

Telemetría

Período 2016-2019

Nombre:	Maximiliano Vargas Vargas
Cargo:	Encargado de telemetría
Repositorio:	https://git.io/JeMOC
Fecha de ingreso:	17 de Mayo de 2016
Fecha de entrega:	17 de Diciembre de 2019

Resumen

En el presente informe se detallan los procesos de investigación, diseño, construcción, implementación, ejecución y proyección de los sistemas de telemetría de eolian fénix y eolian áuriga: la quinta y sexta versión de autos solares del equipo eolian. El informe se divide en dos grandes secciones, una para cada versión eolian. En cada sección se detallará en profundidad todo el proceso de desarrollo en el área de telemetría. Sin embargo, se hará mucho más énfasis en la última versión ya que consta de un trabajo más completo y elaborado, además de que implementa soluciones frente a los errores de la versión anterior.

Para eolian fénix se expondrán los componentes físicos utilizados, se describirá el principal objetivo del sistema desarrollado y posteriormente se detallará el trabajo realizado en temas de programación y disposición física de los elementos de telemetría. Luego se expondrá de desarrollos adicionales con respecto a la versión base del sistema, se expondrán dificultades y se analizará el sistema como un todo quedando expuestas sus debilidades y fortalezas.

Para eolian áuriga se comenzará por analizar las principales dificultades de la versión anterior y se plantearán objetivos generales a cumplir. Posterior a esto se detallará el vasto proceso de investigación y diseño haciendo énfasis en tres líneas de trabajo que servirán como síntesis para el desarrollo del sistema como tal. En cuanto al desarrollo, se expondrán varios diagramas en conjunto con explicaciones para su mejor entendimiento. Se expondrá el desarrollo adicional al sistema base planteado por los objetivos y se dejarán propuestos trabajos futuros.

Se realizarán conclusiones de todo el período de trabajo, comparando ambos sistemas de telemetría y rescatando lo mejor de cada uno. Se adjuntan diagramas esquemáticos de conexiones para cada sistema. Se expondrán las ventajas del sistema de eolian áuriga y su proyección para el futuro.

Se entiende que es el primer informe desarrollado por un estudiante de Ingeniería en Computación dentro de eolian, por lo cual se adjuntan vastas referencias a lo largo del informe para su mejor comprensión.

Finalmente, se deja fuera el extenso trabajo realizado por el autor en la página web eolian.cl durante estos cuatro años, ya que el documento se enfoca en el trabajo de telemetría.

Índice de Contenidos

1. Eolian Fénix	1
1.1. Contexto	1
1.2. Objetivos	1
1.2.1. Componentes de eolian	2
1.2.2. Componentes de telemetría	3
1.3. Investigación y Diseño	5
1.3.1. Protocolo CANBUS y lectura de componentes	5
1.3.2. Uso de Xbees	6
1.3.3. Interfaz visual	7
1.4. Desarrollo	8
1.4.1. Protocolo entre Xbees	8
1.4.2. Interfaz de visualización	9
1.4.3. Almacenamiento de datos	11
1.5. Desarrollo adicional	12
1.5.1. Velocímetro en tablet	12
1.5.2. LCD 16x2 de emergencia	14
1.5.3. Sensores análogos de temperatura	14
1.5.4. GPS	15
1.5.5. Conexiones	16
1.6. Trabajo futuro y conclusiones	17
2. Eolian Áuriga	18
2.1. Contexto	18
2.2. Objetivos	18
2.3. Investigación y diseño	19
2.3.1. Aprendizaje de versión anterior	19
2.3.2. Tecnologías de transmisión	20
2.3.3. Dispositivo electrónico principal	21
2.3.4. Utilización de dos pantallas	22
2.3.5. Integrando la Raspberry Pi con la red inalámbrica	23
2.3.6. Probando Xbees y su capacidad	24
2.3.7. Primera línea de trabajo	27
2.3.8. Uso mínimo de bits y <i>datatype</i>	28
2.3.9. Compresión de datos	29
2.3.10. Otra estrategia de compresión	31
2.3.11. Segunda línea de trabajo	31
2.3.12. Raspberry y el protocolo CANBUS	32
2.3.13. GPS y acelerómetro	34
2.3.14. Almacenamiento de datos	35
2.3.15. Aplicación web como interfaz visual	36
2.3.16. Tercera línea de trabajo	37
2.4. Desarrollo	40
2.4.1. Metodología de programación	40

2.4.2. Protocolo entre Xbees	40
2.4.3. Interfaz de visualización	45
2.5. Desarrollo adicional	46
2.5.1. Integración de Eolian Fénix a nueva telemetría	46
2.5.2. Navegación por mapa	46
2.5.3. LCD base	47
2.5.4. Conexiones	47
2.6. Trabajo futuro	48
2.6.1. Mejorar y terminar la interfaz	48
2.6.2. Integrar radio a Raspberry Pi e interfaz	48
2.6.3. Integración de modelo energético	48
2.6.4. Inteligencia artificial	49
2.6.5. Comandos para maniobrar a distancia	49
2.7. Conclusiones	50
Anexo A. Resultados pruebas de stress Xbees	51
Anexo B. Diagrama de clases y flujo de datos del protocolo	54
Anexo C. Esquemático de conexiones eolian áuriga	56
Anexo D. Esquemático de conexiones eolian fénix	57
Referencias	58

Listado de Figuras

1 Orion BMS de eolian fénix	2
2 Inversor Kelly de eolian fénix	2
3 MPPT Race de eolian fénix	3
4 Arduino UNO usado en telemetría de eolian fénix	3
5 CANBUS shield usado en telemetría de eolian fénix	4
6 Xbee usada en telemetría de eolian fénix	4
7 Software QT Creator 4.3.0 utilizado en eolian fénix	5
8 Arduino MEGA utilizado en eolian fénix	6
9 Xbee shield utilizado en eolian fénix	7
10 Captura de desarrollo de algoritmo de recepción en Qt	9
11 Captura de desarrollo de interfaz en Qt	10
12 Captura de uso de interfaz en Qt	11
13 Módulo SD para guardar datos en Arduino	11
14 Entorno de desarrollo Android Studio	12
15 Interfaz visual al interior de eolian fénix	13
16 Esquema conexión Arduino-LCD 16x2	14
17 GPS shield para Arduino usado en Eolian fénix	15
18 Esquemático de caja de telemetría e interior eolian fénix	16
19 Parámetros Zigbee, Bluetooth, and Wi-Fi	21

20	Raspberry Pi 3B+	22
21	Conexiones de pantalla 7 pulgadas en la Raspberry	22
22	Conexiones de pantalla 5 pulgadas en la Raspberry	23
23	Xbee Explorer	24
24	Capacidad de transmisión vs Tamaño de mensajes para distintos baudrates	26
25	Rendimiento de codificaciones universales	30
26	Conexión de componentes en red CANBUS	32
27	Raspberry Pi 3 B+ con Interfaz CANBUS dual	33
28	Interfaz CANBUS - Serial para Arduino	34
29	Acelerómetro ADXL345 para Raspberry Pi	34
30	Adafruit Ultimate GPS HAT para Raspberry Pi	35
31	Ejemplo de aplicación web	36
32	Esquema de conexiones del sistema completo	38
33	Flujo de datos del sistema completo	39
34	Valores de un componente	41
35	Ejemplo práctico del protocolo	41
36	Diagrama de clases y flujo de datos al enviar mensaje	43
37	Diagrama de clases y flujo de datos al recibir mensaje	44
38	<i>Primera iteración de interfaz gráfica eolian áuriga</i>	45
39	Esquemático de caja de telemetría e interior eolian	48
B.1	Diagrama de clases y flujo de datos al enviar mensaje	54
B.2	Diagrama de clases y flujo de datos al recibir mensaje	55
C.1	Esquemático de caja de telemetría e interior eolian áuriga	56
D.1	Esquemático de caja de telemetría e interior eolian fénix	57

Lista de Tablas

1	Configuración de las Xbee	25
2	Máxima transmisión para distintos baudrates	27

1. Eolian Fénix

1.1. Contexto

A Mayo de 2016, el equipo se encuentra en una etapa de restauración de eolian 4, renombrándolo a eolian fénix. Previo a la restauración, se pierde mucho en términos de conocimiento y trascendencia hacia los integrantes nuevos del equipo que en su momento fueron mayoría. En particular, el área de telemetría queda como hoja en blanco ya que se carece de programas, diagramas o cualquier forma explicativa de la telemetría anterior. Sólo se conocen algunos componentes usados previamente.

Bajo este alero, el autor del presente informe junto a Bryan Bizarro [1], ambos integrantes nuevos para Mayo de 2016, quedan a cargo de desarrollar un sistema de telemetría del vehículo con objetivos y herramientas bien concretas y dadas por el capitán del equipo eléctrico del entonces: Sebastián Díaz [2].

Para el año 2017, el proyecto busca realizar el desafío 'Santiago-Arica' que pretende recorrer 2.000 Km de distancia sólo con energía solar. Este desafío será clave para fijar fechas límites y de entregas.

1.2. Objetivos

A modo general, se pide lo básico de un sistema de telemetría: leer todos los componentes del vehículo bajo un protocolo dado, mostrar los datos dentro de eolian y mandarlos por una red inalámbrica hasta un computador externo donde también puedan verse los datos del vehículo. A modo concreto los objetivos planteados son los siguientes:

1. Se deberá obtener información de todos los componentes de eolian fénix: un BMS [3], dos Inversores Kelly KHB[5] y cuatro MPPT Race [6].
2. Para leer los componentes se usará uno o varios Arduinos UNO [7] con la ayuda de un shield para cada Arduino [8], que permitirá la lectura del protocolo CANBUS [9] de los componentes
3. Se deben usar Xbees [10] para la transmisión inalámbrica de los datos desde eolian hacia el computador fuera del auto. Se usará un shield de Xbee para poner la Xbee sobre un Arduino
4. Se usará Qt Creator [11] como software principal para el desarrollo de un programa que reciba la información enviada inalámbricamente y muestre los datos en forma visual
5. Se deberá implementar un protocolo de comunicación para enviar y leer los datos de eolian enviados desde los Arduinos al computador externo, por medio de las Xbees
6. Se deberá poder leer sensores análogos adicionales con los Arduinos
7. Se deben guardar los datos de telemetría de alguna forma

1.2.1. Componentes de eolian

Para detallar mejor los componentes de eolian fénix se expone una pequeña imagen y descripción de cada uno:



Figura 1: Orion BMS de eolian fénix

La Figura 1 muestra el BMS (Battery Monitoring System) el cual se encarga de medir los voltajes de todos los módulos del banco de baterías además de balancearlos y calcular el estado de carga del banco. Se le añade un módulo para medir temperaturas: Thermistor Expansion Module ???. Tanto el BMS como el módulo de temperaturas comunican por CANBUS a una velocidad configurable.



Figura 2: Inversor Kelly de eolian fénix

La Figura 2 muestra al inversor usado en eolian fénix. Convierte la corriente directa del banco de baterías en corriente alterna para los motores trifásicos. Éste arroja información de corriente por fases, temperatura de motores y de inversor entre otros. Se comunica por CANBUS a una velocidad configurable.



Figura 3: MPPT Race de eolian fénix

La Figura 3 muestra al MPPT (Max Power Point Tracker) usado para poder obtener la máxima potencia de los paneles solares, variando la carga resistiva de ellos. Arroja información de potencia y temperatura. Sólo se comunica a 125 Kbps en CANBUS.

1.2.2. Componentes de telemetría

También se adjunta una figura y descripción de los componentes de telemetría escogidos y de las herramientas a utilizar:

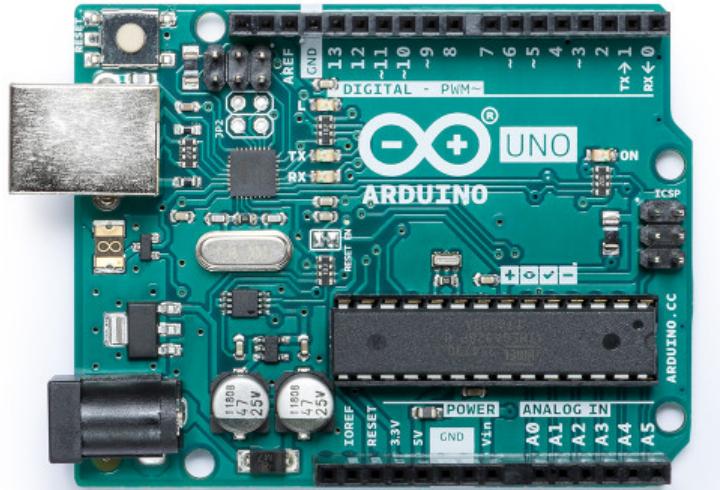


Figura 4: Arduino UNO usado en telemetría de eolian fénix

La Figura 4 exhibe un Arduino UNO que se usa en eolian para la capturación de datos de los componentes del vehículo.

