UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DEL SISTEMA DE TRANSMISIÓN, ALMACENAMIENTO Y VISUALIZACIÓN DE DATOS PARA EL AUTO DE CARRERAS DE URACING TEAM

MEMORIA PARA OPTAR AL TITULO DE
INGENIERO CIVIL EN COMPUTACIÓN

IGNACIO JOSÉ MOORE RUBIO

PROFESOR GUÍA:
HUGO MORA RIQUELME

MIEMBROS DE LA COMISIÓN:
EDUARDO GODOY VEGA
CLAUDIO GUTIÉRREZ GALLARDO

Este trabajo ha sido parcialmente financiado por URacing Team

SANTIAGO DE CHILE
2022

1. Resumen

En la Universidad de Chile se está realizando un proyecto que consiste en construir un auto de carreras, este proyecto es realizado por un equipo de estudiantes, URacing Team, que pertenecen a distintas especialidades y departamentos dentro de la Universidad. El auto que se construirá formará parte de una competencia internacional llamada Formula SAE, en esta competencia se hacen varias pruebas en el cual tanto el auto como el equipo forman parte, esta competencia también establece las restricciones y reglas a las que debe ser sometida la construcción y diseño del auto.

En Uracing Team se requiere desarrollar un sistema de telemetría, esto se refiere a un sistema que logre trabajar con todos los datos generados por el auto, esto puede ser transmisión, almacenamiento, análisis, visualización, etc. Se necesita este sistema, para que pueda dar retroalimentación sobre cómo funciona el auto, que partes hay que arreglar y cuales funcionan correctamente.

En esta etapa del proyecto y de lo que trata esta memoria es sobre el desarrollo de la transmisión, almacenamiento y visualización de datos en tiempo real. Para desarrollar una solución que logre satisfacer las necesidades de URacing Team, se implementaron subsistemas que en conjunto logran formar un proceso que parte desde la captura de datos en un auto de carreras y termina con la visualización de los datos en tiempo real en un dispositivo externo. Estos subsistemas son transmisión, que a la vez está dividida en envío y recepción de datos; almacenamiento; y visualización.

Cada uno de los subsistemas son desarrollados por separado en contenedores de Docker, para que puedan ser ejecutados en cualquier sistema operativo. Se estudiaron y analizaron cada uno de estos subsistemas, para implementar la mejor solución posible. Cada uno de ellos trae sus dificultades y obstáculos, una de estas es el hardware que es requerido para algunos aspectos del proyecto.

El hardware requerido no se pudo conseguir a tiempo para esta instancia del proyecto y se terminó desarrollando un prototipo del sistema, en donde se puede simular y probar los subsistemas desarrollados. Los resultados no fueron los deseados, muchas de las fallas de requisitos que se esperaban del proyecto fueron relacionados a la cantidad de datos que pueden ser enviados desde el auto, esto se debe a que no se utilizó el hardware inicialmente elegido. El resto de los subsistemas, recepción, almacenamiento y visualización en tiempo real si funcionan correctamente, las pruebas realizadas son satisfactorias.

En el futuro se seguirá trabajando en este proyecto por algún integrante de URacing Team, hay varios aspectos que aún hay que mejorar en este proyecto y también muchos subsistemas que aún hay que desarrollar y adjuntar a este sistema.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Contexto

Las competencias automovilísticas han sido de gran interés para miles de personas desde hace décadas, teniendo como antecedente las primeras carreras surgidas en Francia en el año 1894, con un auge en la década de 1930 y luego la creación de la Fórmula 1 en el año 1950. Esto es de gran relevancia, ya que da la motivación para una nueva competencia en el contexto de los deportes de motor, la Fórmula SAE [1].

La Fórmula SAE es una competencia que nace en el año 1980 en la Universidad de Texas, bajo el concepto de diseñar y construir un pequeño auto de carreras tipo Fórmula 1 por los mismos estudiantes de la universidad. El objetivo final de la competencia es que los jueces evalúen el prototipo del auto construido para ser un elemento de producción por una empresa ficticia.

Para llevar a cabo lo anterior existen diversos eventos en los cuales los jueces asignan puntaje al auto dependiendo de su rendimiento. Existen dos grandes categorías de eventos, los cuales a su vez poseen diferentes pruebas. La primera categoría son los eventos estáticos en el cual se evalúa el diseño del automóvil, los costos, la fabricación y una presentación de estos. La segunda gran categoría son los eventos dinámicos, en donde existen pruebas de aceleración, *endurance* (carrera de larga duración), economía de combustible, *skipad* (pista mojada y curvas cerradas) y *autocross* (carrera más corta).

Es en este contexto que en el año 2019 nace el primer equipo de Fórmula SAE de Chile, *URacing Team* [2], compuesto por estudiantes de la Universidad de Chile. Sin embargo, por los eventos del ”estallido social” y posteriormente la pandemia de Covid-19, no fue hasta finales de 2020 donde se retomó la idea de crear un equipo interdisciplinario con el objetivo de construir un auto de carreras y participar en las competencias oficiales de Fórmula SAE.

URacing Team está dividido en distintos equipos, como se ve en la figura 1, el área ejecutiva tiene un rol muy importante en el equipo, su principal función es generar recursos económicos mediante patrocinadores, los cuáles aportan con servicios, materiales y/o dinero. Esta parte del proyecto es sumamente importante y amplía el rango de dificultad que éste abarca.

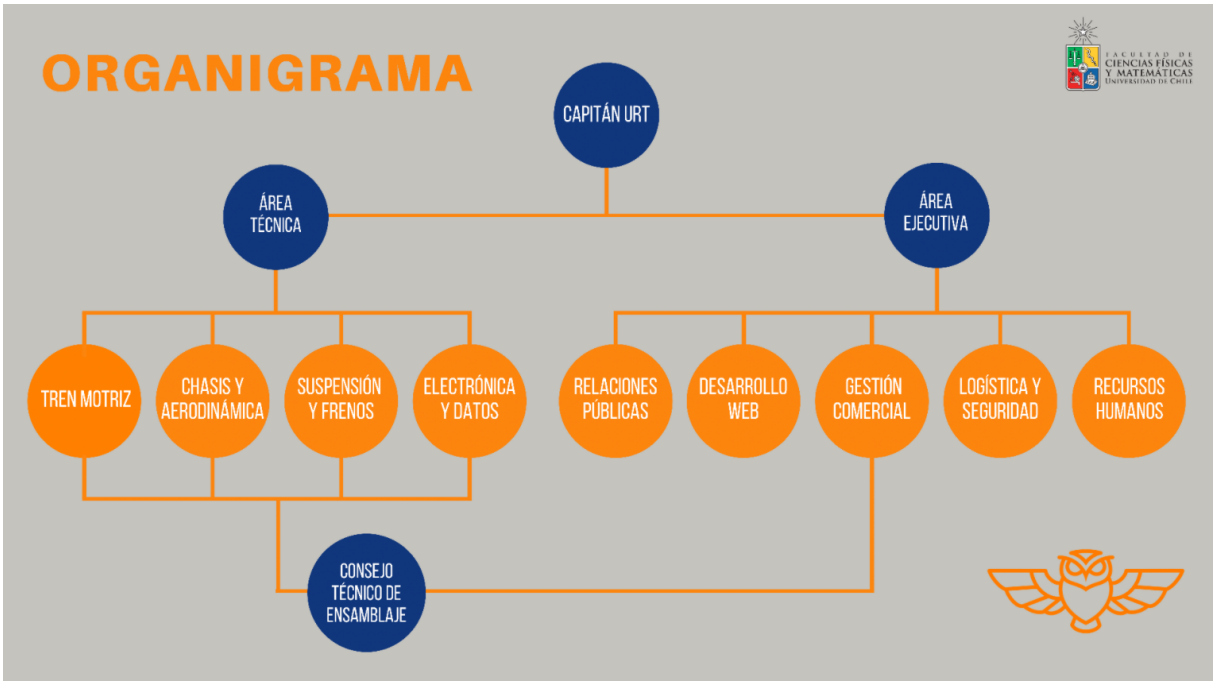

Figura 1: Organigrama de URacing Team

El área técnica de este equipo está a cargo de la construcción del auto y de que funcione de la mejor forma posible. Dentro de esta área hay sub-equipos, estos son: tren motriz, chasis y aerodinámica, suspensión y frenos y electrónica y datos. Esta memoria forma parte del sub-equipo de electrónica y datos, el cual está compuesto mayoritariamente por estudiantes de ingeniería eléctrica y de computación.

El sub-equipo de electrónica y datos se consolidó hacia abril de 2021. La tarea principal de electrónica y datos es diseñar y construir todo lo relacionado al sistema eléctrico del auto, incluyendo luces, pantalla, ramal eléctrico, conexiones entre sensores, el computador del motor, etc. Además de estar encargado de aspectos de telemetría, tales como comunicación y recopilación de datos del rendimiento del automóvil.

El equipo actualmente tiene en total de aproximadamente 45 integrantes, donde la mayoría son estudiantes de la facultad de ciencias físicas y matemáticas en la universidad, pero también hay estudiantes de la facultad de arquitectura y urbanismo. En el equipo de electrónica y datos hay alrededor de diez integrantes, los cuales la mayoría son estudiantes de ingeniería eléctrica y están asignados a distintos proyectos dentro del equipo, el cual uno de ellos es telemetría, donde actualmente solo se encuentra un integrante, el cual pertenece al departamento de ciencias de la computación. Este integrante soy yo, el rol permite plantear, diseñar y desarrollar este sistema, los comentarios y sugerencias del resto del equipo también son una gran ayuda para tener una perspectiva distinta y así hacer cambios, si es que es necesario, esto logró que se tuviera bastante trabajo, dificultad y contenido para realizar una memoria.

1.2. Problema, relevancia y desafíos

En el proyecto se requiere crear un sistema de telemetría que capture datos desde sensores en un auto, para que luego sean enviados, almacenados, procesados, visualizados y analizados. La telemetría es muy importante para el equipo, ya que, permitirá que distintos equipos de *URacing Team* puedan monitorear y optimizar diversos aspectos del auto. Algunas de estas áreas son, seguridad, rendimiento y recursos. La computación es una herramienta emergente en el mundo de los automóviles, ya que muchas partes del auto no son visibles directamente y se necesita tener información precisa sobre estas.

El desarrollo de un sistema de este estilo es algo que está ocurriendo en todas las áreas de deportes de motor. Por ejemplo, en el Fórmula Uno, unos de los deportes de motor más populares del mundo, es el líder en el área de innovación a nivel tecnológico con sus autos. Grandes empresas como *Amazon*, la cual le ofrece servicios de procesamiento y análisis de datos a la Fórmula Uno y a la *Scuderia Ferrari Formula 1* [4], influencia mucho en el desarrollo y en la evolución de estos autos a través de los años. *Oracle* es otro ejemplo de una empresa grande que ofrece servicios de análisis y procesamiento, eso sí, para uno de los equipos rivales, *Oracle Red Bull Racing* [5], al tener esta rivalidad y competencia, aporta a una aceleración en la innovación en estos ámbitos tecnológicos.

El equipo de URacing Team se está preparando para participar en un evento que será realizado en Michigan International Speedway [3], en los Estados Unidos, la cual se realiza dos veces al año. El 2022 hubo un evento del 18 al 21 de mayo y otro del 15 al 18 de junio, se espera que URacing Team tenga su auto disponible para participar en algunos de los eventos del 2023. Esto implica que el sistema de telemetría debe ser capaz de funcionar en cualquier lugar y bajo cualquier clima, es decir, extremo calor, frío, lluvia, viento, nieve, etc. Con estos obstáculos, tanto el hardware como el software del sistema deben resistir y superar cualquier situación, para que pueda cumplir su objetivo.

URacing Team es un equipo de estudiantes y es totalmente voluntario, esto hace que mantener a los integrantes motivados y trabajando constantemente sea un desafío, ya que muchos proyectos, como es este, dependen de que otras partes del auto avancen en paralelo. Hay varias personas involucradas con el proyecto, como son los auspiciadores del equipo y la universidad, para que el equipo reciba aportes de los auspiciadores, estos deben ser procesados por la universidad, lo cual hace que el proceso de obtención de materiales sea más lento. Estos son unos de los grandes desafíos que tiene el equipo de URacing Team y este proyecto en particular, pues se requiere del avance de otros proyectos y de la adquisición de materiales físicos.

