.003	0.003	0.003	0.003	0.004
Transformer	0.004	0.004	0.007	0.005	0.004	0.005	**<0.001**	**<0.001**	**<0.001**

Tabla B.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.

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.

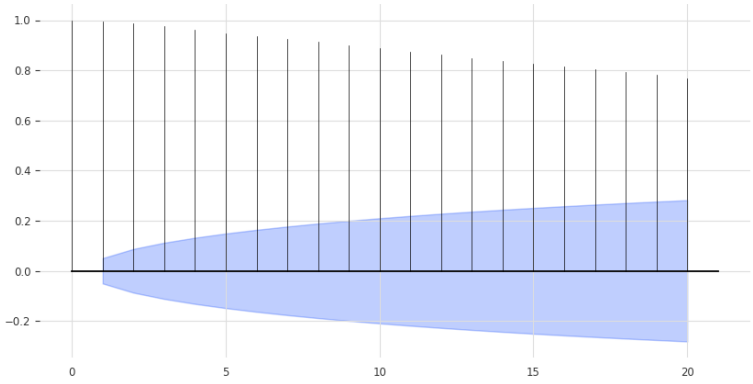

Figura B.4: ACF del encoder derecho

1

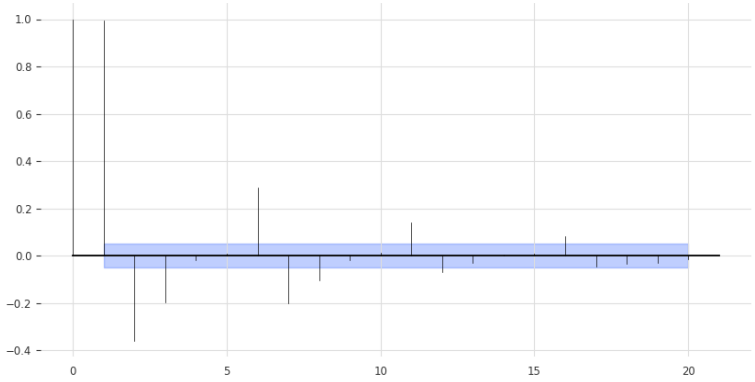

Figura B.5: PACF del encoder derecho

1

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.

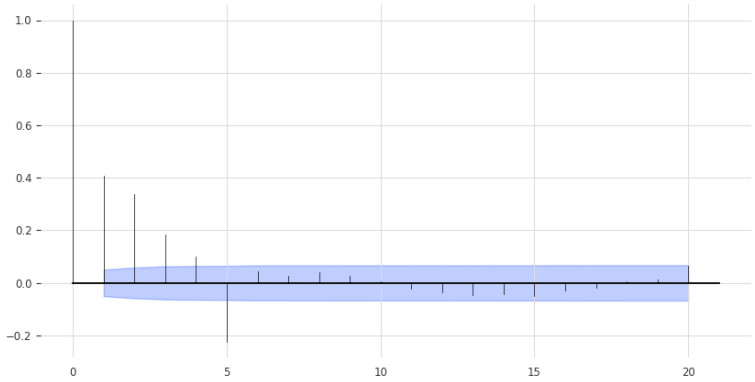

Figura B.6: ACF del encoder derecho después de diferenciar

1

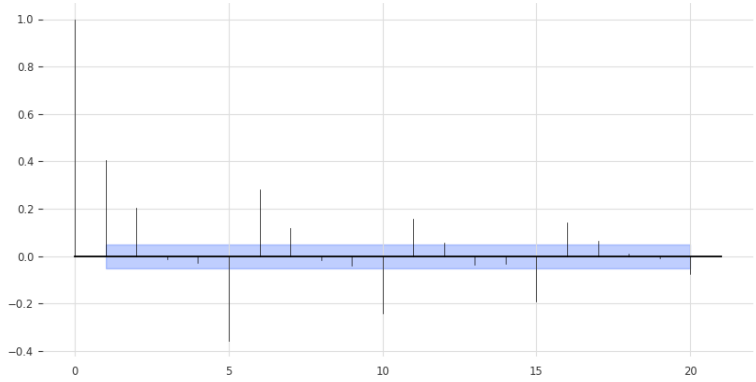

Figura B.7: PACF del encoder derecho después de diferenciar

1

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).