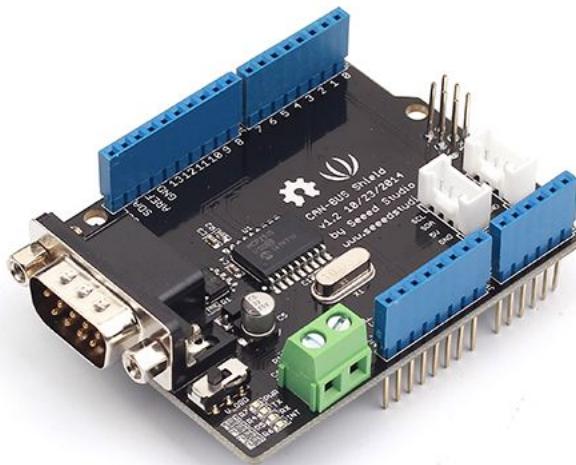


Figura 5: CANBUS shield usado en telemetría de eolian fénix

La Figura 5 muestra el shield CANBUS que se monta sobre el Arduino UNO y que mediante una librería permite la lectura de los componentes del eolian. Sólo soporta una sola red CANBUS.



Figura 6: Xbee usada en telemetría de eolian fénix

La Figura 6 muestra una de las dos Xbee necesarias para generar una red inalámbrica y comunicar hacia el sistema de monitoreo externo la información completa del vehículo.

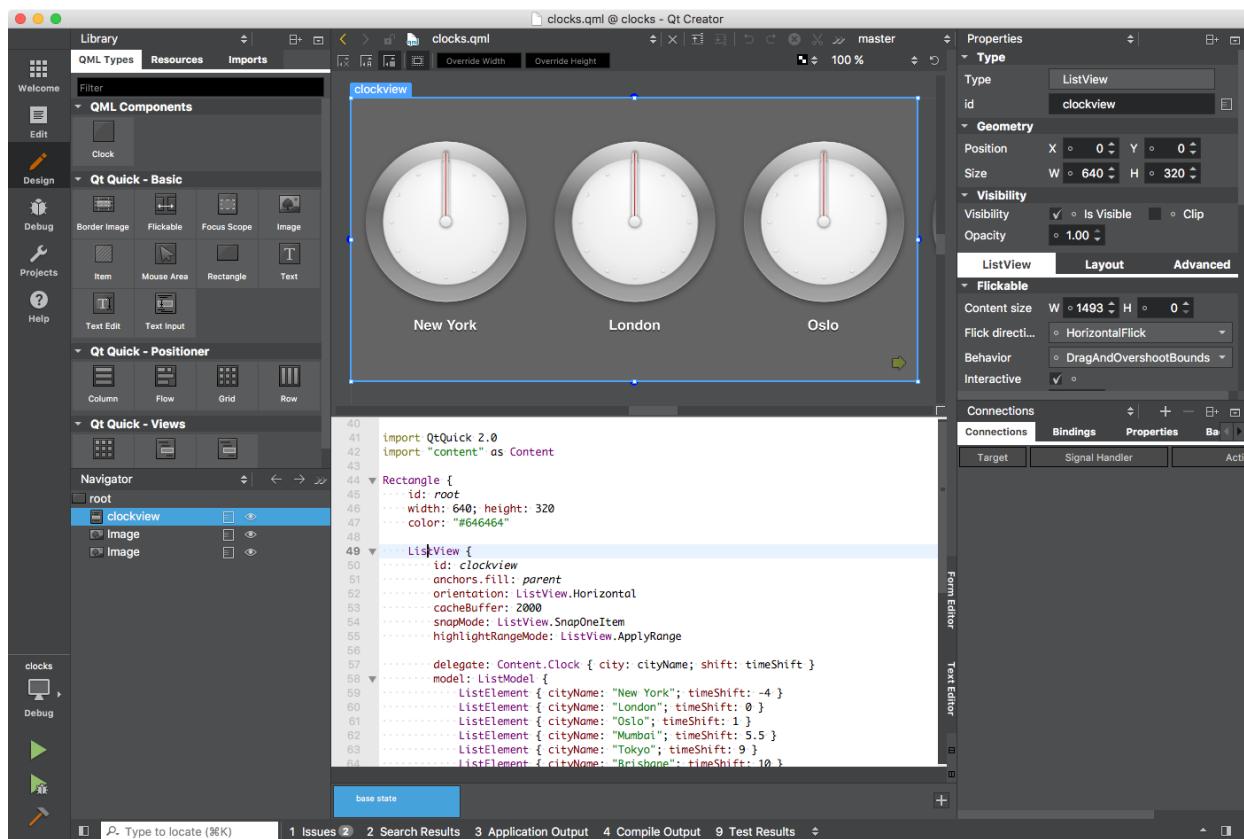


Figura 7: Software QT Creator 4.3.0 utilizado en eolian fénix

La Figura 7 muestra el software utilizado tanto para leer la información entregada por la Xbee como para mostrar dinámicamente la información de eolian en un computador externo.

1.3. Investigación y Diseño

1.3.1. Protocolo CANBUS y lectura de componentes

Gran parte de la investigación realizada trata sobre el uso de los componentes anteriormente señalados. Se aprende el funcionamiento del protocolo CANBUS, también a usar la librería del CANBUS shield para enviar y recibir mensajes desde la red CANBUS de eolian hacia el Arduino. Sólo con estas pruebas base se divisan dos problemas:

1. El BMS realiza un broadcast de voltajes. Pero al ser tantos datos desde el BMS, se anulan los mensajes de los demás componentes de la misma red obligando a tener una red exclusiva para el broadcast
2. Los MPPT se comunican fijamente a 125 Kbps mientras que el BMS obliga a una velocidad sobre 500 Kbps para hacer broadcast de los voltajes y temperaturas, lo cual obliga a tener dos redes CANBUS

Frente a los problemas encontrados se hace necesario usar al menos 3 redes CANBUS, teniendo dos Arduinos UNO y CANBUS shields adicionales. Una red tendría el broadcast de los voltajes, otra leería los MPPT a 125 Kbps y la última la información base del BMS y los inversores.

Además, la información debe ser centralizada en algún lado para poder enviarse, lo que obliga a comunicar a los Arduinos con un nuevo ente: un Arduino MEGA [62]. Este Arduino, mostrado en la Figura 8, tiene 3 puertos seriales adicionales los que permitirían comunicar a los Arduinos UNO a este Arduino mediante la librería SoftwareSerial [13].

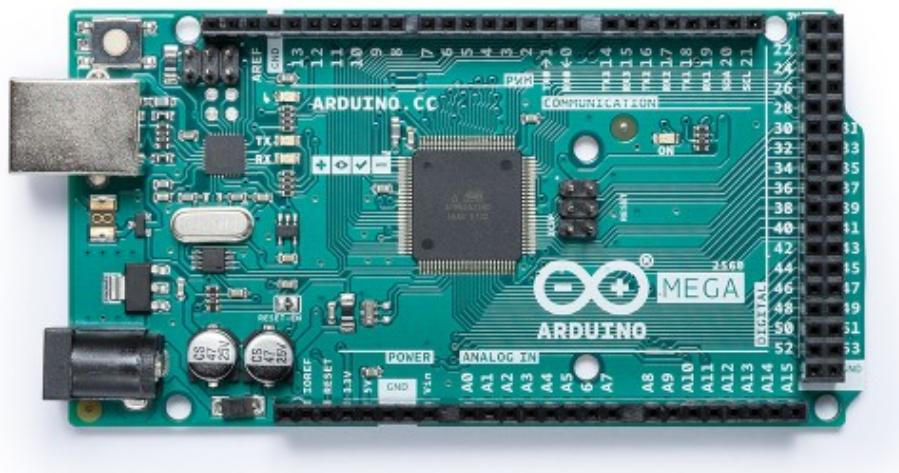


Figura 8: Arduino MEGA utilizado en eolian fénix

Dado que se tenían varios Arduinos a la manos, se probó la librería con éxito y se decide adoptar esta estrategia. Gracias a todo esto, se pudieron leer todos los componentes de eolian y centralizarlos en el Arduino MEGA. Posteriormente se procede a buscar una forma de enviar los datos por las Xbees.

1.3.2. Uso de Xbees

Se aprende cómo funcionan las Xbee, se configuran mediante el software XCTU [14], y se encuentra una forma de conectarlas al Arduino MEGA: mediante un XBee shield [15] como el de la Figura 9. Gracias al shield se podía utilizar el puerto serial del Arduino MEGA para enviar datos que serían reflejados en la red inalámbrica de las Xbee. De esta forma todos los datos puestos en el puerto serial del Arduino se reflejarían en el puerto serial del computador externo.

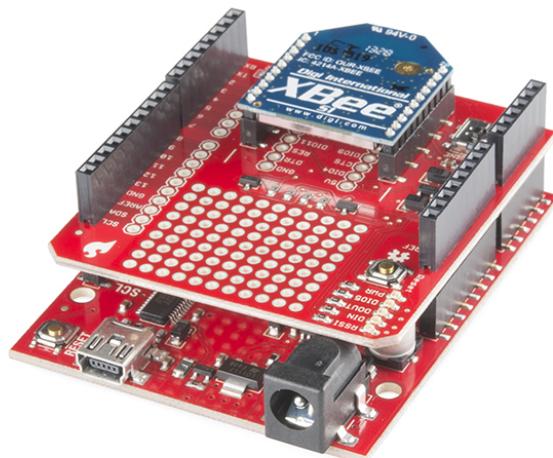


Figura 9: Xbee shield utilizado en eolian fénix

Para comenzar a desarrollar el programa de recepción de datos en C++ utilizando Qt Creator, se sigue el ejemplo de un [repositorio](#) del usuario vannevar-morgan [16] [17]. En él se usa una librería nativa de Qt para leer el puerto serial y gracias a ella se puede leer la Xbee conectada al computador. Como se tiene una base del programa funcionando, la interfaz visual y el resto del programa se desarrolla a partir de este repositorio. Cabe recalcar que el repositorio está bajo la licencia GPLv3.0 [18], por lo cual todo el trabajo realizado debe quedar bajo la misma licencia de distribución y uso.

Si bien, se hicieron pruebas para enviar datos básicos desde eolian al computador externo gracias a las Xbee y el repositorio base, aún no se tiene definido cómo enviar todos los datos desde eolian al computador en forma ordenada para que puedan codificarse y decodificarse tanto en el Arduino como en el programa en Qt. Esto será parte del desarrollo central de la telemetría.

1.3.3. Interfaz visual

Finalmente se busca cómo crear una interfaz visual a partir de Qt lo cual no fue tan difícil ya que Qt Creator es un programa donde se obtiene lo que se ve, término denominado WYSIWYG: What you see is what you get [19]. Se logró hacer funcionar visualizaciones básicas lo cual bastó para reafirmar el uso de Qt. La interfaz completa forma parte de la sección siguiente.

1.4. Desarrollo

1.4.1. Protocolo entre Xbees

Quizás la parte más difícil de la telemetría de eolian fénix fue crear un *sistema de mensajes* que pudieran mandarse desde los Arduinos, pasando por las Xbees, hacia el programa de Qt que pudiera leer los mensajes. En primer lugar se define la estructura básica de cualquier mensaje: un identificador y los valores.

El identificador o *header* constaría de 8 bits fijos definidos por el programador, esto permitiría hasta 256 mensajes diferentes. En segundo lugar, los valores serían enviados desde el Arduino usando la función `Serial.print()` [20] que imprime en el puerto serial los dígitos correspondientes al valor, **esto significa que si un valor ocupa 5 dígitos se enviarían 5 bytes**, lo cual sumamente ineficiente. Esta decisión se discutirá más adelante.

En tercer lugar se identifica la gran limitación del sistema de mensajes: la capacidad de la Xbee. Esta sólo podía enviar algo así como 80 bytes por mensaje, por lo que se quedaba limitado a este tamaño. Sin embargo, se logran enviar mensajes menores e iguales a este largo, logrando con éxito enviar valores desde Arduino hasta la aplicación en C++ de Qt en el computador remoto. Sin embargo **se tuvo que desarrollar un algoritmo de recepción** ya que a veces los mensajes no llegaban completos sino por partes, cortados, obligando a tener un *buffer* [21] que acumulara mensajes.