A partir del obstáculo anterior, impide que el proyecto avance de forma estable y segura,

por eso, no se ha podido cumplir con el cronograma inicialmente programado, ya que existen muchas dependencias entre los integrantes del equipo. Esto implica que los objetivos han tenido que ser modificados para esta etapa del proyecto, a continuación, se explicará en qué consisten estos objetivos.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo general de la solución del problema a resolver es desarrollar un sistema, en donde se puedan enviar datos desde un auto de carreras hacia otra ubicación, para que luego puedan ser procesados, almacenados y visualizados por personas en URacing Team. Se necesita que sea en otra ubicación, ya que la visualización debe ser en tiempo real y como el equipo no puede estar en la cabina del auto con el piloto, se requiere que los datos puedan ser observados desde un dispositivo en otro lugar. Este sistema debe funcionar independiente del lugar en el mundo, tipo de clima o terreno, ya que no se sabe dónde podría estar corriendo el auto en el futuro.

1.3.2. Objetivos específicos

1. El estudio y diseño de un sistema que pueda transmitir, almacenar y visualizar datos en tiempo real es una parte crucial para este proyecto, por lo cual es un objetivo importante para que se desarrolle una solución de calidad.

2. Hay que investigar distintas alternativas de transmisión de datos que logren ser capaz de enviar información desde el auto de carreras en movimiento, hacia un dispositivo ubicado en otro lugar, esto requiere que la conexión entre el auto y el dispositivo sea inalámbrica. Luego de ser investigado, se debe elegir e implementar la solución, utilizando los recursos disponibles.

3. Utilizando el tipo de transmisión de datos elegida en el punto anterior, se debe desarrollar un sistema que logre recibir los datos enviados desde el auto a un dispositivo en otro lugar, esta recepción debe tener una conexión continua y con poca latencia, es decir, el tiempo en que los datos son enviados y recibidos debe ser de una magnitud aceptable.

4. Se deben estudiar los tipos de datos que tendrán que ser almacenados, para que luego se busque una manera para almacenar esta información de la mejor forma posible. Estos datos después deben ser utilizados para diversas tareas, como análisis, visualizaciones, registros, etc. Esta solución de almacenamiento también debe resolver problemas de volumen y respaldo de información.

5. Al igual que los otros puntos anteriores, se deben analizar distintos sistemas de visualización de datos, para que luego puedan ser aplicadas a este proyecto, el sistema finalmente elegido debe visualizar información en tiempo real, ya que se debe monitorear el estado del auto cuando el piloto esté a bordo.

6. Hay que realizar pruebas, mediante simulaciones, para verificar que cada parte del sistema funcione correctamente, estas simulaciones ayudaran con el desarrollo de una solución más completa y con menos errores.

2. Estado del arte

2.1. Estado inicial

Al iniciar este proyecto, hay componentes del auto ya existentes que no se pueden cambiar, los cuales afectan directamente al proyecto. Estos componentes corresponden a las partes que generan datos, tales como el motor del auto, el cual es de combustión interna, la ECU o *engine control unit* y una gran cantidad de sensores. Es bueno saber sobre estos componentes, pero para esta instancia del proyecto no es necesario explicarlos con tanto detalle, por esta razón están mencionados en los anexos de este documento.

Se había tomado la decisión de usar un protocolo de transmisión alámbrica entre microcontroladores dentro del auto llamado CAN Bus [6], en inglés *Controller Area Network* y Bus, en el ámbito de la informática, es un elemento que transmite una elevada cantidad de información. Este protocolo es usado ampliamente en el mundo automotriz e incluso en otras áreas como la aeroespacial, equipos médicos y automatización industrial [7].

2.2. Soluciones existentes

Se han analizado distintas soluciones al problema que se va a abordar, tanto desde otros equipos de Fórmula SAE, equipos de Fórmula Uno y otros sistemas adaptados para autos en general, los cuales son mencionados más adelante. En varias de estas soluciones el uso de CAN Bus está presente, entonces se analizarán las soluciones utilizando un microcontrolador como punto de partida del sistema.

2.2.1. Transmisión

Uno de los sistemas más complejos y con soluciones más diversas son los sistemas de transmisión inalámbrica de datos, estos varían tanto en hardware como en protocolos de transmisión. En el auto de *Mercedes-AMG Petronas' F1 Team* en la Fórmula Uno se usan frecuencias de radio para enviar datos encriptados desde un servidor en el auto, estos son enviados desde una antena y recibidas por el equipo. A la vez, también se usan microondas para enviar datos directamente al equipo, el cual se encuentra en el *garage*, al lado de los *pit stops*. Estas microondas sirven para enviar una ráfaga de información cuando el auto está en rango de esta ubicación, lo cual sirve como método de respaldo de información si es que por alguna razón los datos no son recibidos [8].

En el equipo de Formula SAE, AMZ (*Akademischer Motorsportverein Zürich*) [43], se descartan varios protocolos previamente estudiados y finalmente se elige un protocolo llamado WLAN, *Wireless Local Area Network* [9]. Este protocolo es uno de los más comunes en todo el mundo, pues es el que se usa para acceder a internet desde hogares, colegios, oficinas,

etc, también es comúnmente denominado WiFi. La elección de este protocolo se debe a que proporciona un rendimiento más alto, con la misma frecuencia de 2.4GHz que los demás protocolos de transmisión inalámbrica, los cuales son, Bluetooth, ZigBee y redes de celular 3G/4G.

Algunas empresas implementan productos para poder transmitir datos desde cualquier tipo de auto, venden un kit con el hardware necesario para poder implementarlo de la forma más simple posible. Peplink tiene un producto llamado BR1 Pro 5G [10], el cual es un router con 5G al cual se le pueden conectar distintos dispositivos de forma local, también, tiene la posibilidad de que se le inserte una tarjeta SIM y así poder conectarse a internet vía una red telefónica.

Bosch también tiene su propio producto especializado para deportes de motor, LTE65 modem [11], el cual es un modem parecido a BR1 Pro 5G, pero con la diferencia de usar LTE en vez de 5G. Con LTE hay bajas latencias y la transmisión de datos es extremadamente confiable, incluso a altas velocidades de vehículos y con coberturas de red inestables.

Existe un producto llamado *Ubiquiti Bullet M2 HP Outdoor* [47] que actúa como modem y puede generar su propia red. Este producto genera una señal de radio y se le puede conectar cualquier tipo de antena, gracias a su conector de radio frecuencia de tipo N. Su fuente de poder es por un cable de Ethernet, a esto se le llama PoE (Power over Ethernet), entonces al conectar el microcontrolador, este le da energía y al mismo tiempo este se integra a la red.

2.2.2. Almacenamiento

Un auto de carreras genera muchos datos, los cuales deben ser almacenados, visualizados y analizados. El tipo de datos que son generados se llaman series de tiempo [44] [45] [46], es decir, todo valor tiene asignado una instancia en el tiempo. Esto hace que se pueda ver la evolución de los datos en el tiempo, lo cual, para el caso de este proyecto, es sumamente importante.

Un ejemplo de una serie de tiempo, en relación con este proyecto, es la velocidad medida por los sensores, cada cierta frecuencia se toma la medida de la velocidad en ese instante, esto resulta en una tabla con dos columnas, el valor medido y el tiempo asignado. Para este proyecto se tendrá que trabajar con varias series de tiempo, cada tipo de valor medido tendrá su propio tiempo, no todas las series de tiempo son iguales, la diferencia de tiempo que hay entre valores seguidos puede o no ser constante, es decir, la toma de medidas puede variar en su frecuencia.

Las series de tiempo tienen la característica de que extrañamente los datos son modificados, ya que representan un instante en el pasado, los cuales siempre serán los mismos.

Estos datos pueden ser utilizados para predecir el comportamiento de lo que puede pasar en el futuro, para esto se usan distintas técnicas de *forecasting* o predicción. Algunos tipos de series de tiempo pueden tener la característica de ser periódicos, es decir, que cada cierto tiempo, ciertos patrones en los datos empiezan repetirse, esto puede ser por otros eventos no representados en la data, un ejemplo de estos periodos puede ser, un año, un día, una hora, una vuelta a la pista de carrera, etc. Aquí es donde se pueden realizar estudios analíticos comparando datos cíclicos y así encontrar patrones y conclusiones.

Encontrar donde almacenar estos datos no es difícil, cualquier sistema de administración de base de datos relacionales podría funcionar, tales como PostgreSQL, MySQL o SQL Server. Eso sí, filtrar y ordenar estos datos no es óptimo usando estos sistemas, es por esta razón que se buscan herramientas especializadas en series de tiempo.

Una de estas herramientas es *kdb+* de KX Systems, la cual es el proveedor de análisis de datos en tiempo real de *Alpine F1 Team* [12]. Esto implica que también se preocupa de facilitar la visualización y el análisis, de hecho también es utilizado por *MIT's Formula SAE Team* [13]. MIT (*Massachusetts Institute of Technology*) logró obtener un patrocinio con KX Systems y fue aplicado en su auto del 2020.

Hay varios sistemas de administración de base de datos y herramientas que se dedican específicamente a almacenar series de tiempo, por ejemplo, Prometheus, Graphite, Amazon Timestream, RRDTool, etc. Una de las mejores rankeadas es InfluxDB, a diferencia de alguna de las otras herramientas, InfluxDB es *open source* y gratuita. Una de sus mejores características es su capacidad de integración con otros sistemas, su API está disponible en varios idiomas de programación. Al comparar el rendimiento de consultas con Elastic Search, InfluxDB utiliza 9 veces menos memoria en disco y puede ser hasta 7,7 veces más rápida [44].

2.2.3. Visualización

La visualización de series de tiempo es muy importante, pues ver el transcurso de datos en una tabla puede ser difícil de analizar, estas visualizaciones deben ser capaz de ver la continuidad del tiempo, para esto usualmente se utilizan gráficos de línea o *scatter plots*, donde uno de los ejes es el tiempo. El poder comparar datos periódicos, también, es un desafío dentro de las visualizaciones, pues tener muchos ciclos en una sola visualización puede resultar en algo difícil de entender [48].

Como se ha mencionado anteriormente, hay herramientas que ofrecen servicios de visualización de series de tiempo con sus propios ambientes y restricciones. Si se quisiera tener total libertad para poder crear visualizaciones lo mejor sería usar *d3.js*, una librería de JavaScript en la cual se puede programar todo tipo de visualizaciones, Una herramienta como ésta es

ideal para tener total libertad para crear visualizaciones, pero *d3.js* no es la única, sino que también hay librerías en muchos otros idiomas de programación como Python o Java, aunque *d3.js* es ideal para subir el trabajo a una aplicación web.

A veces soluciones con mucha programación no son lo mejor para un equipo multidisciplinario como es URacing Team, esto se debe a que existe mucha dependencia entre integrantes más especialistas en sus áreas y el analista de datos, una herramienta que tienen un ambiente más amigable para generar visualizaciones sería ideal para que equipos de otras especializaciones puedan generar sus propios reportes y análisis. Una de estas herramientas es Grafana [27], esta es especializada en visualizar series de tiempo, pero a diferencia de las demás herramientas, esta es *open source* y muy fácil de conectar con distintas fuentes de datos como las mencionadas anteriormente.

Una de las características más llamativas de Grafana es la posibilidad de crear *plugins*, es decir, un programa externo que puede ser integrado a Grafana, para que luego pueda ser utilizado. Esta característica es posible, ya que Grafana es *open source* y le da tanto flexibilidad como sencillez a la plataforma. Otras herramientas con interfaces amigables que se podrían utilizar para visualizar datos son, Tableau, PowerBI de Microsoft, Data Studio de Google, entre otros, eso sí, estas no son *open source* y no tienen la fluidez que dispone Grafana para actualizar tan rápidamente sus visualizaciones. Estas otras herramientas tampoco tienen la facilidad de exportar y publicar sus visualizaciones, para que sean visualizadas por varias personas, de forma gratuita.

Existen abundantes servicios que ofrecen almacenamiento, procesamiento, visualizaciones y análisis de datos, los cuales pueden ser aplicados para distintos mercados, algunos de estos servicios son los que ofrece Amazon, Oracle y *kdb+*. Como es mencionado anteriormente, varios equipos de Fórmula Uno y algunos de Formula SAE, utilizan estos servicios para optimizar su auto y su equipo. Estos servicios probablemente tengan una solución mejor desarrollada para este proyecto, eso sí, tiene ciertas características que podrían resultar negativas para este proyecto en particular, uno de ellos es el estrecho límite que hay con las reglas, ya que dice que el auto debe ser desarrollado por estudiantes y no por otras personas. Otro aspecto negativo que podrían tener estos servicios es el gran costo que estos demandan, se podría pedir que auspicien al equipo con sus servicios, eso sí, sumarían al costo de una de las pruebas estáticas en la que participará el auto.