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	**38.961**	**133.312**	1904.280	-	-	-	-	-	-
TCN	1964.383	1664.512	2301.540	1246.520	1288.028	2032.812	1306.301	1351.884	2309.763
N-HiTS	224.531	410.060	2076.162	567.246	570.008	2242.317	490.646	611.851	**1879.377**
Transformer	810.602	740.833	2128.102	913.553	1078.878	2483.066	492.486	646.667	2266.571

Tabla B.12: MAE de los modelos optimizados

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	**0.180**	**0.617**	8.817	-	-	-	-	-	-
TCN	9.096	7.707	10.657	5.772	5.964	9.413	6.049	6.260	10.695
N-HiTS	1.040	1.899	9.613	2.627	2.639	10.383	2.272	2.833	**8.702**
Transformer	3.753	3.430	9.854	4.230	4.996	11.497	2.280	2.994	10.495

Tabla B.13: MASE de los modelos optimizados

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	**38.961**	**110.171**	1606.009	-	-	-	-	-	-
TCN	1964.383	1664.512	1108.831	1246.520	1288.028	1291.965	1306.301	1351.884	1266.140
N-HiTS	224.531	410.060	**940.509**	567.246	570.008	1416.521	490.646	611.851	1179.411
Transformer	810.602	740.833	1018.206	913.553	1078.878	1851.306	492.486	646.667	1181.562

Tabla B.14: DTW de los modelos optimizados

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	**0.843**	**1.610**	**2.410**	-	-	-	-	-	-
TCN	50.430	50.648	52.969	60.830	61.489	64.225	48.641	49.186	53.759
N-HiTS	140.357	141.576	168.711	154.149	166.998	161.123	141.904	174.071	142.991
Transformer	185.125	174.690	186.759	189.849	199.023	186.065	191.429	195.641	189.349

Tabla B.15: Tiempo de entrenamiento en segundos de los modelos optimizados

	Univariante			Covariante			Multivariante		
Tiempo	0.2	1	10	0.2	1	10	0.2	1	10
ARIMA	0.014	0.015	0.015	-	-	-	-	-	-
TCN	0.003	0.003	0.004	0.003	0.003	0.003	0.003	0.003	0.006
N-HiTS	0.005	0.033	0.006	0.005	0.004	0.004	0.005	0.004	0.006
Transformer	0.004	0.004	0.006	0.004	0.004	0.005	**<0.001**	**<0.001**	**<0.001**

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

Como se puede observar, excepto en ARIMA, la optimización arroja peores resultados que los modelos por defecto. Esto se puede deber principalmente a dos factores:

1. Los modelos por defecto son lo suficientemente buenos.
2. El tiempo de ejecución de la optimización no ha sido el suficiente, por lo que no se han encontrado las mejores combinaciones de hiperparámetros.

En cuanto a ARIMA, a cambio de obtener mejores resultados tanto el tiempo de entrenamiento como el de predicción son algo más del doble de grandes. Tampoco supone un gran problema, ya que a la hora de entrenar solo supone unos segundos más, y en el caso de la predicción el aumento está en el rango de los milisegundos.

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.

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, o bien utilizar el modelo Transformer por sus bajos tiempos de predicción.

Apéndice C

Especificación de diseño

C.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.

C.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

Como se ha especificado en el anexo B, la base de datos elegida es InfluxDB. Esta es una base de datos de series temporales no relacional, por lo que las características de esta base de datos no siguen las de una base de datos relacional.

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 [10].

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 la 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 C.1 y **??**

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

Tabla C.1: Bucket AGV

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 "Reciever", que los introducirá en la base de datos.
- Database API: El "Reciever" 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.

C.3. Diseño procedimental

C.4. Diseño arquitectónico

Apéndice D

Documentación técnica de programación

D.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.