Para mejorar este algoritmo de recepción se incluye un byte final común en todos los mensajes: el byte 'E' de *end*. Esto ayudaba a recorrer el buffer en forma más eficiente pudiendo leer más rápido los mensajes recibidos. El algoritmo costaba a lo más $O(n/m)$ donde n era el largo del buffer y m el tamaño del mensaje [22]. En palabras simples, verificaba que tanto el byte header como ending calzaran con el trozo de buffer que se estaba analizando, si calzaba entonces se extraían los valores, si no se avanzaba en el buffer hasta llegar al final de éste. Esto implicaba que en cada recepción de un mensaje en Qt se hicieran $O(n/m)$ operaciones siendo una clara ineficiencia en el sistema desarrollado.

Una vez lograda la comunicación Arduino-Xbee-Qt, se tienen que programar a mano todos los mensajes definitivos a enviar desde el Arduino, para un largo fijo dado. Esto tomó bastante tiempo, pero logró el objetivo de reflejar los valores reales de los componentes eolian en Qt. A la vez, programar a mano los mensajes (decir qué valor va en qué mensaje de qué componente en qué posición) fue la ineficiencia más grande del protocolo, ya que, en un momento dado, días antes de hacer el desafío Santiago-Arica, el Arduino comenzó a enviar todos los mensajes en un largo diferente, haciendo inútil el sistema completo. A veces hasta habían bytes que simplemente no llegaban a Qt. La razón de esta inestabilidad se desconoce hasta el día de hoy.

Tan grave fue el problema, que se tuvieron que cambiar los mensajes a largos de 20 bytes, teniendo que reprogramar todo de nuevo (todos los mensajes hechos a mano), consumiendo mucho tiempo y terminando al límite. La Figura 10 muestra una captura del desarrollo del algoritmo:

```

dialog.cpp @ monitor_eolian [master] - Qt Creator
File Edit Build Debug Analyze Tools Window Help
monitor_eolian
  +-- Headers
    +-- dialog.h
    +-- qcgaugewidget.h
  +-- Sources
    +-- dialog.cpp
    +-- main.cpp
    +-- qcgaugewidget.cpp
    +-- qcustomplot.cpp
  +-- Forms
    +-- dialogui
  +-- Resources
    +-- resources.qrc
  Open Documents
    dialog.cpp
    dialog.h
    dialogui
    main.cpp
    qcgaugewidget.cpp
  monitor_eolian
  Debug
  Run
  Stop
  Type to locate (Ctrl+K)
  1 Issues 0 Warnings 2 Search Results 3 Application Output 4 Compile Output 5 QML Debugger Console 6 General Messages 8 Test Results

```

Figura 10: Captura de desarrollo de algoritmo de recepción en Qt

Como se aprecia en la Figura 10, programar los mensajes a mano requería de muchas condicionales `if()` y `else if()`. Se propone entonces que para un futuro sistema de telemetría se lean los mensajes de otra forma, ya que al hacerlo así se gana mucha rigidez.

Finalmente, se logra desarrollar lo solicitado en 3.000 líneas de código obteniendo el comportamiento necesario. Sin embargo queda en evidencia poca flexibilidad del programa, ya que para hacer un cambio se tuvo que reprogramar la mayoría de las cosas. Posteriormente se descubre la función `Serial.write()` [23] de Arduino que envía los bits necesarios y no un byte por dígito como `Serial.print()`. Esto habría ahorrado mensajes ya que los datos totales usarían menos bytes.

1.4.2. Interfaz de visualización

Qt mostró ser un gran aliado en temas de visualización ya que tenía las herramientas necesarias para implementar lo mínimo pedido. Por ejemplo incluía un 'LCD value' al cual se le podía pasar un valor desde la parte lógica que se actualizaba en la parte visual. Además se implementa un velocímetro para hacer más intuitivo el programa y se importa una librería para graficar en tiempo real: qcustomplot [24].

Como se ve en la Figura 11, se colocan dos gráficos. El primero muestra velocidad, consumo y generación de los paneles: se junta velocidad con potencia ya que el rango de velocidad 0-100 Km conjuga con el rango de potencia 0-10kW de consumo de eolian. El segundo muestra las temperaturas de los componentes eolian en el tiempo. Para ambos la ventana de tiempo es de 1 minuto.

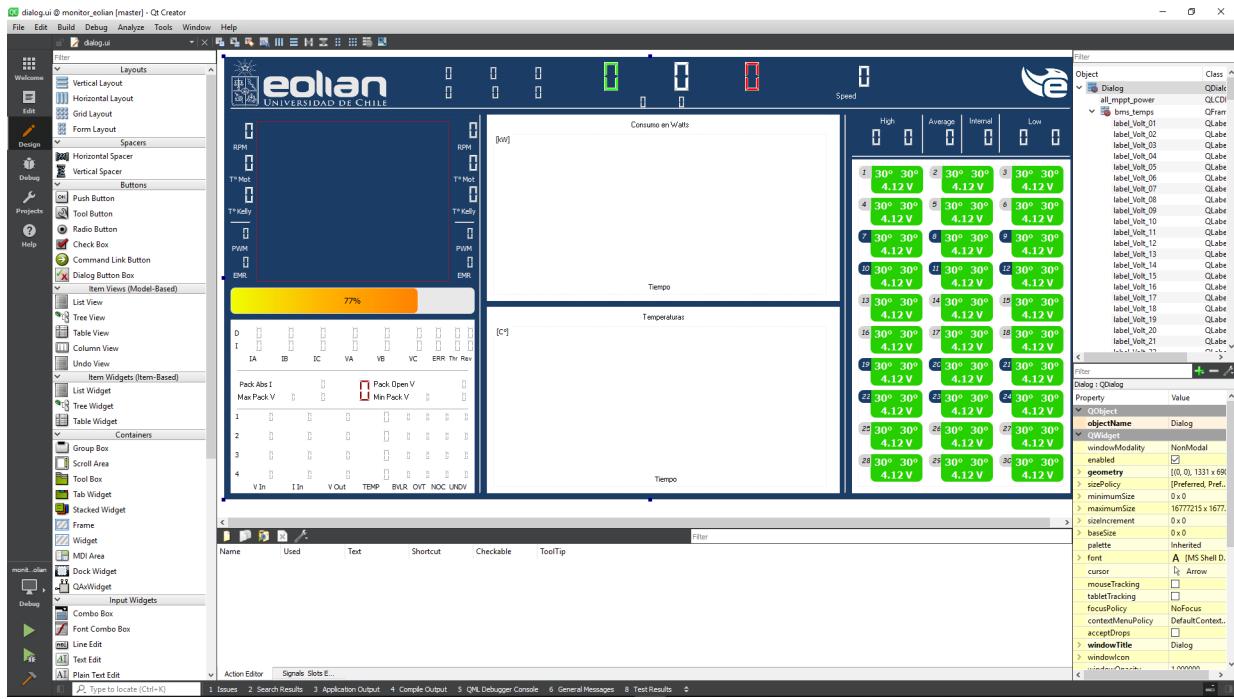


Figura 11: Captura de desarrollo de interfaz en Qt

Por otro lado, también es apreciable que en el lado derecho existan módulos con 3 valores. Estos módulos simbolizan los módulos del banco de baterías. El valor más grande corresponde al voltaje y los dos pequeños a las temperaturas del módulo. Además estos módulos cambian de color cuando el valor entra en rangos críticos como por ejemplo bajo/alto voltaje o baja/alta temperatura.

Al lado izquierdo se coloca toda la información de los inversores y los MPPT. Arriba se coloca en grande el consumo total, la generación y la resta de ambos. La barra horizontal a la izquierda representa el porcentaje de carga del banco.

En cuanto a diseño se dejan en mayor tamaño los datos de mayor relevancia, además la interacción de los colores permite identificar fácilmente valores fuera de rango. Se coloca el logo eolian y se usa de fondo el azul característico de eolian fénix. Se escoge un tema en general claro por temas de estética.

Un ejemplo de uso real de la interfaz se aprecia en la Figura 12. Esta figura es interesante ya que captura el momento de máxima generación de las celdas solares en el desafío Santiago Arica: 1070W.



Figura 12: Captura de uso de interfaz en Qt

1.4.3. Almacenamiento de datos

Se decidió optar por lo más simple: se guardarían en un archivo de texto todas las variables de estado de eolian cada vez que una de estas se actualizase. Esto se implementó en el programa de recepción en Qt usando una librería nativa sin problemas. Cabe destacar que cada fila guarda el tiempo de llegada de los datos.

Adicional a esto, se busca la forma de generar el mismo archivo de datos dentro de eolian. Se encuentra un componente, mostrado en la Figura 13, que permite al Arduino guardar datos en una tarjeta SD [25]. Se logra con éxito guardar los datos a medida que se leen los sensores bajo un simple tutorial [26]. Sin embargo, hubo problemas de velocidad con respecto a la telemetría: si se guardaban los datos en la SD, el envío a través de las Xbee no funcionaba bien y por lo tanto se decide no usar en la práctica. No se buscan más alternativas por falta de tiempo y recursos.

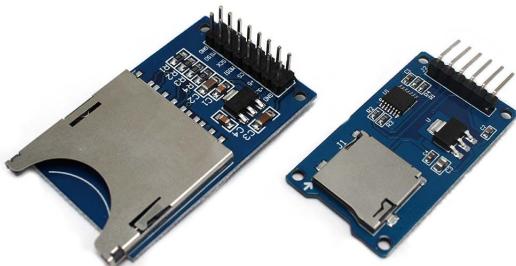


Figura 13: Módulo SD para guardar datos en Arduino

Es importante mencionar la importancia de los datos generados por eolian, ya que serían usados en el futuro para diversos motivos como por ejemplo validar el modelo energético. Sin embargo, los datos obtenidos hasta el momento no están correlacionados geográficamente ni temporalmente y por lo tanto faltaría obtener las coordenadas GPS de algún modo, desafío que se aborda en la sección de desarrollo adicional.

1.5. Desarrollo adicional

1.5.1. Velocímetro en tablet

Luego de tener un sistema de telemetría base, **se hace sumamente importante visualizar la velocidad y el consumo de eolian dentro del vehículo**. Para esto se solicita desarrollar un velocímetro claro y simple. Se utiliza una tablet android dentro de eolian, lo cual obliga al desarrollo de una aplicación en esta plataforma. Esta aplicación se crea con Android Studio [27], software y entorno de desarrollo en Java [28] especializado para aplicaciones. El entorno de trabajo del programa luce como la Figura 14 a continuación:

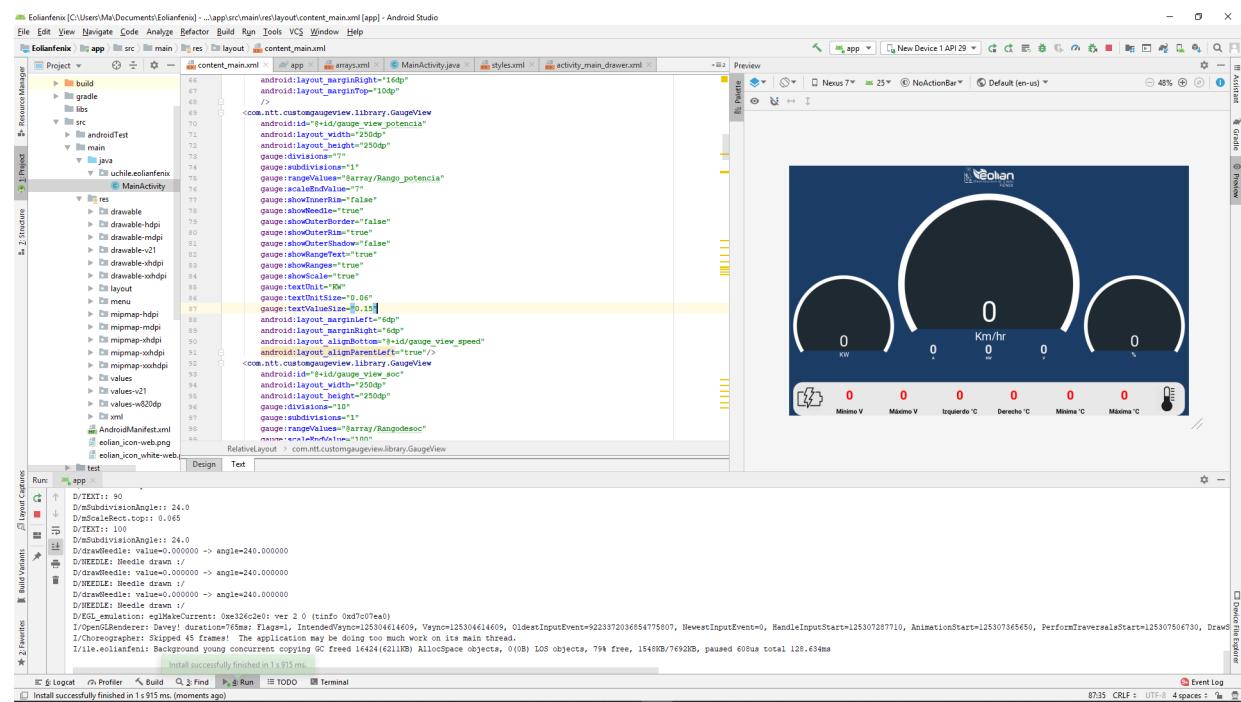


Figura 14: Entorno de desarrollo Android Studio

Luego de un tiempo aprendiendo desarrollo en Android, pasando por varios intentos, errores y pruebas, se logra obtener la visualización deseada con el uso de una librería para el velocímetro. La interfaz luce como en la Figura 15:

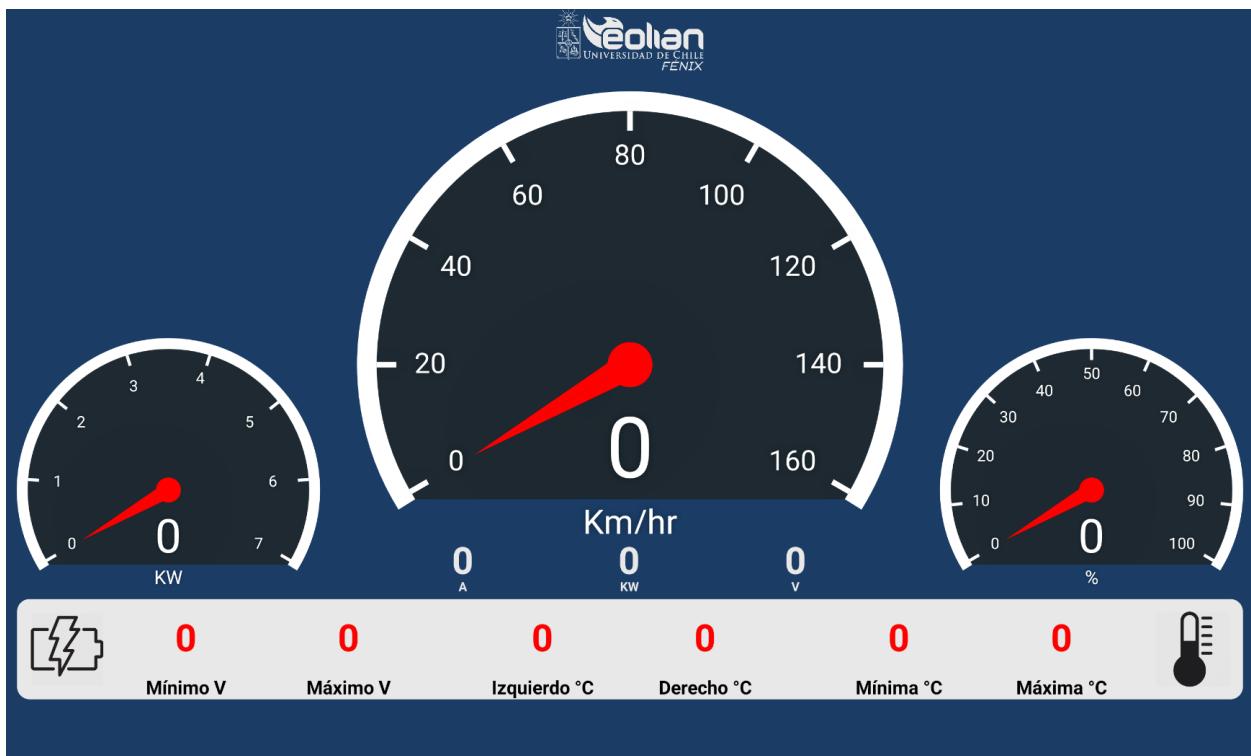


Figura 15: Interfaz visual al interior de eolian fénix

Los datos serían enviados desde el Arduino MEGA hacia la tablet por medio del micro-USB de la misma y un adaptador OTG [29]. Sin embargo esto evitaría que la tablet pudiera ser alimentada, por lo que se crea un adaptador que permita alimentarla y transmitir datos al mismo tiempo. La entrada micro-USB tenía 4 entradas: TX, RX, 5V y GND. Sin embargo en la práctica había que sumar una resistencia al circuito no siendo trivial el adaptador a desarrollar.

Finalmente se logra desarrollar lo pedido con la ayuda de otro integrante del equipo: Danilo Sánchez [30]. Sin embargo, no se hicieron las pruebas suficientes ya que en el desafío Santiago-Arica, luego del séptimo día, la entrada de alimentación de la tablet se quema no pudiendo continuar con la tablet y tomando el plan B preparado de antemano: el display mediante LCDs de 16x2.

1.5.2. LCD 16x2 de emergencia

Se pretende mostrar la información de velocidad y consumo son un LCD HD44780 de 16x2 celdas con el Arduino MEGA. Siguiendo el tutorial oficial de [Arduino](#) [31] se logra mostrar la información sin problemas aparentemente. El diagrama se conexiones se muestra en la Figura 16:

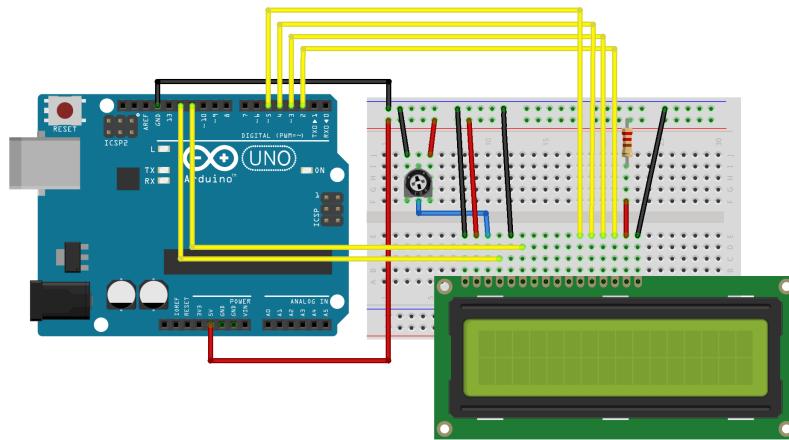


Figura 16: Esquema conexión Arduino-LCD 16x2

Es importante señalar que hubieron problemas de configuración ya que las tierras GND del LCD y Arduino no eran las mismas. También hubo problemas en el desafío Santiago-Arica, porque al parecer el Arduino MEGA no podía tener el envío de datos por las Xbee y el LCD al mismo tiempo, y se tuvo que sacrificar el LCD quedando eolian ciego de datos dentro del vehículo luego del fallo de la tablet. Situación bastante crítica que deberá ser evitada en el futuro. Esto pasó por no hacer pruebas previamente, y no se hicieron por falta de tiempo y recursos tanto físicos como humanos.

1.5.3. Sensores análogos de temperatura

También se solicitó leer dos entradas análogas de voltaje en el Arduino MEGA que correspondían a sensores análogos externos de temperatura que serían colocados en los motores. De esta forma se contrastaría la medición de temperaturas del inversor con la temperatura de estos sensores. Esto se tuvo que hacer ya que las mediciones del inversor no correspondían a las reales generando pánico en algunas ocasiones, ya que al medir externamente las temperaturas generalmente eran más bajas.

Esta tarea no fue difícil de implementar ya que es algo básico y típico en el entorno de los Arduinos [32]. Sin embargo los valores leídos tenían que ser escalados usando una función proveída por el datasheet [33] de los sensores.

1.5.4. GPS

Dado que se quiere relacionar los datos de eolian con la geolocalización temporal en el mapa, se busca un componente que funcione con Arduinos que guarde latitud y longitud en el tiempo. Dicho componente existe y se muestra en la Figura 17. Más detalles en sitio oficial. Bryan Bizarro se encarga de hacer el sistema funcionar con éxito en un Arduino UNO extra e independiente de todo el sistema de telemetría. Este Arduino se incluye en el desafío Santiago-Arica. Se sigue un tutorial [34] y se hacen pruebas básicas sobre el componente.

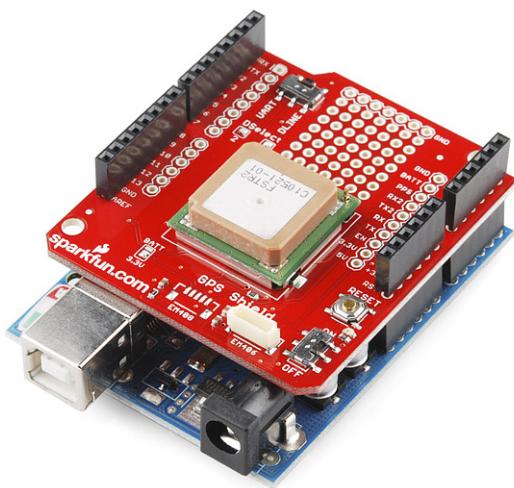


Figura 17: GPS shield para Arduino usado en Eolian fénix

Lamentablemente los datos guardados en la memoria del shield están corruptos al final del desafío no pudiendo lograr el objetivo principal. Nuevamente la falta tiempo y recursos es la principal causante. El componente se logra comprar a menos de una semana antes del desafío.

1.5.5. Conexiones

La disposición física de los elementos de telemetría se distribuye en dos cajas: La primera contiene al Arduino MEGA principal con el Xbee shield y CANBUS shield para leer BMS e inversores y un Arduino UNO encargado de leer el broadcast de voltajes con otro CANBUS shield. La segunda caja contiene otro Arduino UNO con un CANBUS shield que lee los MPPT a 125 Kbps.

De esta forma las cajas de telemetría lucen más o menos como la Figura 18. También se adjuntan en un tamaño más grande en Anexo D.1.

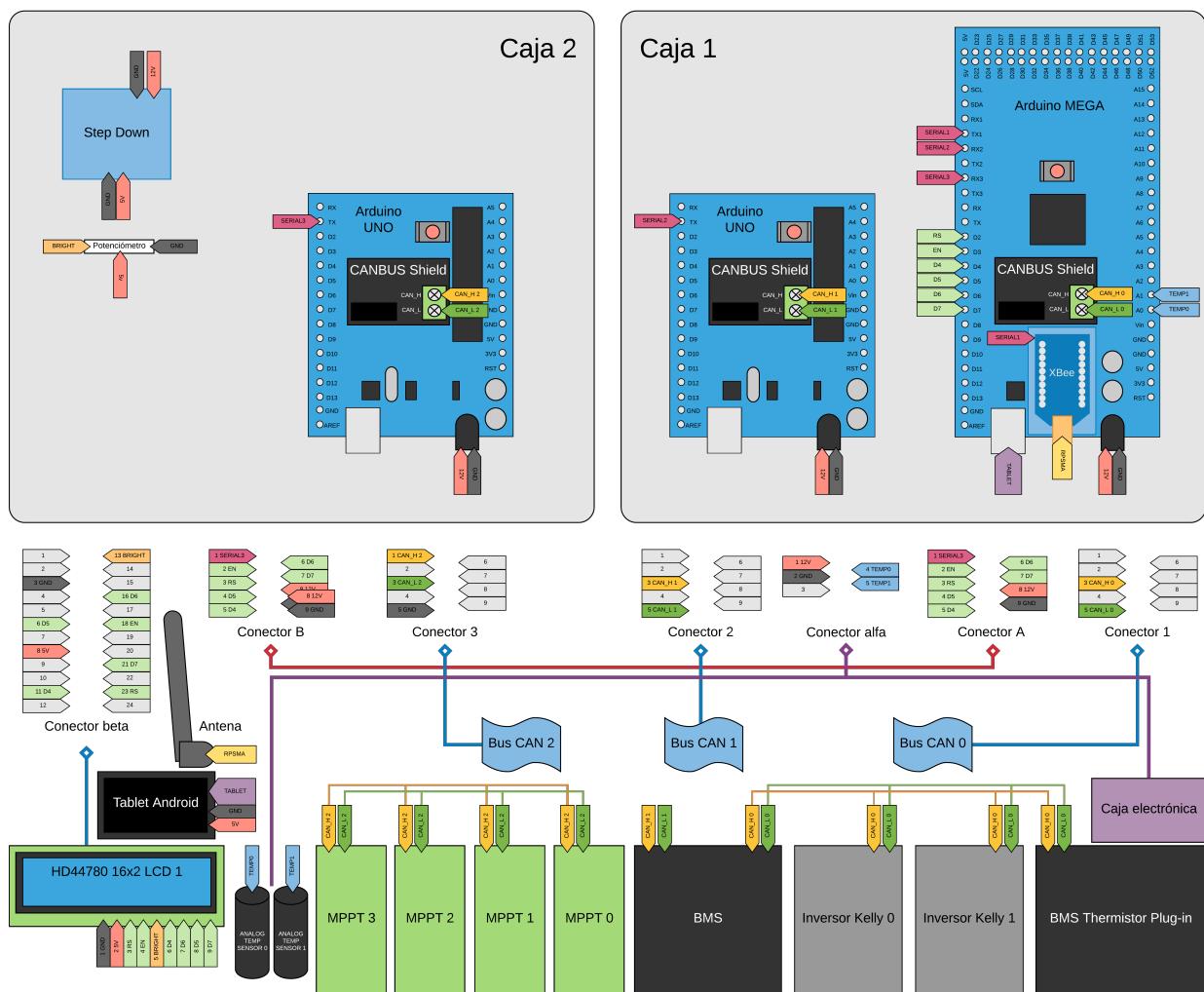


Figura 18: Esquemático de caja de telemetría e interior eolian fénix

1.6. Trabajo futuro y conclusiones

Sin duda alguna existen muchas mejoras que hacer en el sistema de telemetría: el protocolo de mensajes entre Xbees, la flexibilidad y escalabilidad de los programas desarrollados y también buscar otras soluciones para leer los componentes de eolian con menos componentes electrónicos para bajar el consumo. Se debe buscar una metodología de desarrollo de software que evite o limite los errores espontáneos como los que se tuvieron en el sistema desarrollado.