3. Solución

3.1. Arquitectura

El sistema que será desarrollado está dividido en varios subsistemas, estos parten desde la generación de los datos, los cuales corresponden a los sensores dentro del auto, para que luego de una serie de procesos lleguen a un sistema final que visualiza en tiempo real los datos generados. En la figura 2 se pueden ver los distintos subsistemas separados por su ubicación física, los cuales pueden estar tanto en el auto o en un computador externo y estático fuera del auto. Tanto el subsistema de sensores y CAN Bus presentes en la figura 2 no serán desarrolladas en esta instancia del proyecto, ya que será desarrollado por otras personas del equipo con más experiencia en el área eléctrica. Es por esta razón y el hecho que aún no ha sido desarrollado que están marcadas con un color rojo, eso sí, es necesario mencionarlos para poder entender mejor la arquitectura del sistema de manera más completa.

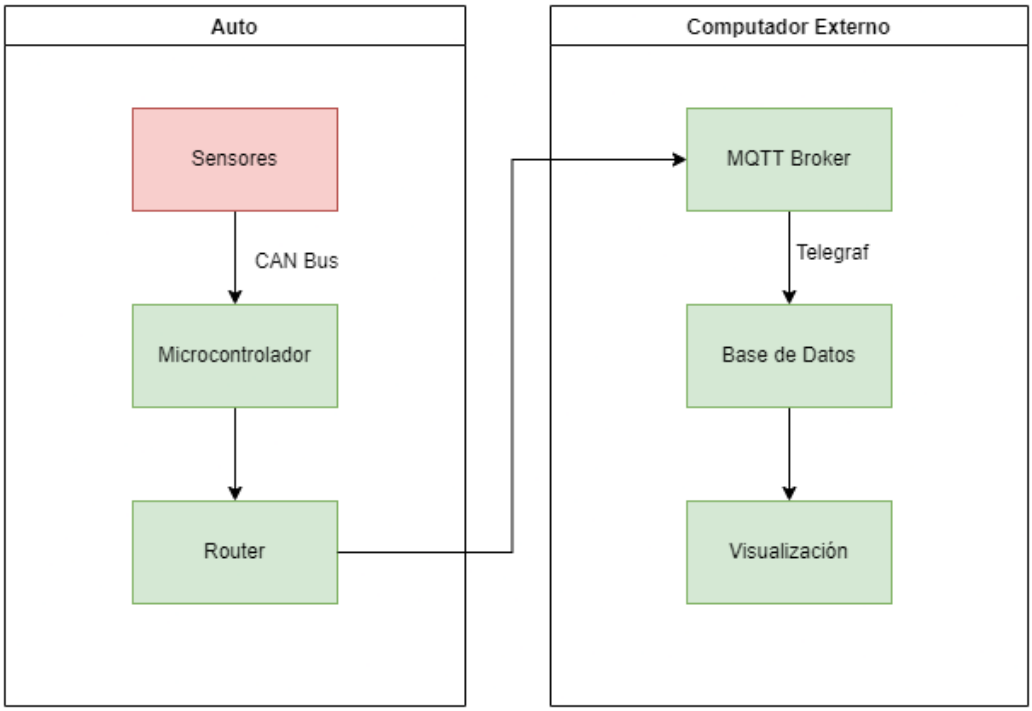

Figura 2: Arquitectura del sistema de telemetría.

El proceso que será desarrollado en esta instancia del proyecto inicia con un microcontrolador, en el cual se ejecutará una simulación que busca imitar la forma en que los datos serán exportados del auto hacia un computador externo. En este microcontrolador se asume que los datos generados por los sensores y enviados a través de CAN Bus ya están disponibles, por lo tanto, solo se preocupa de enviar los datos. Se utilizará un router que generará una red a la cual tanto el microcontrolador como un computador se conectarán de forma local, es decir, sin la necesidad de utilizar internet. Este router permite la conexión entre el microcon-

trolador y el computador, las características de este router determinan el rango máximo de la conexión, velocidad de transmisión de datos y la estabilidad de la conexión entre los aparatos.

Una vez que la conexión esté establecida, hay varios subsistemas que solo dependen de software, estos son los que se encuentran presentes en el computador externo. Uno de los procesos es la forma en que los datos son transmitidos desde un dispositivo, el microcontrolador, a otro dispositivo, el computador externo. Para poder enviar información a través de la red generada por el router, se utiliza un protocolo de transmisión de información llamado MQTT (*Message Queue Telemetry Transport*). Es un protocolo de transporte de mensajería basado en publicación/suscripción y es extremadamente ligero, lo cual es ideal para conectar dispositivos remotos con un espacio de código pequeño y un ancho de banda de red mínimo. Esto permite programar el microcontrolador en el auto fácilmente, ya que, usa poca memoria en el microcontrolador. El protocolo MQTT depende de un núcleo al cual los dispositivos pueden publicar mensajes o suscribirse a ellos, este "núcleo"se denomina un MQTT Broker. Este es el destino hacia el cual los datos del microcontrolador serán enviados, esto permite que el mantenimiento de las bases de datos sea mucho más fácil, esto será explicado más adelante.

A partir del MQTT Broker, los datos son enviados a una base de datos en el computador externo, esto se hace con una herramienta llamada Telegraf. Esta herramienta, la cual será explicada en más detalle en la solución del problema, se preocupa de suscribirse a los mensajes publicados por el microcontrolador, para que luego sean ingresados a la base de datos en el computador. La base de datos recibirá la información enviada por el microcontrolador en tiempo real, es decir, Telegraf le estará insertando información a la base de datos múltiples veces y en un intervalo de tiempo pequeño.

El último subsistema del proceso es la visualización, esta etapa también es desarrollada en el computador externo y tendrá una conexión directa con la base de datos. Las visualizaciones serán en tiempo real, por lo tanto, utilizarán la pantalla del computador o un monitor conectado a el computador. Aquí se verán los resultados del proyecto, visualizar la simulación de datos generados por el microcontrolador es el alcance de esta instancia del proyecto, los datos reales generados por los sensores pueden ser distintos a los que utilizarán en la simulación, por lo tanto, el estado final de las visualizaciones sirve de referencia para trabajos futuros.

3.2. Metodología de trabajo

Para llevar a cabo el desarrollo de la solución, se tiene que establecer una metodología de trabajo, que logre avanzar con el proyecto de tal forma que se llegue a la meta dentro de los plazos establecidos por la universidad. En URacing Team cada sub-equipo tiene su propia metodología de trabajo, los cuales pueden ir variando en el tiempo, en el equipo de electrónica y datos, siempre se ha tratado de mantener la misma metodología. Esta metodología consiste en reuniones semanales, de forma virtual, en donde se realizan varias tareas, una de ellas es

actualizar al equipo con el trabajo desarrollado durante la semana, luego hay otra etapa de la reunión, donde otras personas pueden aportar con comentarios y así luego se verificar que el desarrollo no afecte de forma negativa el progreso de otros proyectos dentro del auto. Estas reuniones pueden variar en duración, a veces pueden ser cortas y concisas, pero a veces pueden durar varias horas tratando de resolver problemas en conjunto o estableciendo las bases para iniciar un nuevo proyecto. Para cada reunión se realiza un acta, donde se mencionan las personas que asistieron, los temas a tratar durante la reunión, apuntes con comentarios o datos relevantes y las tareas que se realizarán para la próxima reunión. Las reuniones no siempre se realizan todas las semanas, a veces, cuando hay mucha carga académica en el semestre, se discute entre el equipo, si es que es conveniente realizar la reunión, ya que muchos de los integrantes no han avanzado lo suficiente con sus proyectos, por ende, las reuniones se aplazan en una semana.

La solución que se desarrolla para este proyecto se divide en tres etapas, estas consisten en el sistema de almacenamiento, visualización y transmisión de datos, donde la última también está dividida en dos sub-partes, envió y recepción de datos. En la figura 2, donde se muestra la arquitectura de la solución, la base de datos corresponde al sistema de almacenamiento, la visualización trivialmente hace referencia al sistema de visualización, el MQTT Broker representa a la recepción de los datos y tanto el microcontrolador como el router forman parte del envío de datos. A cada una de estas etapas se les busca una solución que logre satisfacer los objetivos específicos propuestos, aparte de que estos sistemas logren su objetivo, estos deben funcionar en conjunto para que se logre el objetivo general.

El desarrollo de cada etapa fue realizado en un orden, el cual tiene el objetivo de avanzar en el proyecto de forma óptima para lograr el objetivo dentro de los plazos establecidos. Este orden busca depender lo menos posible de factores externos, como otras personas, sistemas externos o recursos físicos, es por esta razón que se decide iniciar el proyecto desarrollando todo el software de la solución, esto abarca la visualización, almacenamiento y recepción de datos. En la figura 3 se puede ver el orden en que de decide desarrollar la solución, las etapas de color amarillo corresponden a las partes de la solución que dependen mayoritariamente de software, el cuadro azul tiene una dependencia de hardware, el cual, al momento de iniciar el proyecto aun no estaba disponible, por ende se implementa en la parte final del desarrollo.

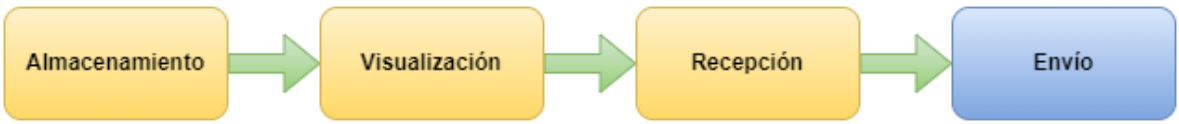

Figura 3: Orden en que se desarrolla la solución.

Se inicia con el almacenamiento, ya que, la visualización depende de una fuente de datos y parte de la recepción necesita almacenar los datos recibidos en algún lugar. La recepción de datos no depende de hardware, pero si requiere de un conocimiento teórico de las tecno-

logías físicas que serán utilizadas en el proyecto, las cuales corresponden al tipo de red de transmisión y los tipos de sensores, las cuales ya han sido definidas, pero se decide avanzar con la visualización hasta tener un plan más concreto sobre el hardware que será utilizado. La visualización solo depende de software y puede ser desarrollada sin dependencias, a excepción de las visualizaciones finales que serán utilizadas por el equipo, las cuales no forman parte del alcance de esta solución. Una vez desarrollada la visualización y con los protocolos de transmisión y hardware definidos, se desarrolla el sistema de recepción de datos, el cual todavía depende solo de software para que sea desarrollado.

La siguiente etapa es el sistema de envió de datos, aquí se necesita estar en posesión del microcontrolador y router que serán parte del auto en su versión final, para conseguir estas piezas, el equipo de gestión comercial debe comunicarse con auspiciadores o con el grupo ejecutivo del equipo para poder adquirir estos elementos. Por esta razón, se decide desarrollar esta etapa al final del proceso, ya que le da tiempo al equipo de gestión a conseguir lo requerido y así poder desarrollar el sistema en su forma final y no solo un prototipo. Al tener todos estos sistemas terminados se puede avanzar con la evaluación del sistema, realizando simulaciones y así observar cuales son los límites y mejoras que se pueden desarrollar a futuro.

3.3. Diseño y desarrollo

Teniendo la arquitectura y la metodología que serán utilizadas para desarrollar la solución, esta puede empezar a ser implementada. Como se puede observar en la figura 3, las distintas etapas están divididas como secciones en esta parte del documento, manteniendo el orden en que estas fueron desarrolladas, de esta manera se puede seguir el desarrollo de la solución en orden cronológico.

3.3.1. Almacenamiento

Lo primero que se definió fue el sistema de gestión de base de datos, el cual se eligió a partir de los estudios realizados sobre los sensores, conociendo la forma en que se envían datos en otros sistemas de telemetría y sabiendo que se iba a estar trabajando con series de tiempo. Esto guió el estudio a encontrar una herramienta llamada InfluxDB, creada por la compañía Influx Data, la cual es un sistema de gestión especializada en bases de datos que almacenan series de tiempo.