D.2. Estructura de directorios

La organización de los directorios 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
├── forecasting
├── graphite
└── influxdb
```

```
├── mongo
├── prometheus
├── sqlite
└── timescaledb
```

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.

```
rendimiento
├── database
└── forecast
```

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 forecast se encuentra también el código utilizado para la optimización de los hiperparámetros de los modelos comparados.

```
services
├── forecasting
│   ├── cpu
│   ├── gpu
│   ├── model
│   └── src
├── reciever
│   └── src
└── simulator
    ├── src
    └── data
```

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. 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.

D.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 [11].

Docker

Docker [12] 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 utilizara 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 [13] 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 [14] 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 [15].

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 [16].

Contribuciones

D.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 ejecuar 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
```

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.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.

E.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 [17].

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 [18], 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 [12] y docker-compose [13]. En cuanto al sistema operativo, el pro-

yecto 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 [14]. Esta herramienta solo funciona en Linux, por lo que este es el único sistema operativo soportado para este caso.

E.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.

E.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 la 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.

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 cuando en el caso de que se entrene en vez de cargarlo.
- wait_time_before_train: tiempo a esperar antes de entrenar para dar tiempo a que haya datos que utilizar en la base de datos para dicho entrenamiento. 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.

Reciever

El archivo de configuración para este servicio se encuentra en el directorio "./services/reciever" 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

Bibliografía

[1] A. Meier and M. Kaufmann, *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*. Springer Vieweg, 2019.

[2] InfluxData, "What is time series data?" [Internet; read on 22-march-2023]. [Online]. Available: https://www.influxdata.com/what-is-time-series-data/

[3] DB-Engines, "Db-engines ranking of time series dbms," March 2023, [Internet; read on 22-march-2023]. [Online]. Available: https://db-engines.com/en/ranking/time+series+dbms

[4] P. Grzesik and D. Mrozek, "Comparative analysis of time series databases in the context of edge computing for low power sensor networks," in *Computational Science – ICCS 2020*, V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, Eds. Cham: Springer International Publishing, 2020, pp. 371–383.

[5] Chris Davis, "Functions — Graphite 1.2.0 documentation." [Online]. Available: https://graphite.readthedocs.io/en/latest/functions.html

[6] Timescale, "Timescale Documentation | Time-series forecasting." [Online]. Available: https://docs.timescale.com/tutorials/latest/time-series-forecast/

[7] InfluxData, "Influxdb design principles," [Internet; read on 22-march-2023]. [Online]. Available: https://docs.influxdata.com/influxdb/cloud/reference/key-concepts/design-principles/

[8] P. community, "Prometheus pushgateway," [Internet; read on 26-april-2023]. [Online]. Available: https://github.com/prometheus/pushgateway#readme

[9] Grafana Labs. (2018) Grafana documentation. [Online]. Available: https://grafana.com/docs/

[10] C. Newman and G. Klyne, "Date and Time on the Internet: Timestamps," RFC 3339, Jul. 2002. [Online]. Available: https://www.rfc-editor.org/info/rfc3339

[11] Git, "git," 2023. [Online]. Available: https://git-scm.com/

[12] Docker, "Docker," 2023. [Online]. Available: https://www.docker.com/

[13] ——, "Docker compose overview," 2023. [Online]. Available: https://docs.docker.com/compose/

[14] Nvidia, "nvidia-docker," 2023. [Online]. Available: https://github.com/NVIDIA/nvidia-docker

[15] Python Core Team, *Python: A dynamic, open source programming language*, Python Software Foundation, 2019, python version 3.10. [Online]. Available: https://www.python.org/

[16] InfluxData, "Dockerhub - influxdb," 2023. [Online]. Available: https://hub.docker.com/_/influxdb

[17] Nvidia, "Cuda compatibility," 2023. [Online]. Available: https://docs.nvidia.com/deploy/cuda-compatibility/index.html

[18] InfluxData, "Influxdb oss guidelines," 2023. [Online]. Available: https://docs.influxdata.com/influxdb/v1.8/guides/hardware_sizing/#influxdb-oss-guidelines