Sin embargo, se rescata que bajo las condiciones de limitadas de conocimiento, tiempo y recursos monetarios disponibles para el momento se haya podido dar con una solución funcional que de todas formas pudo llevar a eolian de Santiago hasta Arica.

Al asumir como miembro del equipo, el autor del informe no tenía conocimientos de programación ni de electrónica y todo se aprendió investigando y con la ayuda de otros integrantes del equipo presentes. Además haber comenzado de cero ya fue una gran desventaja al inicio. Se propone pues, buscar alguna forma de mejorar el traspaso de conocimientos de una generación a otra.

2. Eolian Auriga

2.1. Contexto

A Enero de 2018, tras cumplir el desafío Santiago-Arica con eolian fénix, el equipo propone el desarrollo de un nuevo vehículo solar, bajo los objetivos de poder participar y **ganar** en carreras solares nacionales e internacionales, trabajando a toda costa para salir dentro del TOP 3. Por otro lado, se quiere mejorar en gran medida el trabajo realizado con eolian fénix, ya que la generación del equipo que apadrinó a eolian fénix desde 2016 no participó en su diseño ni construcción.

Al mismo tiempo, se quiere mejorar en términos de difusión del proyecto y la ciencia en el país, mostrando que la tecnología actual permite el desarrollo de una electromovilidad solar, ayudando en gran medida a frenar el cambio climático y evadiendo emisiones de CO₂.

Dicho todo esto, comienza una nueva etapa de concepción y diseño del proyecto entero con mucho entusiasmo y energía. Posteriormente se incurre en etapas de diseño de detalles y desarrollo. Actualmente se está en esa última etapa y se espera construir el vehículo para Marzo de 2020.

Telemetría en particular permite el desarrollo sin la necesidad de tener todos los componentes físicos a la mano. Además como se verá posteriormente se usará parte de la misma tecnología de bajo consumo utilizada en eolian fénix, lo que permitirá el desarrollo de al menos un 80 % de todo el sistema sin haber comprado ningún componente. De esta forma el diseño y desarrollo de telemetría avanza rápidamente.

2.2. Objetivos

A diferencia de la versión anterior, donde los componentes y herramientas a utilizar fueron dados y fijos, ahora se da la libertad de establecer objetivos generales para luego diseñar y buscar componentes. Dicho esto los objetivos generales para un nuevo y mejorado sistema de telemetría son los siguientes:

1. Debe existir una transmisión de datos bidireccional entre el sistema al interior y exterior del vehículo
2. La transmisión de datos debe ser eficiente, enviando el mínimo de bits posibles
3. El ancho de banda, junto con la frecuencia de la transmisión deben permitir actualizar el estado completo del auto en menos de un segundo
4. El sistema al interior del auto debe ser de bajo consumo
5. El sistema debe ser fácil de utilizar, mediante documentación y manuales de uso para su utilidad en un futuro
6. El sistema debe ser escalable y flexible, es decir, debe permitir agregar y eliminar componentes en forma sencilla (Radio, parlantes, cámaras, sensores análogos como medidores de voltaje, corriente, temperatura, etc)

7. El sistema debe guardar los datos del vehículo en forma ordenada para su posterior análisis, como por ejemplo, calcular el consumo real del auto bajo diferentes condiciones, utilizar redes neuronales para establecer la mejor curva de aceleración del vehículo o utilizarlo en el modelo energético
8. El sistema debe comunicarse mediante protocolo CANBUS con los componentes del vehículo (BMS, Inversores, MPPT, etc)
9. El sistema debe permitir trabajar con al menos dos redes CANBUS a diferentes velocidades de transmisión (de 125Kb/s hasta 1Mb/s)
10. El sistema deberá considerar en su diseño la incorporación de un GPS y acelerómetro, para calcular altura e inclinación
11. El sistema debe permitir un alcance de 350m al menos
12. El sistema integrará al menos dos pantallas al interior del vehículo. La primera mostrará un velocímetro simple, y la otra la información completa del auto y será de al menos 7"
13. El código desarrollado deberá ser ordenado y documentado. Se utilizará Github[35] para su versionamiento y colaboración. Se utilizará español como idioma estándar de documentación, pudiendo al final del proceso traducirlo a inglés para su uso externo. Además contará con una licencia de uso a definir
14. Se deberán aplicar técnicas de programación eficiente, como lo es el uso de patrones de diseño, programación orientada a objetos (y clases) [36] o técnicas de administración de memoria
15. El sistema deberá en lo posible utilizar nuevas tecnologías como medida de innovación y eficiencia

2.3. Investigación y diseño

2.3.1. Aprendizaje de versión anterior

La principal **falencia del sistema anterior era la forma de enviar los datos y la irregularidad de recepción**, ya que se usaba un byte por dígito enviado, por ejemplo, si se enviaba un valor de 4 dígitos (como 4.012 para voltaje) se usaban 4 bytes, y no $\lceil \log_2(4012) \rceil = 12 \text{ bits} = 1,5 \text{ bytes}$ que eran los necesarios. Además al llegar cortados los mensajes (llegando irregularmente de 2-30 bytes por lectura) se tuvo que implementar un *buffer* de recepción para luego aplicar un algoritmo de costo $O(|\text{buffer}|/|\text{mensaje}|)$ por cada recepción para extraer los mensajes, lo cual es bastante costoso e innecesario.

Esto plantea dos desafíos: ¿Es posible comprimir los datos antes de ser enviados por la red? ¿Cuál es el mínimo de bits que se necesitan para codificar un dato? ¿Es posible añadir estabilidad al sistema de envío/recepción de datos?

Por otro lado, **el código desarrollado en eolian fénix era bastante rígido**: Cada mensaje debía estar resuelto por el programador, quien definía qué valores irían en cada mensaje. Además el mensaje era de largo fijo y en un momento el Arduino comenzó a enviar mensajes de distintos largos complicando más las cosas y provocando largas horas de trabajo.

El sistema a desarrollar debe ser flexible al punto tal que el programador sólo se preocupe de ingresar los parámetros de la red (como bit rate, largo de mensaje, etc) y las variables que quiere enviar. El resto lo hará el sistema quien replicará las variables de eolian en el sistema de monitoreo. Por lo tanto, otro desafío es crear un sistema flexible y simple de usar.

2.3.2. Tecnologías de transmisión

Existen muchas bandas y tipos de redes inalámbricas a parte de la Zigbee utilizada en eolian fénix. Wifi de 2.4 GHz sería la más conocida y comúnmente usada, sin embargo existen muchas más. Sería prudente abordar el problema de elección de una tecnología para eolian auriga desde el punto de vista del Internet de la Cosas o IoT. Eolian podría ser descrito como un auto inteligente, capaz de comunicarse con el exterior y desplegar muchas más funcionalidades que las de un auto normal. Por ejemplo, podría navegar por el mundo eficientemente mediante la optimización de rutas, la decisión de velocidad en cada punto y la conducción automática con la integración de un modelo matemático y GPS integrado. Podría conectarse a internet para navegar o desplegar información importante, entre muchas otras cosas más.

Se investigaron varias redes utilizadas por el IoT para la transmisión de datos resaltando: ZigBee, LorA y Wifi. [37] [38]. LorA se destaca por su alcance de 10Km y ultrabajo consumo de energía. Wifi por ser ampliamente usada con un alcance de 150m, y por último ZigBee por ser una tecnología de bajo consumo energético con un rango de 100m.

Se decide seguir adelante con las últimas dos, ya que ZigBee es conocida por estar en eolian fénix, y Wifi por su compatibilidad con diversos dispositivos. Por otro lado existe un paper [40] explicando en detalle la capacidad de transmisión y consumo tanto de Wifi como de Xbee. En él se aprecia que la capacidad Wifi es de hasta 11 Mbps a 400 mA de corriente, mientras que ZigBee tolera 250 Kbps a 25-35 mA tal como muestra la Imagen 19. Esto será clave para tomar la decisión final.

Wireless Parameter	Bluetooth	Wi-Fi	ZigBee
Frequency band	2.4 GHz	2.4 GHz	2.4 GHz
Physical/MAC layers	IEEE 802.15.1	IEEE 802.11b	IEEE 802.15.4
Range	9 m	75 to 90 m	Indoors - 30m Outdoors (LOS) up to 100m
Current Consumption	60 mA (Tx mode)	400 mA (Tx mode) 20 mA (Standby mode)	25-35 mA (Tx mode) 3 µA (Standby mode)
Raw data rate	1 Mbps	11 Mbps	250 kbps
Protocol stack size	250 KB	1 MB	32 KB
Typical network Join time	>3 sec	Variable, 1 sec typically	30 ms typically
Interference avoidance method	FHSS (Frequency-hopping spread spectrum)	DSSS (Direct-sequence Spread spectrum)	DSSS (Direct-sequence Spread spectrum)
Minimum Bandwidth required	15 MHz (Dynamic)	22 MHz (Static)	3 MHz (Static)
Maximum nodes	7	32 per access point	64K
Number of channels	19	13	16

Figura 19: Parámetros Zigbee, Bluetooth, and Wi-Fi

2.3.3. Dispositivo electrónico principal

Dentro del vehículo debe existir un elemento electrónico capaz de leer todos los sensores y componentes del auto, además de incorporar una pantalla touch de al menos 7 pulgadas de donde sea posible navegar e implementar diferentes funcionalidades. Por otro lado, debe utilizar el mínimo de energía en lo posible y ser escalable, es decir, se debe poder incorporar o quitar sensores a eolian sin afectar en gran medida lo desarrollado hasta el momento.

La tecnología Arduino incorporada en eolian fénix es bastante útil pero a la vez bastante limitada. En cuanto a memoria y poder de procesamiento es bastante precario. Y si los nuevos objetivos apuntan a un auto inteligente, un Arduino queda bastante corto. Es necesario una capacidad de procesamiento mayor, por ejemplo, para poder navegar por GPS mediante una pantalla interactiva. Además trabajar con Arduinos y una tablet como en eolian fénix tiene la obligación de desarrollar una aplicación en Android o IOs, lo cual es bastante poco conveniente para un equipo de estudiantes que no tiene conocimientos en ello.

Se necesita más que un Arduino y menos que un computador pequeño. Luego de investigar, se

concluye que lo más flexible es el uso de una Raspberry Pi [45], específicamente el modelo 3B+ [46] que permite el uso de dos pantallas [47] y además es compatible tanto el protocolo Wifi como Zigbee como se verá más adelante. De esta forma se pueden cumplir los objetivos planteados inicialmente.



Figura 20: Raspberry Pi 3B+

2.3.4. Utilización de dos pantallas

Dentro de la investigación y el desarrollo se da con las siguientes dos pantallas compatibles con la Raspberry Pi3+. La primera es *touch* de 7 pulgadas y ocuparía un puerto USB y el HDMI [48], tal como se muestra en la Figura 21:



Figura 21: Conexiones de pantalla 7 pulgadas en la Raspberry

Como se planteó en los objetivos, la más grande debe comportarse como una pantalla interactiva,

y dado que la pantalla encontrada es *touch* cumple con el requisito. Por otro lado la pantalla secundaria encontrada es de 5 pulgadas [49] y servirá para mostrar información esencial del eolian, como velocidad, carga y consumo, tal como se plantea en los objetivos.