La forma en que InfluxDB almacena esta información puede ser vista como una tabla, pero en realidad se usa una tecnología NoSQL, esto permite que las consultas y almacenamiento sea mucho más eficiente. Pareciera que en InfluxDB se utilizan tablas, pero en realidad estas son reemplazadas por estructuras llamadas *measurements*, a cada *measurement* se le asignan *fields*, donde se almacenan los valores medidos y *tags*, las cuales pueden tener un valor categórico. Todo *measurement* debe tener al menos una serie de tiempo, es decir, un *field*, con su

tiempo asignado, al agregar más *fields*, implica que el *measurement* almacena más series de tiempo, al agregar un *tag*, también se agregan más series de tiempo, ya que *field* y *tag* serían las llaves primarias.

Para este proyecto, se inició utilizando un solo *measurement*, al cual se le asigna cada valor medido como un *field*, posteriormente se cambió éste para que cada valor medido tenga su propio *measurement*, esto se hizo solo para facilitar la importación de datos en las visualizaciones, que más adelante serán explicadas. En la figura 4 se puede ver unos ejemplos de las estructuras de bases de datos utilizadas en el proyecto, a la izquierda se ve la estructura inicial, donde en verde se ve el nombre del *measurement*, en amarillo el tiempo asignado y en azul todos los *fields* asignados al *measurement*. Posteriormente, la estructura de la derecha se puede ver que a cada *field* se le crea su propio *measurement*, acá se puede ver el ejemplo de un solo *field*, *speed*.

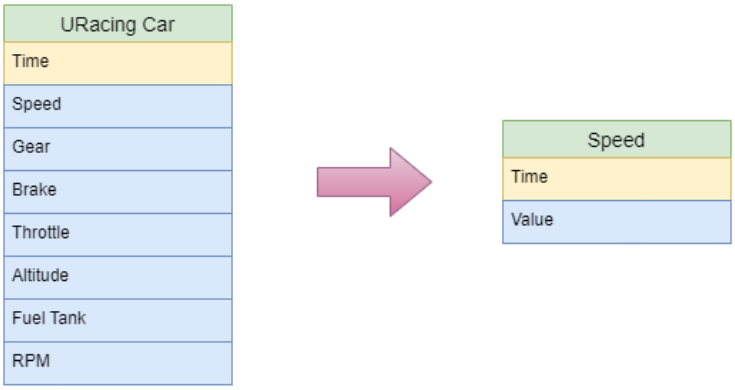

Figura 4: Ejemplo de estructuras de base de datos utilizadas en el proyecto.

Se decidió instalar y habilitar las bases de datos en un contenedor de Docker [17], en esta plataforma de software se permite construir, testear y ejecutar aplicaciones rápidamente, crea un ambiente que permite ejecutar y escalar código en cualquier sistema operativo. Esto permite que independiente de la máquina con que se esté trabajando a futuro, solo se necesita que tenga instalado Docker para que pueda ejecutar los programas necesarios. Una de las formas para crear contenedores de Docker, es usando un archivo llamado *docker-compose* [18], aquí se permite definir y ejecutar múltiples contenedores para una aplicación. Esto se hace mediante un archivo de tipo YAML, donde se escriben todas las configuraciones de los servicios de la aplicación, para que luego ejecutando un solo comando este cree e inicie la aplicación.

A partir de esto se creó un archivo de *docker-compose*, en el cual se configura y especifica que se usará la imagen de InfluxDB en su versión 1.8 [19], también se especifica que estará conectada al puerto 8086 y se crea un usuario que tenga permisos para poder modificar las bases de datos. Por defecto, un contenedor de Docker tiene un límite de 10 gigabytes de

memoria, lo cual por el momento es suficiente para poder crear simulaciones.

Una vez ya creado el archivo *docker-compose*, este se ejecuta y se habilita InfluxDB en un computador con Windows 10, que fue facilitado por uno de los auspiciadores del equipo, el cual podría actuar como el computador externo mencionado en la arquitectura de la solución. Luego se familiariza con InfluxDB, creando tablas, insertando valores y realizando consultas, esto sirve para que a continuación se puedan realizar pruebas de simulación, insertando grandes cantidades de datos. Para probar la primera base de datos, se buscó en internet por un *dataset* que tenga datos similares a los que podrían ser generados por los sensores del auto de URacing Team. En esta búsqueda es donde se encontró un *dataset* [16] generado con datos de *F1 2020* de Codemasters, un juego de simulación de autos de Fórmula Uno disponible en consolas de juegos como PlayStation o Xbox. Los datos generados son lo suficientemente detallados y realistas para que pueda ser utilizado como la fuente para la simulación final del proyecto, eso sí, se filtró la data total por los datos de un solo piloto y se utilizan columnas con datos más realistas con respecto a este proyecto, en la tabla 1 se pueden ver algunos datos interesantes con respecto al *dataset* filtrado que será utilizado.

Tabla 1: Análisis del *dataset*

Columnas	19
Filas	59.032
Minutos	52 (aprox.)
Vueltas	35
Tamaño	25,7 MB (aprox.)

Para ingresar los datos a la base de datos en InfluxDB, se crea un archivo de Python, donde se busca filtrar, ordenar, agrupar e ingresar los datos del *dataset* de tal manera que simule un envío de datos en tiempo real. Aquí, se supone que los datos serán enviados cada segundo, esto implica que hay que agrupar los datos generados en el segundo que fueron generados para que luego sean insertados. En esta simulación se utilizó la librería de InfluxDB para Python, la cual viene con todos los métodos para que se pueda conectar, insertar y consultar a una base de datos en InfluxDB. Los resultados fueron positivos, los valores de los datos ingresados no sufrieron modificaciones y el tiempo asociado a los datos fueron los adecuados.

El siguiente paso es analizar donde almacenar los datos potenciales del auto, esto se refiere a la situación en que los datos son demasiados para ser almacenados en un computador personal. Para esto se usan los servicios de Digital Ocean para crear una máquina virtual gratuita en uno de sus servidores, se creó otra base de datos con el archivo *docker-compose* y se realizó la misma prueba anterior, pero ahora con la base de datos montada en un servidor. Los resultados de las pruebas fueron positivos y confirma que el archivo de *docker-compose*

funciona en otras máquinas. *Digital Ocean* es un servicio pagado, que permite pruebas gratuitas a universitarios hasta que el uso de sus servicios no supere los 100 USD. Se decidió buscar un auspiciador que ayude al equipo con el almacenamiento de datos, ya que almacenar datos en dispositivos personales no tiene la seguridad de tener suficiente almacenamiento o respaldo de información que es requerida en este proyecto.

3.3.2. Visualización

Para visualizar la información de las bases de datos, se tendrían que usar técnicas específicas para visualizar series de tiempo. Para esto se estudiaron distintas técnicas para visualizar datos temporales tanto para automóviles como para otras áreas, se tomaron como referencia visualizaciones utilizadas en la Fórmula Uno y otros eventos de deportes de motor. En la figura 5 se puede ver una imagen del *pitwall* de la *Scuderia Alpha Tauri*, un equipo de Fórmula Uno, en una carrera del 2020. En el *pitwall* se ubican personas que toman decisiones importantes en cada equipo, estas personas deben ver data en tiempo real y debe mostrar información de tal manera que sea fácil de analizar, para que luego se pueda actuar de manera rápida con respecto a problemas o cambios de estrategia.

Figura 5: Alpha Tauri pitwall

Tomando en cuenta las distintas alternativas para poder visualizar estos datos, se decidió que Grafana es una buena opción para aplicar en esta etapa del proyecto. Grafana fue elegida por diversas razones, una de ellas es que permite establecer una conexión a InfluxDB de una manera muy sencilla, ya que tiene una interfaz gráfica amigable, intuitiva y con mucha documentación. Otra razón es que tiene una imagen en Docker, lo cual permite que se pueda agregar al archivo de *docker-compose* junto con InfluxDB. Para la configuración de Grafana en el archivo se utilizó la versión más actual, se le asigna el puerto 3000, se crea un usuario para poder entrar a la plataforma y se crea un *network* de Docker, el cual permite que contenedores tengan una conexión disponible entre sí.

Al ejecutar el archivo de *docker-compose*, se ingresa a la interfaz de Grafana utilizando cualquier navegador web e ingresando al puerto 3000. Desde aquí, hay tutoriales hechos por Grafana para poder crear las primeras visualizaciones, pero antes de eso hay que especificar una fuente de datos. Realizando el tutorial para esto, rápidamente se puede conectar al puerto 8086, donde se encuentra InfluxDB, se elige la base de datos a la que se quiere conectar, se ingresan las credenciales de InfluxDB y la conexión estará lista. Para crear las primeras visualizaciones, se utilizan los datos insertados a InfluxDB en las pruebas realizadas anteriormente, se eligen datos específicos y siguiendo los tutoriales, se crean las primeras visualizaciones de forma muy rápida.

Figura 6: Visualización inicial en Grafana

En la figura 6 se pueden ver las primeras visualizaciones realizadas en este proyecto, aquí se puede ver un *dashboard* con siete gráficos, todos estos gráficos muestran el mismo intervalo de tiempo, el tiempo en que se da una vuelta a un pista de carrera, pero con distintos valores medidos, los cuales corresponden a la marcha, altura, velocidad, nivel de combustible, rotaciones por minuto, frenos y acelerador de un auto. Entrando en un poco más detalle, uno de los valores medidos es el *Fuel Tank*, estanque de combustible, el cual muestra la cantidad de combustible que va quedando en el auto, se puede ver que en algunas partes de la pista se utiliza más combustible que en otros lados. Observando el gráfico de *Altitude*, altura, justo arriba del estanque de combustible, se puede ver que hay un tipo de cerro por el cual el auto tiene que pasar. Analizando ambos gráficos de *Fuel Tank* y *Altitude*, se puede ver que hace sentido que se utiliza más combustible en ciertas partes de la pista, ya que se debe subir un cerro, lo cual requiere más energía desde el motor. Este es un ejemplo de análisis que pueden realizar los especialistas del equipo de URacing Team al utilizar estas visualizaciones, estos especialistas, pueden ser de motor, aerodinámica, etc., serán los usuarios de esta solución, las visualizaciones futuras serán diseñadas para ellos.

Se puede observar que todas las visualizaciones son gráficos de línea, donde el eje horizontal representa el tiempo, esto se debe a que se está trabajando con series de tiempo y Grafana usa este método de visualización por defecto. Eso sí, hay varios otros tipos de visualizaciones disponibles, Grafana dispone de 21 tipos de visualizaciones en su instalación básica, pero en internet hay varios *plugins* que se pueden instalar en la plataforma, ingresándolos al archivo de *docker-compose*. Estas visualizaciones a la vez se van actualizando automáticamente cuando se ejecuta el código de prueba realizado en Python para InfluxDB. Para ser una de las primeras visualizaciones, Grafana ha mostrado que tiene mucho potencial para que siga siendo la principal herramienta para visualizar información en tiempo real.

3.3.3. Transmisión

Esta parte del proyecto se dividió en dos partes, una de ellas es el tema de ver cómo se envían los datos desde el auto y otra es cómo se reciben los datos en el computador externo, pero primero hay que establecer tecnologías generales para ambos. Se eligió el protocolo MQTT (*Message Queue Telemetry Transport*) [21] pues es un protocolo de transporte de mensajería de publicación/suscripción que es extremadamente ligero, lo cual es ideal para conectar dispositivos remotos con un espacio de código pequeño y un ancho de banda de red mínimo. Esto permite programar el microcontrolador en el auto fácilmente, ya que, usa poca memoria. Una de las partes más importantes de este servicio de publicación/suscripción es el núcleo hacia el cual se publica y suscribe. Como ya ha sido mencionado anteriormente, a este "núcleo"se le denomina un MQTT Broker, el cual ordena los mensajes por tópicos, para que luego los mensajes puedan ser ingresados y enviados. Cada vez que se publica un mensaje, este debe llevar un tópico, el cual puede tener una jerarquía, es decir, puede haber tópicos dentro de otros tópicos. Esta característica es muy útil para poder ordenar datos desde el auto, ya que a estos tópicos es a lo que se debe suscribir para que luego sean recuperados y utilizados.