La segunda ocuparía el puerto DSI de la Raspberry en conjunto con el micro-USB de alimentación [50]. En la Figura 22 se muestra una pantalla conectada, dejando libre el puerto HDMI para la otra pantalla:

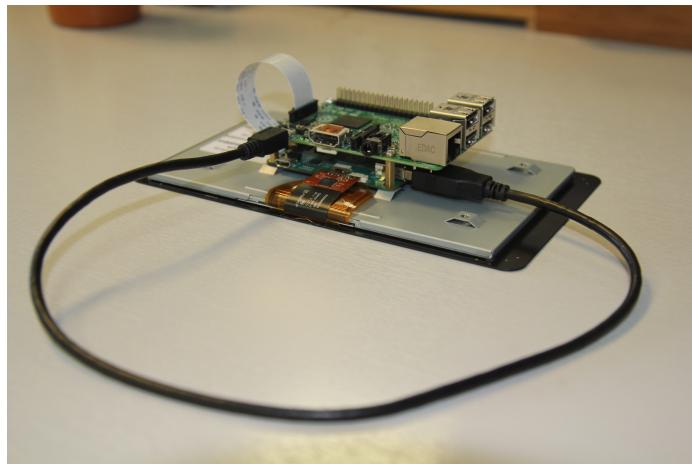


Figura 22: Conexiones de pantalla 5 pulgadas en la Raspberry

2.3.5. Integrando la Raspberry Pi con la red inalámbrica

La Raspberry Pi puede usarse como *wifi hotspot* [39], es decir, puede crear su propia red wifi donde otros dispositivos puedan conectarse. Esto hace aún más atractiva la Raspberry, ya que bastaría de una antena para amplificar el alcance de la señal wifi. Se plantea utilizar wifi como alternativa real, pero tiene una gran desventaja: el consumo de energía.

Tal como se describe en el paper mencionado anteriormente[40], el consumo del Wifi es hasta 16 veces mayor que el de Zigbee. Por otro lado, la capacidad de transmisión de 250 Kbps del protocolo ZigBee es más que suficiente considerando que eolian fénix tenía menos de 1 Kb de datos.

Como bien se sabe de eolian fénix, se necesitarían Xbees con Xbees explorer [10] como los de la Figura 6 y 23 para trabajar sobre ZigBee. Pero esto es bastante simple de integrar mediante el puerto USB de la Raspberry y no se necesitaría de ningún *shield* o aparato adicional para funcionar como el caso de eolian fénix.

Por el bajo consumo, la capacidad de transmisión suficiente y la simpleza de uso, **se decide por avanzar con las Xbee**.



Figura 23: Xbee Explorer

2.3.6. Probando Xbees y su capacidad

Se comprobará que la capacidad de las Xbee es de un máximo de 250 Kbps o 31 KBps. Para ello Digi, el fabricante de las Xbee, nos da soporte a diversas librerías [51] [52] de las cuales destacan las de Python y Java. Se escogerá la librería de Java, ya que es un lenguaje compilado y no interpretado como Python, lo cual lo hace más eficiente en tiempos de ejecución [53].

En primer lugar hay que descargar el software XCTU [14] de Digi para poder configurar las Xbee. Luego instalar los drivers en el sistema operativo a utilizar, en este caso Windows 10. Se conectan las Xbee mediante los Xbee Explorer y se abre el software para configurarlas.

En segundo lugar hay que configurar las Xbee. Es necesario saber que las redes de Xbees tienen un identificador de red que permite a las Xbees trabajar en una misma red (**PAN ID**). El protocolo ZigBee soporta además el uso de ACKs (Acknowledgement [41]) para evitar duplicados, y un *header* extendido en cada mensaje. En nuestro caso no nos interesa mucho evitar duplicados ni mucho menos un mensaje tan largo, por lo que ahorrar estos bits en cada mensaje puede ayudar un poco. Se realizan pruebas y se aprecia que evitar el uso de ACKS no mejora ni empeora el rendimiento, así como el uso o no de header extendido. Se escoge entonces la opción con ACKS y header (**MM**).

Además en la red debe haber como mínimo un *coordinador* y un *end point* (**CE**). Cada Xbee tiene un nombre identificador (**NI**) que permite comunicar una Xbee exclusivamente con otra. Es sumamente importante el valor de data rate **baudrate (BD)**, este configura la frecuencia de transmisión de bits y por lo tanto, la capacidad máxima de bps (bits por segundo) que puede soportar la red. Se harán pruebas para todos los valores de baudrate.

Por otro lado, existe lo que se llama CTS flow control (**D7**), que sirve para sincronizar la transmisión de las Xbee al estilo: “Hey ¿Puedo enviar un mensaje?; Sí, envíalo” [42]. Sin embargo esto no es estrictamente necesario para nuestro caso y se deshabilita. El parámetro Application Interface (**AP**) sirve para poder usar la Xbee mediante una aplicación externa, por lo que lo necesitamos activar para poder usar las Xbee en conjunto con la librería de Java. La configuración a usar en ambas Xbees es la default en todos los parámetros, salvo los mostrados en la Tabla 1.

Tabla 1: Configuración de las Xbee

Código	Parámetro	Valor
ID	PAN ID	1111
MM	MAC Mode	802.15.4 + MaxStream header w/ACKS [0]
CE	Coordinator Enable	{Coordinator [1], End Device [0]}
NI	Node Identifier	{EOLIAN, RECEIVER}
BD	Interface Data Rate	{1200,2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400}
D7	DIO7 Configuration	Disabled [0] (No CTS flow control)
AP	API Enable	API enabled [1] (API Mode for Java)

En tercer lugar se debe configurar la librería en Java para poder enviar mensajes desde y hacia las Xbee. Se utiliza el ambiente de desarrollo IntelliJ [43] para poder versionar en Github y debuggear en forma cómoda. Además el convenio de la universidad permite obtener este software en forma de licencia para estudiante. La serie de pasos para configurar IntelliJ con la librería de Java es la siguiente:

1. Descargar la librería oficial de Digi y extraerla ([link](#))
2. Crear un proyecto vacío en IntelliJ y crear el directorio /libs en el proyecto
3. Copiar **xbee-java-library-X.Y.Z.jar** en /libs
4. Copiar de /extra-libs a /libs los siguientes JARs: **rxtx-2.2.jar**, **slf4j-api-1.7.12.jar**, **slf4j-nop.1.7.12.jar**
5. Copiar /extra-libs/native a /libs
6. Ir a Project Structure/Modules/Dependencies, seleccionar el botón '+' y luego 'JAR or directories...' Agregar todos los JARs que se copiaron a lib, menos lo de /native
7. Seleccionar rxtx-2.2.jar dos veces para modificar la dependencia
8. Si se está en Windows se agrega la ruta /libs/native/Windows/Win64/rxtxSerial.dll. Si se está en linux se agrega la ruta /libs/native/Linux/x86_64-unknown-linux-gnu/librxtxSerial.so .
9. Además, si se está en Linux, el usuario actual debe concederse permisos para escribir y leer en los directorios **/dev/ttyUSB0** y **/var/lock** uniéndose a los grupos **dialout** y **lock** con tales privilegios. Para ello se usa **sudo usermod -a -G grupo usuario**, con grupo dialout y lock, y usuario el usuario de linux. Para verificar los grupos de un directorio se usa **getent group**. Ejecutar **sudo chgrp lock /var/lock** y **sudo chown -R root dialout /dev/ttyUSB0**. Reiniciar equipo
10. A partir de este momento pueden ejecutarse los ejemplos de envío que vienen en /example/- communication. Los más útiles son SendBroadcastDataSample, SendDataSample y ReceiveDataSample. Sólo hay que cambiar los puertos COM a los cuales están asociados las Xbee en el programa.

En cuarto lugar se desarrolla el programa. Lo primero es determinar el largo máximo de un mensaje que puede ser enviado mediante la API: 114 bytes (Mensajes de mayor tamaño no pueden ser enviados por error de ejecución).

Luego se hace lo siguiente: Para cada baudrate de la Xbee, se envían 1000 veces mensajes de largo entre 1 y 114. Luego se saca un promedio de mensajes por segundo para cada largo de mensaje. Finalmente, se calculan los bytes/s multiplicando la frecuencia de mensajes por el largo de mensajes. El código de este programa puede ser encontrado en el repositorio . Se obtienen curvas como se muestran en la Figura 24:

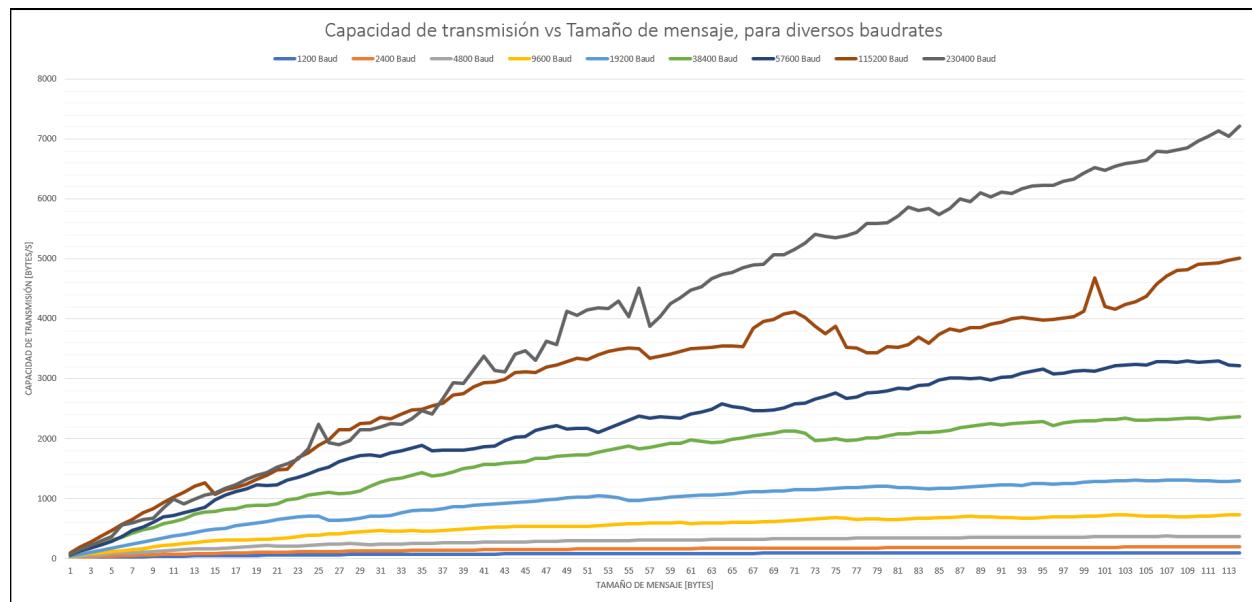


Figura 24: Capacidad de transmisión vs Tamaño de mensajes para distintos baudrates

Cabe mencionar que los resultados completos se encuentran en Anexo A. Se destaca que para cada baudrate se obtiene una transmisión máxima como se muestra en la Tabla 2:

Las xbee no subieron su temperatura considerablemente durante la prueba. Estuvieron a menos de 10 cm de distancia en posición cara-a-cara. Las pruebas se hicieron dentro de un cuarto de 3x3x2.5 metros. La temperatura promedio de las pruebas fue de unos 21°C.

De los resultados es posible observar que el límite superior de Bytes/s para cada baudrate es precisamente muy cercano al baudrate configurado. Sin embargo, **existe un límite por tamaño de mensaje de 114 bytes lo cual será clave para lo que viene en desarrollo**. La velocidad de transmisión puede aumentarse aumentando el baudrate, que aumenta los mensajes/s, sin embargo no se sabe aún si el baudrate afecta la distancia máxima de transmisión entre las Xbees. En eolian fénix era posible ver que la distancia de comunicación aumentaba al disminuir el baudrate, pero no se hicieron pruebas contundentes. Es por ello que si bien se tiene la capacidad para enviar hasta 6.000+ Bytes/s, la frecuencia de actualización podría ser menor.

Tabla 2: Máxima transmisión para distintos baudrates

Baudrate	Máx Capacidad [Bit/s]	Máx Capacidad [Byte/s]	Tamaño mensaje [Bytes]	Duración total del test [s]
1200	787.33	98.41	114	77400
2400	1554.64	194.33	114	39958
4800	2992.85	374.10	107	20986
9600	5870.65	733.83	114	11146
19200	10518.57	1314.82	109	6322
38400	18895.68	2361.96	114	3594
57600	26393.85	3299.23	109	2749
115200	40075.58	5009.44	114	1986
230400	57703.25	7212.90	114	1591

Por otro lado, eolian fénix tenía menos de 1.000 Bytes de tamaño en variables de estado, por lo cual esta prueba deja en evidencia que **es posible actualizar todas las variables del eolian en menos de un segundo** cumpliendo los objetivos planteados.