Para desarrollar un MQTT Broker, se utilizó una herramienta llamada Mosquitto [28], hay muchos MQTT Brokers disponibles en internet, pero no es confiable para este proyecto usar un *broker* público, ya que, no se saben todos los distintos tópicos que este pueda tener, cuanto es el tráfico de datos que sostiene y también se debe depender de una conexión a internet, lo cual puede ser problemático dependiendo de la ubicación geográfica. Mosquitto permite desarrollar y configurar un *broker* con las características necesarias para que satisfaga las necesidades de este proyecto. Un Mosquitto MQTT Broker al igual que InfluxDB y Grafana, tiene una imagen de contenedor en Docker, lo cual permite agregar su configuración al archivo original de *docker-compose*. Para agregar Mosquitto al archivo *docker-compose* se utiliza su versión de imagen 2.0 y que utilice ambos puertos 1883 y 9001, también requiere de un archivo externo de configuración llamado *mosquitto.conf*, donde se especifica que escuche a publicaciones por el puerto 1883 y que cualquiera pueda publicar tópicos.

Ahora se puede ejecutar el archivo *docker-compose* para realizar pruebas sobre el nuevo Mosquitto MQTT Broker. Para esto se utiliza el *dataset* con el que se han realizado pruebas anteriores y se crea un archivo de Python, junto con una librería llamada paho-mqtt desarrollada por la empresa *Eclipse*, para publicar datos al puerto 1883, donde está escuchando el *broker*. Utilizando los métodos que ofrece la librería paho-mqtt para publicar y suscribirse a mensajes, se verifica que el *broker* funciona correctamente, al tener dos archivos de Python ejecutados en paralelo, uno publicando y otro suscribiéndose a mensajes, se confirma que, al publicar un mensaje desde un archivo, este aparece rápidamente en los resultados del código suscriptor, eso sí, esta librería de Python solo funciona para archivos ejecutados en Windows. Lo que sigue es implementar los procesos de envío y recepción de datos hacia y desde el Mosquitto MQTT Broker.

3.3.3.1. Recepción

Este proceso consiste en crear una conexión entre el Mosquitto MQTT Broker y una base de datos en InfluxDB, para esto se realizó una investigación y se descubrió una nueva herramienta: Telegraf [23], también creada por Influx Data, logra satisfacer la función de intermediario entre InfluxDB y diversas fuentes de datos, a este tipo de herramientas se les denomina un *server agent*. Una de las fuentes de datos a la cual Telegraf puede establecer una conexión es a MQTT Brokers, esto implica que la conexión entre el Mosquitto MQTT Broker e InfluxDB es posible y fácil, gracias a la compatibilidad que hay entre Telegraf e InfluxDB, ya que son productos de la misma empresa, Influx Data.

Entrando en más detalle, Telegraf, al igual que las otras herramientas utilizadas, tiene una imagen en Docker [24]. Al agregar la configuración de este contenedor al archivo de *docker-compose*, se especifica que se utilizará la versión 1.18 de la imagen en Docker. De la misma manera que el Mosquitto MQTT Broker, Telegraf también requiere de un archivo de configuración aparte, este lleva el nombre *telegraf.conf*, aquí se especifica que se debe suscribir a todos los tópicos en el puerto 1883 (*broker*) y luego insertar los mensajes en el puerto 8086 (InfluxDB). Utilizando la misma prueba que se utilizó para publicar datos a el Mosquitto MQTT Broker, se espera ver resultados en las visualizaciones de Grafana, ya que la prueba de conexión entre InfluxDB y Grafana ya fue aprobada. Los resultados no fueron exitosos, los datos publicados no fueron visualizados, luego verificando la base de datos creada por Telegraf, llamada *telegraf*, esta estaba vacía.

Estudiando el error se encontró que la estructura de los mensajes a los cuales Telegraf se suscribe deben tener una estructura específica. Si es que se envían mensajes con la estructura predeterminada en la documentación, los mensajes sí son recibidos por la base de datos, a este tipo de envió se le denomina *line protocol*, en donde solo se puede enviar un dato a la vez. Para enviar mensajes con más información, la estructura del mensaje debe ser configurada

en el archivo *telegraf.conf*, para este proyecto se usa una estructura de *JSON Array*, donde cada valor en el arreglo es un dato.

Figura 7: Ejemplo del mensaje que es enviado al MQTT Broker.

En la figura 7, se puede ver un ejemplo de la forma que llevan los mensajes que son enviados al MQTT Broker, en este ejemplo se puede ver que el mensaje enviado lleva tres datos, estos están compuestos por tres partes, *measurement* es la tabla en la base de datos en que el dato será ingresado, en este caso velocidad, rotaciones por minuto y la marcha que lleva el auto, *value* es el valor del dato, en este caso 200 kilómetros por hora, 5 marcha y 7.000 rotaciones por minuto, la unidad de medida depende del sensor utilizado. Finalmente *time* es el tiempo en que el dato fue generado, el cual será el tiempo que acompañará al dato, como es característico en series de tiempo, en este caso se usa el tiempo en el formato *unix*, este formato representa la cantidad de segundos que han pasado desde el instante que inició el año 1970 en el tiempo universal coordinado (UTC), para este proyecto se requiere precisión, por lo tanto, también se incorporan los microsegundos de este.

Se modificó el código inicial con la prueba en Python para publicar datos con la nueva estructura requerida por Telegraf. Esta estructura se debe a que Telegraf debe procesar los datos para que sean insertados a InfluxDB con el tipo de dato adecuado y sin pérdida de información. Al realizar las pruebas nuevamente, con el nuevo código, los resultados fueron exitosos, se pueden ver las visualizaciones de Grafana actualizándose automáticamente con la información adecuada. En la figura 8 se puede ver el proceso que está funcionando hasta esta instancia. Los mensajes están siendo recibidos desde el Mosquitto MQTT Broker mediante Telegraf hacia InfluxDB, para que luego Grafana tome los datos y muestre la información en tiempo real mediante técnicas de visualización específicas para series de tiempo.

Figura 8: Diagrama del sistema de telemetria dockerizado

Todo este proceso corresponde al software en el computador externo mencionado en la arquitectura de la solución, todo está presente en un archivo de *docker-compose* y puede ser ejecutado en cualquier computador con Docker instalado. Dentro del archivo de *docker-compose* se tuvo que crear un orden en que los contenedores son activados, ya que como algunos contenedores dependen de la apertura de otros puertos, se tuvo que especificar en el archivo. Hubo muchas veces que las pruebas no funcionaron, debido a que las versiones de las imágenes de Docker no eran compatibles entre sí, por esta razón se especifican las versiones de algunos de los contenedores del sistema.

3.3.3.2. Envío

Primero se eligió utilizar un Ubiquiti Bullet M2 HP Outdoor, el cual se puede ver en la figura 9, su función es actuar como router para generar una red y transmitir datos desde el auto. Esto se debe a que este producto es una inversión de bajo costo y de alto rendimiento comparado con las otras alternativas analizadas. El hecho de que este producto es la versión *outdoor*, da confianza a que sea resistente al agua, ya que, está compuesto por aluminio, un material robusto, esto también le da la resistencia para soportar la fuerza G generada por un auto a grandes velocidades. El hecho de que se le pueda conectar cualquier tipo de antena con un conector RF de tipo N significa que puede alcanzar rangos de distancia de múltiples kilómetros. Esto también se debe al factor de que transmite datos mediante ondas de radio, las cuales tienen una baja frecuencia, por lo tanto, utilizan poca energía para alcanzar grandes rangos de transmisión.

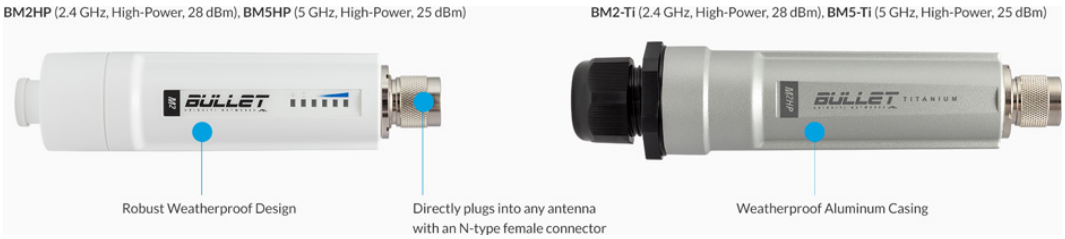

Figura 9: Ubiquiti Bullet

En la figura 10 se puede ver otro dispositivo que se disidió utilizar para el proyecto, un

Teensy 4.1 Development Board [25] será el aparato que actuará como microcontrolador, esté publicará los datos de los sensores hacia el Mosquitto MQTT Broker. Se decidió utilizar este microcontrolador, ya que tiene los suficientes pines para que se le pueda conectar una gran cantidad de sensores, lo cual le da tranquilidad al equipo para agregar los sensores que estimen necesarios. Otra razón por la que se eligió este microcontrolador es que se le puede conectar un cable Ethernet. Este cable es necesario para que el microcontrolador se conecte a la red generada por el Ubiquiti Bullet, pero más que eso, el cable será por donde se alimentará de energía el Ubiquiti Bullet, esta tecnología se llama PoE (*Power Over Ethernet*). Esta decisión fue tomada junto con el equipo eléctrico que debe desarrollar la distribución de los cables dentro del auto.

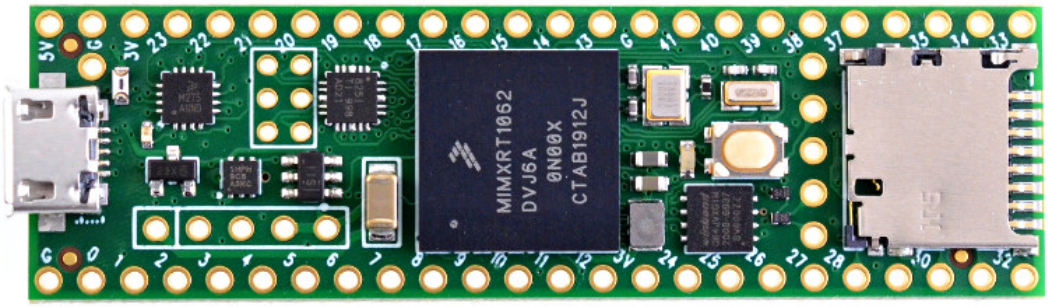

Figura 10: Imagen de Teensy 4.1 Development Board

Al llegar a esta instancia del proyecto, como equipo de URacing Team se decidió invertir capital y recursos en otras áreas más críticas del auto, como en la construcción del chasis y en ensamblar el motor. El área de electrónica y datos vio que se debía ajustar a este cambio y decidió dejar en pausa el desarrollo del hardware del sistema de telemetría, ya que estaba bastante avanzado y los recursos eran necesitados en otras áreas. Aquí es donde se tomó la decisión de utilizar herramientas disponibles y no las requeridas inicialmente. Se reemplazó el microcontrolador por un Sparkfun ESP8266 Thing - Dev Board, este microcontrolador no tiene conexión a Ethernet, pero si tiene una pequeña antena que le permite conectarse a una red. El Ubiquiti Bullet es difícil de reemplazar, ya que es un aparato con características más específicas, por lo tanto, se utiliza un router de red WiFi, común en hogares para poder conectarse a internet.

Para escribir el código en el ESP8266, se utilizó el lenguaje de programación de Arduino, junto con una librería llamada PubSubClient, esta librería se utiliza para publicar y suscribirse a tópicos en protocolos de mensajería MQTT. De esta manera se genera un programa que publica mensajes a él *broker* desarrollado anteriormente cada 5 segundos a través de la red generada por el router. El ESP8266 logra enviar tópicos de prueba exitosamente, los mensajes pueden ser visualizados en Grafana y pasan por todas las herramientas utilizadas, tanto en el software como en el hardware. Lo que sigue es hacer simulaciones y ver los límites de esta solución para poder validar el proyecto.

Todo el desarrollo de la solución está ubicado en un repositorio en la cuenta de GitHub de URacing Team, este repositorio es público y el enlace a este es https://github.com/uracing-team/mqtt_protocol, el ultimo *commit* que corresponde a esta instancia del trabajo tiene asociado un *tag* llamado *ESP8266_version*. En el archivo *README.md* se puede ver una explicación de las distintas carpetas y archivos del proyecto.

4. Validación

Para validar la solución desarrollada en este proyecto, se desarrolla una simulación que tiene como objetivo imitar el flujo de datos que serán recibidos por el microcontrolador. Con esta simulación, se analizará el alcance de esta solución y así verificar si cumple con los objetivos planteados inicialmente propuestos. Para esto se buscará obtener información sobre la cantidad de datos que pueden ser transmitidos, al igual que la frecuencia con que estos datos son enviados. Una parte importante de la validación es observar el tiempo transcurrido entre que los datos que son generados y visualizados, además, habrá que analizar si hay pérdida de información al terminar todo el proceso del proyecto. El desarrollo de la simulación reemplaza temporalmente el trabajo que hacen los sensores y el CAN Bus en la arquitectura de la solución, ésta parte aún no está desarrollada, entonces al crear las simulaciones, ayudaría a plantear las bases para que el microcontrolador pueda recibir estos datos. De esta manera, se puede decir que esta simulación también formaría parte de la solución al problema, ya que tiene el potencial de influir bastante en el desarrollo de los algoritmos que se utilizarán a futuro para el manejo de datos con el Teensy 4.1 Development Board, el microcontrolador que se utilizará cuando el proyecto esté terminado.