2.3.7. Primera línea de trabajo

Ahora que se saben las limitaciones de la red es posible seguir una línea de trabajo. Se decide desarrollar desde un baudrate 9600 como límite inferior dado que es el baudrate que se utilizó para eolian fénix y que demostró un buen comportamiento. Además, si el sistema funciona para tal baudrate, funcionará para baudrates más altos al estar sobredimensionado (con una transmisión lenta). Esto se traduce en un largo de mensajes entre 90-114 bytes y una capacidad de 800 Bytes/s aproximadamente. **Destacar que debe ser posible cambiar el largo del mensaje a gusto, pero nunca superar los 114 bytes.**

Aún más importante **se utilizará Java como lenguaje de programación del sistema**. Primero porque se comprobó el correcto funcionamiento de la librería de las Xbee. Segundo, porque es un lenguaje compilado que nos brinda mayor eficiencia. Tercero, Java es un lenguaje orientado a objetos, lo cual permite el desarrollo con *patrones de diseño y buenas metodologías de programación*, lo cual sigue la línea de eficiencia planteada y otorga mayor flexibilidad al programa. Finalmente, Java tiene una comunidad de desarrolladores enorme, lo cual facilita el desarrollo y los problemas se solucionan rápidamente. Además, millones de cosas están programadas en Java, lo que permitiría al programa principal extenderse como fuese necesario, por ejemplo, para agregar nuevas funcionalidades bastaría de importar librerías.

Dicho lo anterior y siguiendo la línea de eficiencia planteada en los objetivos, **se hace estrictamente necesario buscar una forma de comprimir los datos a enviar, o bien, enviar el mínimo de datos necesarios**.

2.3.8. Uso mínimo de bits y *datatype*

En primer lugar haremos notar que la mayoría de los valores a enviar tienen decimales. Por ejemplo 3.872 podría ser un voltaje de baterías. En lenguajes como C, C++ o Java hay dos alternativas, codificarlo como un datatype estilo *float* o *double*, o bien como datatype estilo *short* o *int* para luego dividir por 10^d , donde d es la cantidad de decimales y obtener el valor [44]

La primera estrategia involucra usar 32 bits por valor como mínimo fijo, ya que es el estándar IEEE para puntos flotantes implementado en la mayoría de los lenguajes de programación [54]. Una codificación de 16 bits que también sería útil, pero es no estándar en estos lenguajes deparando mayores complicaciones. El punto flotante utiliza 1 bit para signo, 8 bits para exponente y 23 bits para la *mantissa*. Esta alternativa permite el uso de números negativos y un rango de valores entre 2^{-24} y 2^{15} , bastante sobredimensionado para lo que se quiere. Otros estándares utilizan hasta 256 bits por valor y claramente no aplican para lo que se necesita.

La segunda estrategia depende del la cantidad total U de valores diferentes que el dato puede tomar. Por teoría de la información se necesitan $\lceil \log_2 |U| \rceil$ bits como mínimo para codificar todos los posibles valores del dato. Por ejemplo, si un valor va de 0 a 63 tenemos 64 valores posibles, por lo que se necesitan $\lceil \log_2 64 \rceil = 6$ bits para codificar ese rango. De esta forma se depende del rango de posibles valores de cada dato, evitando un largo sobredimensionado fijo y **ocupando el mínimo de bits necesarios por dato**

Por otro lado, como se depende de un rango, la codificación del valor puede ir acompañada de un *offset*, un valor a sumar o restar según se necesite, ya que no siempre queremos partir de 0. Por ejemplo, el rango de operación de voltaje en las baterías va de 2.650 a 4.200 [V], por lo tanto tenemos $4200 - 2650 + 1 = 1551$ valores diferentes. Podemos trasladar ese rango de 0 a 1550, utilizando $\lceil \log_2 1551 \rceil = 11$ bits, en comparación a utilizar un rango de 0.000 a 4.200 donde se utilizan $\lceil \log_2 4201 \rceil = 13$ bits. De esta forma, si el banco de baterías tiene 28 módulos utilizamos $11 \times 28 = 308$ bits = 38,5 bytes en vez de 45,5 bytes. Además si comparamos con la forma de eolian fénix de enviar datos, donde se enviaban 8 bits por dígito y en este caso cada valor tiene 4 dígitos, se utilizarían $(4 * 8) \times 28 = 896$ bits = 112 bytes, es decir, obtenemos una reducción del 65 % en cuanto a bytes utilizados.

En definitiva, **se escoge la segunda estrategia** porque se conoce el rango de valores de cada dato del eolian, y además va en línea de los límites de la entropía siendo minimal en el uso bits, cumpliendo los objetivos de eficiencia planteados.

2.3.9. Compresión de datos

La motivación de esta área de estudios es poder utilizar menos bits para representar la misma información de un elemento sin perder información: *lossless compression* [55]. ¿Es posible aplicar métodos de compresión a la estrategia de codificación ya analizada?

Las técnicas de compresión *lossless* comunes se basan en la **probabilidad de aparición de cada símbolo** en un conjunto dado. Aquellos símbolos que tienen mayor probabilidad de aparecer se codifican con menos bits, en forma tal que el total de bits utilizados es menor a la representación original. La codificación de Huffmann [56] es un ejemplo clásico de algoritmo que enfrenta este problema, logrando una codificación minimal. Sin embargo, esto funciona siempre que se conozcan de antemano las probabilidades de cada símbolo.

Por ejemplo, si supiéramos de antemano que en un instante dado los 28 módulos del banco de baterías estarán a 4.120 [V], podemos codificar ese valor como '0' y enviar 28 bits, en contraste a los 308 bits que enviaríamos con la estrategia previamente tomada. Adicionalmente enviaríamos que 0 está asociado a 4.120 utilizando al menos unos 8 o 16 bits más, pero aún así siendo mejores que el caso normal. El problema está en que no conocemos en ningún instante el valor siguiente de cada módulo, no conocemos la probabilidad de cada valor en cada instante.

Existe también lo que se llaman “codificaciones universales” [57] que son útiles cuando no se saben las probabilidades exactas de cada símbolo y **también cuando se quieren enviar valores en forma serial, concatenados uno a uno, sin saber cuantos bits usará el valor siguiente**. Son codificaciones que buscan competir con la codificación óptima (que conoce todas probabilidades) mediante un factor fijo. Dicho de otras palabras, sirven para codificar valores cuando se sabe que alguno son más frecuentes que otros.

Formalmente, este problema se puede abordar desde la perspectiva de *algoritmos online* [58], ya que estos algoritmos no conocen todo el *input* de antemano. Suelen comprarse con el *algoritmo óptimo* que conoce todo el input (como Huffmann) y como compite respecto de él. La codificación Elias γ es un ejemplo de codificación universal [59] para enteros positivos. Se utiliza cuando valores pequeños son más frecuentes que los mayores, sin saber la probabilidad exacta de todos los valores. Las codificaciones comparadas al valor a representar se muestran a continuación:

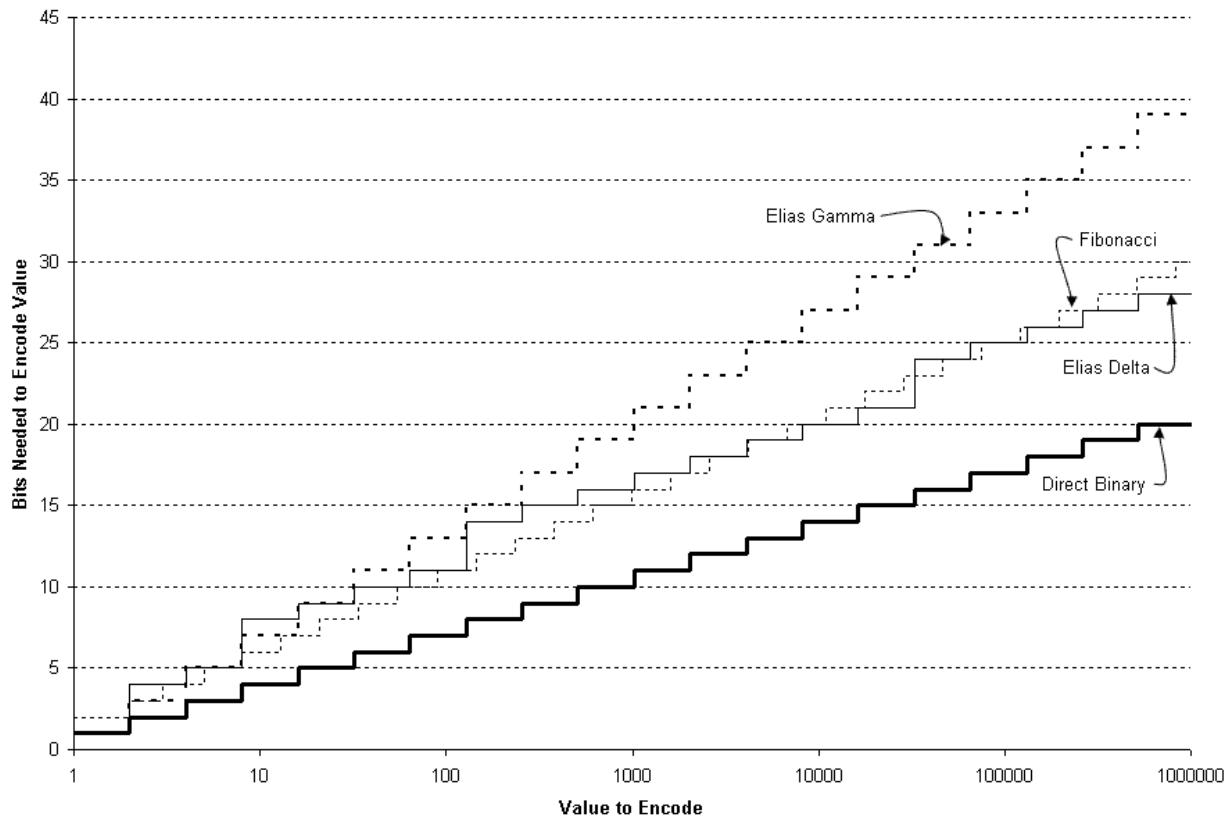


Figura 25: Rendimiento de codificaciones universales

¿Cómo aplica todo esto a lo que queremos? La telemetría del eolian implica un constante envío de diferentes valores en línea desde el vehículo hacia un centro de monitoreo. Estos valores varían en el tiempo sin conocer de antemano cuales serán los valores a enviar en el siguiente instante, por lo tanto no se conocen las probabilidades de aparición de todos los valores a priori no pudiendo usar algoritmos de compresión como Huffman. Sin embargo, se conoce que hay ciertos valores que son más probables que otros, por ejemplo, que las baterías del auto estén a 100°C es menos probable que estén a 25°C.

Además, sabemos que se pueden codificar los valores más frecuentes a códigos que usen menos bits, para así enviar en forma más frecuente mensajes de menor longitud y en forma menos frecuente mensajes de mayor longitud. Por lo tanto, el uso de codificaciones universales es tentativo para comprimir el envío de datos.

Sin embargo, no es trivial ver una distribución de probabilidad monótona creciente (como asumen las codificaciones universales) en ninguno de los datos de eolian, ya que, en algún momento todos los valores posibles de un dato podrían ser enviados por algún tiempo. Por ejemplo, consideremos los 1551 valores de voltaje que un módulo puede tomar. Si bien, la curva de descarga hace más frecuente los valores de voltaje entre 3.6 y 3.9 [V], lo que tentaría una codificación de menos bits para ese rango de valores, también es cierto que valores entre 2.6 y 3.6 [V] deberán ser enviados por mucho tiempo. Esto pondría al sistema de telemetría eficiente cuando el auto este cargado e ineficiente cuando el auto esté descargado.