En el desarrollo de esta simulación, se utiliza el *dataset* mencionado en las pruebas anteriormente realizadas sobre los subsistemas de la solución en un archivo de tipo CSV (*Comma Separated Values*). Se utiliza el lenguaje de programación Python para desarrollar un código que se encarga de ingresar los datos desde el *dataset* periódicamente hacia el microcontrolador. Esta periodicidad, es la variable con la que se harán pruebas para evaluar la frecuencia con que los datos pueden ser recibidos y enviados por el microcontrolador. En la tabla del *dataset* hay aproximadamente 20 filas por cada segundo, es decir, el tiempo en que tarda un sensor en generar datos, es de aproximadamente 0.05 segundos. Este valor es la frecuencia ideal para un auto de carreras de alto rendimiento y sería el valor máximo de frecuencia que se usaría para este proyecto, ya que, con más frecuencia, una mejora en el análisis y la visualización de datos no vería un aumento a gran escala, pero sí habría un gran problema para el almacenamiento y procesamiento de tantos datos.

Los datos de fuente estarán almacenados en un archivo CSV en un computador, los cuales luego serán recuperados por el código en Python, para que después sean enviados a través de un cable *Micro USB to USB* hacia el microcontrolador Sparkfun ESP8266 Thing - Dev Board. Este microcontrolador tiene la ventaja de que es programado igual que un Arduino Board, los cuales tienen mucha documentación de su lenguaje basado en *C++*. Para enviar los datos desde el archivo CSV al microcontrolador, se utiliza un módulo de Python llamado pySerial [26], el cual encapsula el acceso a puertos seriales, de esta manera el puerto USB, el cual está conectado al microcontrolador, se utilizará para enviar los datos recuperados. En el microcontrolador también se recibe la información a través de un módulo serial de Arduino, en ambos lenguajes hay que especificar el límite máximo de cantidad de datos por segundo

que serán enviados, ya que hay cables que tienen distintas capacidades.

Una vez desarrollado el código de la simulación, se ejecuta el archivo de *docker-compose* en otro computador, conectado a la red WiFi emitida por el router de hogar. En esta instancia, como se puede ver en la figura 11, tanto el microcontrolador como el computador externo están conectados a la red de forma inalámbrica y totalmente separados entre ellos físicamente, es ahora cuando se puede empezar a probar la solución completa creada en este proyecto. Uno de los grandes obstáculos al empezar a probar esta solución, fue la sincronización de escrituras y lecturas del puerto serial entre el computador en el archivo CSV y el microcontrolador, es parecido a la programación de hilos en paralelo (*multi-threaded programming*) y contiene muchas de sus complicaciones, como los *dataraces* y *deadlocks*.

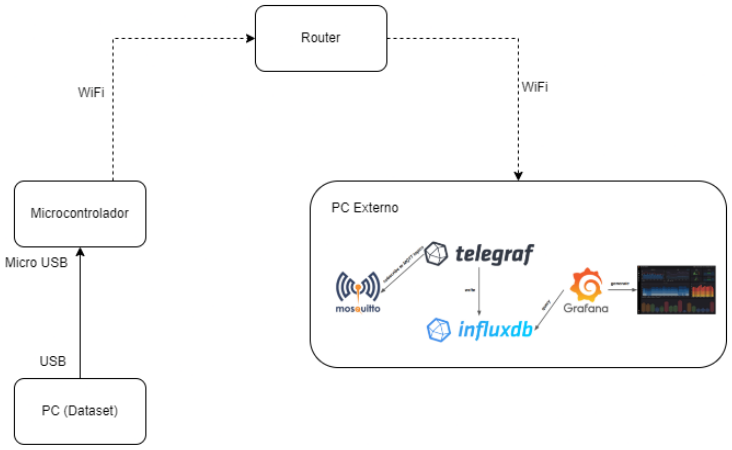

Figura 11: Arquitectura del sistema desarrollado para la simulación.

En la figura 12 se puede observar un pseudocódigo de lo que es ejecutado por el microcontrolador, el *loop* es la parte que se ejecuta reiteradamente hasta que el microcontrolador es apagado. Las partes principales del código es la recepción del mensaje tipo JSON por el puerto serial en el primer *if*, luego, en el segundo *if*, sigue la parte donde se *serializa* el mensaje, lo cual es necesario para que sea enviado al MQTT Broker. Finalmente, este mensaje es publicado al *broker* utilizando la librería PubSubClient, hay que mencionar que se debe especificar el tamaño máximo del mensaje, para que el microcontrolador libere espacio en su memoria para que este pueda ser almacenado.

```
void loop() {
    String payload;

    if (Serial.available())  // Serial port message available.
    {
        payload = Serial.readString();  // Reads message from Serial port.
    }

    if (payload.length() > 2)  // Message isn't empty.
    {
        char JSONmessageBuffer[1024];  // 1024 is the maximum message size in bytes.
        serializeJSON(payload, JSONmessageBuffer);  // payload is serialized into JSON
        Serial.println(payload);  // Message received is sent through the Serial port.
        mqtt_broker.publish("topic", payload);  // Serialized message is sent to MQTT
    }
    payload = "";
}
```

Figura 12: Pseudocódigo del algoritmo utilizado en el microcontrolador.

Como se mencionó anteriormente se medirá la frecuencia, como el tiempo en segundos entre envíos, la cantidad de información enviada, qué porcentaje de información fue perdida en el proceso y el tiempo, en segundos, que tardan los datos en llegar a su destino (latencia). Se inició la primera prueba enviando un solo dato cada 5 segundos, una prueba computacionalmente simple, es decir, sin mucho uso de memoria y con bastante tiempo para que el microcontrolador reenvíe el dato, sin que éste fuera reemplazado por el siguiente dato en la cola. Con el resultado de éste siendo positivo se verifica que el proceso funciona correctamente en su totalidad. Lo que sigue es ejecutar la misma prueba variando la frecuencia y la cantidad de datos por envío, esta cantidad tendrá una unidad de medida llamada *data points*, en donde un *data point* corresponde a un solo valor, por ejemplo, la velocidad en cierto instante fue de 95 kilómetros por hora, equivale a un solo *data point* con valor 95, esto está representado como un *JSON Array* con un solo valor, como se puede ver en la figura 7.

4.1. Resultados

En la tabla 2 se pueden ver los resultados obtenidos en diversas pruebas, la primera fila corresponde a la primera prueba, donde se envía un *data point* cada cinco segundos durante 20 segundos, aquí se recibieron todos los datos enviados y hubo una latencia promedio de aproximadamente 2,73 segundos. Las siguientes pruebas consistieron en aumentar la frecuencia con que se envían datos y al mismo tiempo la cantidad de información que estos mensajes transportan. Los resultados en general no son buenos, la cantidad máxima de datos que se han podido enviar desde el microcontrolador hacia el computador externo es de 16 *data points* por segundo, los cuales en un transcurso de 20 segundos, 320 *data points*, 5,6 por ciento de ellos nunca llegaron a su destino. No es posible hacer pruebas con una mayor frecuencia, ya que el

tiempo en que se demora el código en procesar y enviar los datos es de aproximadamente un segundo. Tampoco se puede probar con una cantidad de información mayor a 16 *data points*, pues los datos nunca son recibidos por el MQTT Broker en el computador externo, pues hay errores de suficiencia de memoria en el microcontrolador, estos resultados no logran tener la cantidad de información necesaria para poder transmitir los datos de todos los sensores.

Tabla 2: Resultados de la simulación.

Frecuencia (Segundos)	Cantidad (Data points)	Pérdida (Porcentaje)	Latencia (Segundos)
5	1	0	2,73
1	1	0	2,73
	2	0	2,22
	4	0	1,97
	8	0	1,87
	16	5.6	1,9

Para justificar estos resultados, hay que mencionar que se debe especificar el tamaño máximo que tendrán los mensajes a la librería PubSubClient, para que éste se preocupe de tener espacio disponible para que los mensajes puedan ser enviados. Al configurar este tamaño máximo a más de 2048 bytes, el microcontrolador comienza a tener un funcionamiento defectuoso, donde los mensajes no son enviados o su contenido no es el que corresponde. El tiempo que demora el microcontrolador en recibir, serializar y enviar los datos, es justo dentro de lo aceptable para este proyecto, un segundo es la frecuencia mínima con que los datos pueden ser actualizados, este lapso de tiempo es suficiente para que se puedan hacer análisis. La latencia obtenida en estos resultados no es la esperada, dos segundos es harto tiempo para un auto que se mueve a altas velocidades, visualizar información que corresponde a dos segundos en el pasado puede servir en algunos casos, pero no asegura que se pueda monitorear el auto de la mejor manera. La cantidad de *data points* no es un factor que afecta este rendimiento, debe ser un problema con la velocidad en que funciona el algoritmo.

A partir de estos resultados, se puede verificar que el envío de datos si es posible y el proceso en su totalidad si funciona para ciertas frecuencias y cantidades de información. Las visualizaciones sí son vistas y actualizadas en tiempo real, la información es almacenada por la base de datos, sin que ésta alcance su límite máximo. Se realizaron muchas pruebas en que se envió una parte del *dataset*, cuatro *data points* cada 0.05 segundos, durante 52 minutos. La base de datos nunca alcanzó su máxima capacidad de almacenamiento, lo cual es un buen resultado para este objetivo y en general para todos los objetivos que corresponden al desarrollo de software de la figura 3, eso sí, el hardware logra el mínimo requerido, lo cual es una base para poder avanzar en el futuro.

5. Conclusiones

El transcurso de este proyecto ha sido largo y ha habido varios obstáculos con los que se ha tenido que adaptar y superar usando los recursos y tiempo disponible, a partir de esto se ha desarrollado una solución que busca satisfacer los objetivos planteados en el inicio de todo el proceso. Para recordar, el objetivo principal corresponde a desarrollar un sistema que logre recuperar y enviar datos desde un auto de carreras hacia un dispositivo remoto que logre recibir, almacenar y visualizar esta información. Este proceso se puede separar en tres categorías, transmisión, almacenamiento y visualización, la cual fue la forma en que se establecieron las áreas centrales a desarrollar para cumplir con los objetivos.

Una vez desarrollada la solución, se implementó una simulación, con la cual se realizaron las pruebas que evaluarán la solución implementada. Los resultados obtenidos en las pruebas de transmisión de datos, como se explicaron en el capítulo anterior, no fueron los que se esperaron, para que los resultados fueran más aceptables, se espera que se acerquen un poco a los datos representados en el *dataset* utilizado para las pruebas que se realizaron sobre los subsistemas en el desarrollo de la solución. En la tabla 1 se observa que se deben enviar alrededor de 19 *data points* por cada envío, ya que se pueden generar 19 o más tipos de datos distintos cada instante. La frecuencia con que se envían los datos no necesariamente tiene que ser tan frecuente como los del *dataset*, ya que, en las visualizaciones en tiempo real, los datos no necesariamente tienen que ser actualizados con tanta frecuencia, una actualización de los *dashboards* menor a un segundo es razonable para poder monitorear y tomar decisiones.

Estos resultados fueron obtenidos a partir de un microcontrolador y router que no fueron planeados utilizar, ya que hubo varios obstáculos que impidieron que el hardware requerido para este proyecto fuera adquirido en este momento. Los resultados que se pueden obtener a partir de otro microcontrolador pueden ser muy distintos, pues tienen el potencial de utilizar más memoria y así generar mensajes con más información, también, se pueden enviar datos con más frecuencia, ya que el cable que conecta la fuente de datos con el microcontrolador puede tener características más favorables para la necesidad de este proyecto, también puede tener un procesador en que se pueda ejecutar el algoritmo con más agilidad.

Una de las grandes lecciones que trae este proyecto, es la forma en que personas en distintos equipos trabajan dentro de un mismo proyecto, la jerarquía y sincronización es una característica sumamente importante para que proyectos se desarrollen de la mejor forma posible. En este proyecto se dependía del trabajo de otras personas, tanto para facilitar el hardware, como para consultar sobre otros componentes, como el CAN Bus y sensores. Uno de los grandes obstáculos y desafíos dentro del equipo es la motivación, todos los integrantes del equipo son estudiantes y no se les ofrece ninguna recompensa monetaria por sus esfuerzos extracurriculares, encontrar la forma para mantener a los integrantes trabajando constantemente fue un desafío y se aprendió que hay que buscar otros medios para lograr esto.

5.1. Trabajo futuro

En todas las áreas se debe seguir trabajando, este proyecto será continuado en el futuro, ya sea por un integrante de URacing Team o por otro estudiante buscando un tema para realizar su memoria. Hay partes del auto con que brevemente se pueden empezar a visualizar datos, el motor ya está corriendo y actualmente se está desarrollando el sistema en que la ECU se conectara a este, esto implica que el CAN Bus puede estar disponible para que se puedan empezar a recuperar datos. Cuando el microcontrolador Teensy 4.1 Development Board y el Ubiquiti Bullet M2 HP estén disponibles, el sistema puede empezar tomar su forma más definitiva, el algoritmo que está escrito en Arduino puede ser optimizado o reestructurado para que se puedan alcanzar frecuencias más rápidas, posteriormente este algoritmo puede ser implementado en el lenguaje de se utilizará para el microcontrolador nuevo.

El almacenamiento de datos puede ser extendido para que tenga respaldos en distintos lugares, esto sirve para que los datos no solo sean visualizados en tiempo real, si no, para que también puedan ser procesados y analizados de manera *offline*, es decir, no instantáneamente. Esto permite aplicar otras tecnologías a los datos para que se puedan optimizar distintas partes del auto y así rendir mejor en las pruebas de Fórmula SAE. Una forma para respaldar datos es mediante una tarjeta SD dentro del auto, la cual puede estar conectada al microcontrolador y al terminar una sesión de entrenamiento o carrera, esta se pueda extraer manualmente para que luego sea ingresada en un computador externo. Otra forma para respaldar información sería que un auspiciador facilite un servidor remoto, por el cual se pueda ingresar toda la información recuperada por el sistema, esto es una vez que se haya establecido una conexión a internet.

En el futuro se espera desarrollar visualizaciones junto con los otros equipos de URacing Team, ya que, a partir de la experiencia de cada equipo, se pueden crear visualizaciones más específicas para ellos. Se eligió Grafana justamente por esta razón, pues es una herramienta fácil de usar que no requiere de mucha capacitación para que otro tipo de profesionales la puedan utilizar. Grafana también tiene la capacidad de que se le desarrollen *plugins*, de tal forma, que, si se quisiera crear una visualización más específica para el equipo, esta se puede crear e importar a la plataforma. Otra manera en que se podría solucionar un problema como el anterior, es creando una aplicación externa, la cual implicaría que se invierta mucho más tiempo y trabajo, pero lograría tener la flexibilidad de crear cualquier visualización, una librería como *D3.js* es ideal para crear visualizaciones en plataformas web.

6. Bibliografía

[1] Pagina web de Formula SAE. https://www.fsaeonline.com/

[2] Pagina web de URacing Team. https://uracingteam.cl/

[3] Pagina web de Michigan International Speedway. https://www.mispeedway.com/

[4] Why F1 chooses AWS. https://aws.amazon.com/f1/

[5] Oracle y Red Bull Racing lanzan un nuevo capítulo de innovación en la Fórmula 1. https://www.oracle.com/cl/news/announcement/oracle-and-red-bull-racing-launch-a-new-chapter-of-innovation-in-formula-one-2022-02-09/

[6] Significado de CAN Bus. https://helloauto.com/glosario/can-bus#:~:text=El%20CAN%20Bus%20es%20un,control%20electr%C3%B3nicas%20de%20un%20autom%C3%B3vil.

[7] Descripción de CAN Bus. https://www.ingenieriaymecanicaautomotriz.com/que-es-el-can-bus-y-como-funciona/

[8] How Formula 1 Car Sensors Create Data at Every Turn. https://blog.purestorage.com/perspectives/how-formula-1-car-sensors-create-data-at-every-turn/

[9] Que es la LAN y la WLAN en un router inalambrico. https://www.redeszone.net/tutoriales/redes-cable/lan-wlan-que-es-caracteristicas/

[10] Telemetry Transmission Going Full Speed on the Connectivity Track. https://www.peplink.com/telemetry-transmission-going-full-speed-on-the-connectivity-track/

[11] Telemetry system connects race cars with pit. https://www.eenewsautomotive.com/en/telemetry-system-connects-race-cars-with-pit/

[12] KX Named Official Supplier of Real-time Data Analytics to Alpine F1 Team. https://kx.com/news/kx-named-official-supplier-of-real-time-data-analytics-to-alpine-f1-team/

[13] MY20 Telemetry System. https://fsae.mit.edu/blog/2020/11/18/my20-telemetry-system

[14] Data-Driven Documents. https://d3js.org/

[15] Paul Dix, "The New InfluxDB Storage Engine: Time Structured Merge Tree", InfluxDB, 2015 https://www.influxdata.com/blog/new-storage-engine-time-structured-merge-tree/#:~:text=We're%20calling%20it%20a,the%20index%20files%20are%20compacted.

[16] F1 2020 Race Data. https://www.kaggle.com/datasets/coni57/f1-2020-race-data

[17] What is Docker? https://aws.amazon.com/docker/

[18] Overview of Docker Compose. https://docs.docker.com/compose/

[19] InfluxDB Docker Official Image. https://hub.docker.com/_/influxdb?tab=tags

[20] The official Grafana docker container. https://hub.docker.com/r/grafana/grafana

[21] MQTT: The Standard for IoT Messaging. https://mqtt.org/

[22] Eclipse-Mosquitto Docker Official Image. https://hub.docker.com/_/eclipse-mosquitto

[23] Pagina web de Telegraf. https://www.influxdata.com/time-series-platform/telegraf/

[24] Telegraf Docker Official Image. https://hub.docker.com/_/telegraf

[25] Teensy 4.1 Development Board. https://www.pjrc.com/store/teensy41.html

[26] Welcome to pySerial's documentation. https://pyserial.readthedocs.io/en/latest/

[27] Dashboard anything. Observe everything. https://grafana.com/grafana/

[28] Eclipse Mosquitto - An open source MQTT broker. https://mosquitto.org/

[29] SparkFun ESP8266 Thing - Dev Board. https://www.sparkfun.com/products/13711

[30] Tabla comparativa de modelos de ECU Haltech disponibles en Jolt Systems. https://joltsystems.net/products/haltech-elite-vms

[31] Página web de haltech https://www.haltech.com/product/ht-150902-elite-1500/

[32] Luz de freno universal de alta posición. https://es.aliexpress.com/item/4001209370898.html?spm=a2g0o.search0304.0.0.abf643cePoQ5Vs&algo_pvid=0f41abf8-3a6f-4535-bb51-f944f06ef218&algo_exp_id=0f41abf8-3a6f-4535-bb51-f944f06ef218-11

[33] Sensor de la temperatura del aire de entrada - Sensor IAT. https://codigosdtc.com/sensor-iat/

[34] Sensor MAP, fallas y funcionamiento. https://www.autoavance.co/blog-tecnico-automotriz/sensor-map-para-que-sirve/

[35] ¿Qué es un sensor CMP? https://mte-thomson.com/es/mte-responde/que-es-un-sensor-cmp/

[36] Sensor Posición Cigüeñal y Sensor Posición Árbol de Levas. https://nosso.com/esp/biblioteca_detalle/81

[37] Funcionamiento del interruptor de la luz de freno. https://helloauto.com/glosario/interruptor-luces-freno#:~:text=Su%20utilidad%20reside%20en%20que,se%20est%C3%A1n%20aplicando%20los%20frenos.

[38] Sensor de Temperatura del Aceite del Motor (EOT). https://www.autoavance.co/blog-tecnico-automotriz/160-sensor-de-temperatura-del-aceite-del-motor-eot/

[39] Funcionamiento sensor de nivel de aceite. http://www.hella.co/truck/assets/media_global/654_KI_GESTIoN_DEL_ACEITE_DEL_MOTOR_ES.pdf

[40] ¿Qué es el sensor de presión del aceite? https://www.motor.mapfre.es/consejos-practicos/consejos-de-mantenimiento/sensor-presion-aceite/

[41] Sensor de Velocidad fallas y Sensor Vss https://www.autoavance.co/blog-tecnico-automotriz/139-fallas-en-sensor-de-velocidad-sensores-de-velocidad-efecto-hall/

[42] Newhaven Display International. "NHD-7.0-800480FT-CSXN-T Datasheet"[Online] Disponible en https://www.mouser.com/datasheet/2/291/NHD-7.0-800480FT-CSXN-T-1219197.pdf

[43] Nils Braune, "Telemetry Unit for a Formula Student Race Car", Swiss Federal Institute of Technology Zurich, 2014, https://pub.tik.ee.ethz.ch/students/2013-HS/SA-2013-67.pdf

[44] Paul Dix, "Why Time Series Matters for Metrics, Real-Time Analytics and Sensor Data", 2021

[45] Francesca Lazzeri, "Machine Learning for Time Series Forecasting with Python", John Wiley & Sons, 2021

[46] Ted Dunning, Ellen Friedman, "Time Series Databases: New Ways to Store and Access Data", O'Reilly Media, 2014

[47] "Bullet M Zero-Variable Wireless Infrastructure Deployment.", Ubiquiti Networks https://dl.ui.com/datasheets/bulletm/bm_ds_web.pdf

[48] Matthew O. Ward, Georges Grinstein, Daniel Keim, Ïnteractive Data Visualization - Foundations, Techniques, and Applications", 2010

[49] Shorai, baterias. (Vehicle Type: Motorcycle, Make: Yamaha, Model: YZF-R6 ('01 - '16), Year: 2001) https://www.shoraidirect.com/en/products?action=vehicle

[50] Formula SAE, Rules 2022. Version 1.0, August 20, 2021.

[51] Cabrera, O. "Modificación de las curvas características de un motor para un prototipo de formula". Universidad Autónoma de México, Ciudad de México, 2018.

[52] Wilcker Neuwald Schinestzki, Daniel Gustavo Schreiner, Carlos Eduardo Guex Falcão , "Design of an Automatic Drag Reduction System Focusing on Cooling for Formula SAE Vehicle", Federal University of Santa Maria (UFSM), Tech. Rep. 2016-36-0511 E, 2016

[53] D. W. Radford, E. Scott and P. A. Fitzhorn , .ª Liftless Electronic 100ms Shift System for Motorcycle-Engined Racecars", Colorado State University , Tech. Rep. 2002-01-3322, 2002

Anexos

Anexo A

Motor

El equipo tiene el motor de la motocicleta deportiva, *Yamaha YZF-R6*, del año 2008. Este tiene cuatro cilindros de 600 centímetros cúbicos, seis velocidades distintas, entre otras especificaciones. El equipo de motor y tren motriz de URacing Team ha modificado el motor para que tenga una cámara de combustión con mayor volumen, para que así se pueda tener más potencia.

Anexo B

ECU

La ECU utilizada en el auto es la Haltech Elite 1500 ECU [31], contiene una carcasa resistente al agua sellada contra el medio ambiente, control de aceleración por cable, control de levas de admisión y escape, aprendizaje de mapas a corto y largo plazo, control de golpes, entre otras características. La ECU Elite 1500 proporciona a los calibradores de motor las herramientas que necesitan para obtener un trabajo de buena calidad, es una de las partes más valiosas dentro del auto.

Anexo C

Sensores

Dentro de la telemetría del auto son necesarios diversos sensores que aportarán información crucial en la fase de prueba y que posteriormente durante la competencia serán datos útiles tanto para el piloto como para el equipo que esté monitoreando el auto, así en caso de aparecer un problema o notar que el vehículo presenta un comportamiento fuera de lo esperado se podrán tomar acciones adecuadas.

URacing Team ya tiene en su posesión varios sensores, se presentan a continuación los sensores relacionados al área de telemetría.

Sensor de posición del cigüeñal

El sensor de posición del cigüeñal o también conocido por sus siglas en inglés CKP (Crankshaft Position Sensor), tiene por finalidad obtener la velocidad y la posición del cigüeñal para poder enviar esta información hacia la ECU. Su funcionamiento se basa en un sensor de tipo inductivo o de efecto hall; el que mide variaciones de corriente o de campo magnético que genera el movimiento del cigüeñal, y luego transforma esta

información en variaciones de voltaje que después la ECU interpreta para poder hacer una correcta gestión del motor y el combustible. Suele ubicarse en la caja de transmisión, en el bloque de cilindros del motor o también junto al volante de inercia. En la figura 13 se muestra una imagen del sensor de posición del cigüeñal de fábrica del motor. [36]

Figura 13: Sensor de posición del cigüeñal del motor Yamaha R6

Sensor de posición del árbol de levas

El sensor de posición del árbol de levas o por sus siglas en inglés CMP (Camshaft Position Sensor) tiene por objetivo tal cual su nombre lo indica, el medir la posición del árbol de levas ubicado en el motor, para enviar esta información hacia la ECU. Consta de una bobina y una sección imantada que obtiene la posición de dicha pieza del motor debido a la variación del campo magnético que se produce con la rotación de esta. Así, esta información viaja hacia la ECU como una señal eléctrica que posee la información de la posición del primer pistón ubicado al interior del motor. Esto permite sincronizar las piezas, que en términos simples, suben y bajan al interior del motor y así obtener una gestión correcta de los componentes para lograr un mejor desempeño. En la Figura 14 se muestra el sensor que se utilizará en este proyecto, el cual corresponde al sensor de fábrica del motor de la motocicleta Yamaha R6. [35]

Figura 14: Sensor CMP de Yamaha R6

Sensor IAT

El sensor de temperatura de aire o IAT por su significado en inglés (Intake Air Temperature) mide la temperatura del aire en la entrada del múltiple de admisión [51]. El principio de su funcionamiento se basa en un termistor, en el cual el valor de la resistencia es inversamente proporcional a la temperatura del aire. El sensor IAT envía la señal del termistor hacia la ECU y ésta regula la cantidad de combustible inyectado a la mezcla con el aire. En la Figura 15 se muestra un sensor compatible con motor de Yamaha. [33]

Figura 15: Sensor IAT

Sensor de pedal de freno

El sensor de pedal de freno, o también BPPS (Brake pedal position sensor) es un sensor que cumple la función de enviar una señal eléctrica hacia la ECU cuando se acciona el pedal de freno. Esto permite encender la luz de freno y en el caso de sistemas "Drive by wire"también accionar el mismo freno en las ruedas. Para el desarrollo del vehículo en cuestión, el sistema de frenos se realizará de manera mecánica, sin depender de este sistema controlado por computadora mencionado anteriormente. De esta manera, el BPPS solo cumplirá la función de enviar la señal para que se encienda la luz de freno. Para más detalles sobre cómo funciona este sistema, revisar la sección "luz de freno". A continuación, en la Figura 16 se muestra una foto del sensor BPPS de la motocicleta Yamaha R6. [37]

Figura 16: Sensor BPPS de Yamaha R6

Sensor MAP

Los sensores MAP (Manifold absolute pressure) permiten encontrar una relación entre la cantidad de aire y la presión que se logra registrar en el colector de admisión, con el objetivo de medir la carga de aire que va a los cilindros y así estimar la cantidad necesaria de combustible [51]. Luego de que se instala el sensor y es capaz de registrar la presión de vacío en la admisión, el sensor envía una señal eléctrica hacia la ECU, la cual va a determinar el volumen de aire que está entrando a la admisión del motor y posteriormente determinará la cantidad de combustible que deben aplicar los inyectores para que la mezcla de aire y combustible sea ideal [34]. El sensor MAP que se utilizará en el automóvil es de 3 bar. En la Figura 17 se muestra un esquema del principio de funcionamiento del sensor MAP.

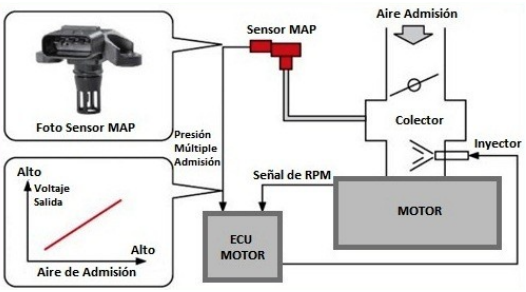

Figura 17: Principio de funcionamiento sensor MAP

Sensor de TPS

Sus siglas vienen del nombre del sensor en inglés, *Throttle Position Sensor*, el cual indica el porcentaje de apertura en la mariposa del cuerpo de aceleración, con lo cual se puede hacer una estimación del flujo de aire que va al motor. Su funcionamiento se basa en una resistencia variable (potenciómetro) que permite determinar la cantidad de corriente que fluye a través del circuito. En la Figura 18 se puede observar el TPS original que incorpora el cuerpo de aceleración del motor Yamaha. [51]

Figura 18: Sensor TPS OEM

Sensor de nivel de aceite

El sensor de nivel de aceite tiene la función de medir el nivel del lubricante en el cárter del motor, el que se ubica en la parte inferior de este. La importancia de medir el nivel de aceite radica en la necesidad de lubricante que tienen las piezas mecánicas al interior del motor, ya que estas no pueden moverse dócilmente sin la cantidad necesaria de aceite. Existen dos tipos de mediciones, la estática y la dinámica. La primera de estas se efectúa cuando el motor no está en marcha, mientras que la segunda es una medición continua que se realiza cuando el motor está en marcha y existe un flujo de aceite desde el cárter a las demás zonas del motor. [39] Para el desarrollo de este proyecto, se utilizará el sensor que viene incluido en el motor, el cual se muestra en la Figura 19.

Sensores de presión y temperatura de aceite

El sensor de presión de aceite ayuda a identificar de mejor manera cuando existen problemas de aceite. Su uso es prescindible, pero se recomienda incluir este sensor para

Figura 19: Sensor de nivel de aceite

que, junto con el sensor de nivel de aceite, se pueda entregar una medición más precisa. Los valores nominales de presión de aceite de un motor fluctúan entre los 30 a 45 PSI.

El sensor de temperatura de aceite, EOT (Engine Oil Temperature), tiene por objetivo medir la temperatura del aceite en el motor, por lo que dicho sensor es una rosca que posee un termistor de tipo negativo, el que envía una mayor resistencia si la temperatura es baja y viceversa. Los rangos típicos de medición van entre los -40 °C y los 170 °C, con un punto de operación óptimo alrededor de los 100°C. Cabe destacar que este sensor se puede ocupar además para saber cuando realizar cambios de aceite oportunos.

Para medir estas dos propiedades, se hará uso de un sensor combinado que mide temperatura y presión de fluidos a la misma vez (figura 20).

Figura 20: Sensor combinado de presión y temperatura de aceite

Sensor de temperatura de coolant

Este sensor, como lo indica su nombre, mide la temperatura del refrigerante del motor (engine coolant temperature). El funcionamiento se basa en una resistencia inversamente proporcional al de la temperatura del refrigerante. En la Figura 21 se observa la forma y ubicación del sensor de temperatura de coolant.

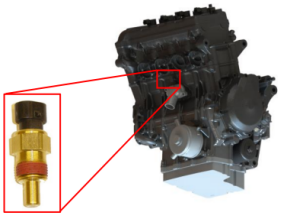

Figura 21: Ubicación del sensor de temperatura de refrigerante

Sensor de velocidad

El sensor de velocidad, también conocido por sus siglas en inglés como VSS (Vehicle speed sensor) cumple la función de medir la velocidad en base al movimiento de un determinado componente mecánico del motor. Este componente puede ser el eje del cigüeñal, o el eje de transmisión. Estos, poseen una rueda dentada, cuyos relieves son detectados por este sensor que es de tipo óptico, recibiendo información del momento cuando pasa cada dentadura del relieve. De esta manera, la ECU puede calcular la velocidad del vehículo a partir del movimiento de esta rueda dentada descrita anteriormente. En la Figura 22 se puede apreciar el sensor de velocidad por defecto que trae el motor. [41]

Figura 22: Sensor de velocidad

Sonda lambda

Uno de los elementos más importantes es la sonda lambda, la cual se encarga de medir la concentración de oxígeno para que luego la mezcla de aire y carburante sea la ideal al momento de la combustión. Existe una relación estequiométrica que determina la relación de aire para que la combustión se lleve a cabo de manera ideal, denominada relación lambda, de donde se obtiene que la relación aire/combustible es 14.7 para una combustión completa. Así, la sonda lambda logra registrar la cantidad de oxígeno que hay en el ambiente y la compara con la que hay después de la combustión, obteniendo un valor lambda que permite determinar si la mezcla tuvo o no la cantidad de combustible necesaria, como se indica en la siguiente tabla:

Tabla 3: Valores de lambda y combustible [51]

Lambda	Combustible en mezcla
1	Mezcla ideal en combustible
< 1	Mezcla rica en combustible
> 1	Mezcla pobre en combustible

En las siguientes Figuras 23 y 24 se puede apreciar el sensor que realiza estas mediciones junto con su controlador respectivo. Cabe destacar que estos sensores son de la misma marca de la ECU seleccionada (Haltech).

IMU

La unidad de medición inercial o IMU por las siglas en inglés (Inertial Measurement Unit) corresponde a un dispositivo capaz de medir y entregar datos acerca de la velocidad,

Figura 23: Sensor de concentración de oxigeno

Figura 24: Controlador de sensor de oxigeno

orientación y fuerzas gravitacionales presentes en un objeto mediante una combinación de acelerómetros y giroscopios. Estas unidades se usan comúnmente en vehículos aéreos, acuáticos, naves espaciales, misiles guiados entre otros.

Los datos recopilados por la unidad de medición inercial permiten aplicar un método de rastreo conocido como *navegación por estima*. Este método estima la ubicación del aparato detectando la tasa de aceleración mediante uno o más acelerómetros, los cambios rotacionales mediante giroscopios y los tiempos. Así, si la IMU informa que el aparato viajó por una hora en dirección sur a una velocidad de 100 Km/h promedio, el computador estima que se movió 100 Km hacia el sur luego de una hora desde su posición inicial.

Potenciómetro rotatorio

Un potenciómetro rotatorio es un resistor eléctrico cuyo valor resistivo es variable entre 0 y 1023 y se ajusta manualmente girando su eje.

Este sensor busca ser utilizado para medir el ángulo de giro del volante mediante el cambio resistivo que se ocasiona al virar, de forma que, al configurar el punto medio del recorrido del potenciómetro con el volante en posición central se logra abarcar completamente el ángulo de giro.

GPS

El sensor de posición de marcha o GPS (Gear Position Sensor) mide el ángulo de rotación del tambor de cambio ubicado en la transmisión del vehículo y lo transforma en

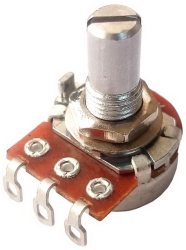

Figura 25: Potenciómetro rotatorio linear

voltaje para mostrar la marcha en el indicador del tablero.

Figura 26: Sensor de posicion de marcha

Galgas extensiométricas

Son sensores que miden la deformación, presión, carga, par y posición del material en el que esté emplazado. Funcionan en base al efecto piezorresistivo, propiedad que tienen algunos materiales de cambiar el valor nominal de su resistencia eléctrica cuando sufren deformaciones al estar sometidos a esfuerzos mecánicos, por lo tanto cualquier esfuerzo que deforme la galga cambia su resistencia eléctrica. Debido a esto es que se buscan utilizar galgas extensiométricas en diferentes puntos del chasis para medir las deformaciones sobre este y así monitorear el comportamiento durante las fases de prueba.

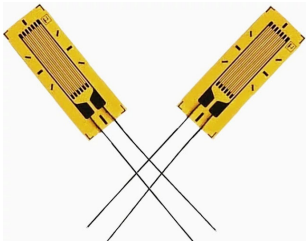

Figura 27: Galgas extensiométricas

Termocupla de escape

La termocupla o termopar es un sensor de temperatura simple cuya capacidad de medición se puede aplicar a un amplio rango de temperatura. Es un transductor formado por dos metales distintos que entrega una diferencia de potencial muy pequeña producto de la diferencia de temperatura entre uno de los extremos denominado punto caliente o

medida y el otro llamado punto frío o de referencia. A este efecto se le denomina efecto Seebeck.

La termocupla de escape mide la temperatura de los gases de escape, siendo de ayuda para mejorar la precisión y la eficiencia del motor. El sensor convierte la temperatura en voltaje y la envía a la ECU con la finalidad de controlar las condiciones del motor y que el vehículo trabaje de forma óptima.

Figura 28: Termocupla de escape tipo K