Esto nos pone ante una situación compleja donde se deben estudiar las frecuencias de envío de cada dato del vehículo para poder aplicar una codificación universal sobre todas ellas para poder ahorrar bits de envío. Es por esto que se deja planteado el uso de codificaciones universales para el sistema de telemetría, y se opta por seguir con la estrategia señalada en la sub-sección anterior

2.3.10. Otra estrategia de compresión

Si bien no conocemos las frecuencias de envío de cada dato del eolian, si sabemos que todos los datos enviados desde eolian son variables de estado, es decir, son variables continuas en el tiempo. Si se conociera el máximo valor absoluto de la derivada en el tiempo de estas variables continuas, podríamos aplicar la siguiente estrategia: Enviamos una vez el valor de cada dato de eolian al sistema de monitoreo. Luego, para cada dato, sólo enviamos lo que varió de un instante al siguiente. Por ejemplo, si un voltaje estaba en 4.120 [V] y baja a 4.110 [V], podemos enviar solamente -10, lo cual usaría al menos 1 bit de signo y 4 bits para el valor, y no 11 bits.

Esta estrategia presupone conocer el máximo valor absoluto de la variación de cada dato del auto. Su debilidad está en la pérdida de mensajes: Si el sistema de monitoreo no recibe el primer mensaje que es fundamental, deberá solicitar a eolian que envíe todos los datos de nuevo. Por otro lado, para asegurarse de que el mensaje recibido es justamente el mensaje del instante siguiente y no se perdió ningún mensaje, se deberá incluir un contador a cada mensaje para verificar que el mensaje recibido es exactamente el siguiente. En caso de falla se deberá solicitar todos los datos a eolian.

Si p es la probabilidad que falle la recepción de un mensaje, l los bits necesarios para enviar todos los datos completos del auto, y r los bits necesarios para enviar las variaciones de todos los datos del auto, se usarían en promedio $b = p \times l + (1 - p) \times r$ bits. Siendo $b \ll l$ si la probabilidad p es lo suficientemente baja.

Esta estrategia predispone un estudio del máximo valor absoluto de la derivada de cada valor, además del un estudio de fallo de recepción de mensajes. Por otro lado, la implementación de esta estrategia es compleja, ya que utiliza comunicación bidireccional y queda corta ante escenarios no previstos, como el mal cálculo del máximo valor absoluto de la derivada de un valor. Por esto, se mantiene la decisión de codificación tomada al inicio.

2.3.11. Segunda línea de trabajo

Hasta este punto se ha definido que:

1. Se utilizará una Raspberry Pi 3B+ como dispositivo electrónico principal de eolian
2. Se usarán dos pantallas en la Raspberry, una de 7 pulgadas touch y una de 5 pulgadas
3. Se integrará a la Raspberry Pi el protocolo ZigBee mediante Xbees para transmitir datos de eolian al sistema de monitoreo
4. Se desarrollará un programa en Java que funcione tanto en eolian como en el sistema de monitoreo para el envío de datos

5. Se usará el mínimo de bits por dato en la transmisión, definido por la cantidad de valores posibles que el dato puede tomar
6. El sistema a desarrollar debe ser simple y encargarse de la forma en que envían los mensajes. Debe ser escalable y aplicar metodologías de diseño de software

Se ha planteado una forma bien detallada del envío de los datos, sin embargo **no se ha mencionado en que forma van a ser visualizados estos datos ni tampoco cómo se va a comunicar la Raspberry Pi con la red CANBUS**. Para ello se incurre en dos investigaciones adicionales

2.3.12. Raspberry y el protocolo CANBUS

Buscar la forma de enviar la información desde los componentes del eolian hasta la Raspberry Pi es una tarea súper importante. Como se mencionó anteriormente, todos estos componentes se comunican mediante el protocolo CANBUS, por lo tanto **basta con buscar alguna interfaz electrónica capaz de comunicar el bus de datos con la Raspberry Pi**

Antes de avanzar, es necesario explicar la conexión entre los componentes. CANBUS es precisamente un bus de datos y por lo tanto los componentes se conectan en paralelo en un bus de datos. CANBUS consta de sólo dos líneas: CAN High y CAN Low. Para evitar problemas de comunicación y efectos no deseados, se deben colocar dos resistencias de 220 [Ohm] a cada extremo del bus, es decir, en los componentes más periféricos [9]. El diagrama de conexiones estándar es el siguiente:

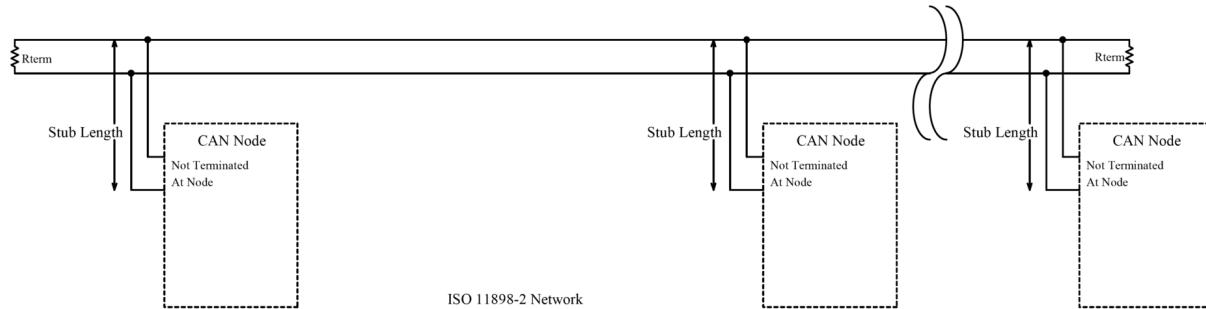


Figura 26: Conexión de componentes en red CANBUS

Como se ve en la Figura 26 la conexión es bastante simple. Por otro lado CANBUS soporta velocidades de transmisión de 125 Kbps, 250 Kbps, 500 Kbps y 1 Mbps, sin embargo puede que no todos los componentes de eolian soporten toda la gama de velocidades. Para eolian fénix, los MMPT Race sólo se comunicaban a 125 Kbps por configuración de compra, mientras que los inversores y el BMS soportaban toda la gama de velocidades. Esto se tradujo en usar más Arduinos e interfaces para poder leer todos los componentes: algo bastante engorroso.

No se puede asegurar que para eolian Áuriga todos los componentes soporten todas las velocidades, por lo tanto, que la interfaz soporte 2 velocidades parece razonable. Por otro lado, se diseñará un plan para tener hasta tres o más buses CAN.

Luego de búsqueda e investigación se logra dar con una interfaz diseñada para una Raspberry Pi, proveída por Copperhill Technologies [60] y que soporta dos velocidades:

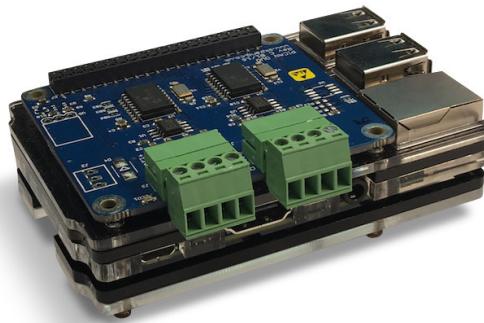


Figura 27: Raspberry Pi 3 B+ con Interfaz CANBUS dual

Esta interfaz posee una librería en C y Python para poder enviar y recibir mensajes por las redes CANBUS que estén conectadas. Esto complicaría un poco las cosas en temas de software, ya que el programa principal estaría en un entorno Java, por lo cual sería necesario comunicar dos procesos mediante un *pipeline* [72] u otra forma de comunicación y serialización. Existen librerías que se encargan justamente de esto [73]. Dentro de las ventajas, simplificaría enormemente las conexiones dentro de eolian ya que las redes CANBUS se conectarían directamente a la interfaz y nada más. La interfaz se comunica mediante los pines UART de la Raspberry por lo cual estarían restringidos para otros componentes. La principal limitación de esta interfaz es que no es escalable a una tercera red CANBUS y su alto costo de 72 USD a Diciembre de 2019.

Para poder soportar una tercera red se crea una solución de bajo costo pero bastante engorrosa por el uso de bastantes componentes. Se encuentran diversas interfaces CAN-to-Serial [?] las cuales podrían conectarse mediante el puerto serial a un Arduino y con una librería leer el bus de datos. El modelo Arduino MEGA [?] tiene 3 puertos seriales adicionales, por lo que bastaría de este Arduino para leer hasta 4 velocidades diferentes. Luego debería comunicarse con la Raspberry Pi mediante I2C [65] u otro protocolo digital común entre ambos.

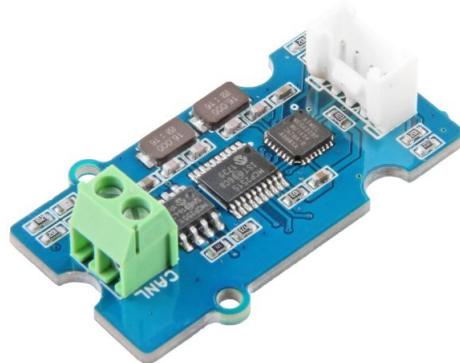


Figura 28: Interfaz CANBUS - Serial para Arduino

Sin embargo, esta solución incurriría en consumos adicionales del Arduino, o los Arduinos (en el peor caso), y además para hacer cambios al programa habría que programar en C los Arduinos y subir el programa con cada cambio. A pesar de todo, no es una alternativa tan mala tampoco ya que en caso de no tener recursos se puede adoptar.

Finalmente, se tienen dos propuestas para la lectura de componentes, y ambas satisfacen los objetivos planteados. En temas de software habrá que trabajar con las librerías disponibles sin embargo no sería un gran problema. Con esto queda solucionada la comunicación Raspberry-Componentes, pero no se podrá avanzar en esta área hasta tener todos los dispositivos en mano.

2.3.13. GPS y acelerómetro

Como se plantea en los objetivos, se necesita de un GPS y un acelerómetro que puedan ser leídos por el sistema de telemetría: a estas alturas por la Raspberry Pi. Se encuentra un GPS shield para la Raspberry Pi [63] que cumple justamente con la función solicitada, y un acelerómetro modelo ADXL345 [64] que funciona mediante I2C. Ambos se muestran en las Figuras 29 y 30 a continuación:

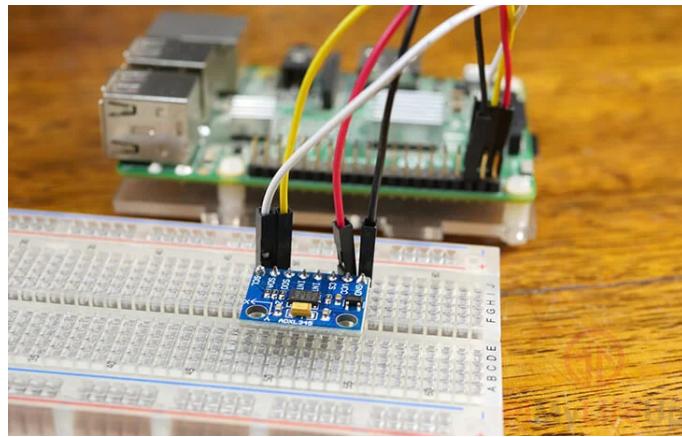


Figura 29: Acelerómetro ADXL345 para Raspberry Pi



Figura 30: Adafruit Ultimate GPS HAT para Raspberry Pi

Se examinan los esquemas de conexiones y las librerías necesarias para hacerlos funcionar y se concluye que no son para nada complicados de utilizar. Por lo tanto se incluyen tanto en el diseño del sistema completo de telemetría como en las cotizaciones.

2.3.14. Almacenamiento de datos

Se investiga lo que son las bases de datos relacionales (SQL) y no-relacionales (no-SQL) [66] como formas convencionales de guardar información sin que sea un archivo de texto plano como en eolian fénix. Usar bases de datos como tal permite hacer consultas y filtrar los datos de ser necesario en forma fácil. Además las bases de datos están diseñadas para una alta concurrencia estando optimizadas para aquello.

Las bases de datos relacionales SQL son las más conocidas y ampliamente usadas: corresponden a tablas de datos a las cuales se les puede hacer consultas. Las bases de datos no-relacionales son ampliamente usadas en redes sociales y aplicaciones web donde generalmente el orden global no importa, su clave está en la velocidad en comparación a bases SQL. Se estudian las opciones SQL MariaDB [67] y MySQL [68] mientras que MongoDB [69] como alternativa No-SQL.

Como se quiere velocidad se piensa en primera instancia usar MongoDB, sin embargo, luego de analizar el problema y lo que es más conveniente se prefiere usar tablas relacionales, ya que posteriormente se necesita de los datos ordenados para poder analizarlos. Por lo tanto MariaDB y MySQL quedan como alternativas concretas.

En cuanto a la forma de almacenar los datos, primero se acuerda que la *llave* o *key* [70] de cada tabla será el *timestamp* [71] ya que dos mensajes no pueden tener el mismo timestamp. En segundo lugar se discute si tener una tabla donde cada fila representa todos los datos (con los componentes concatenados), o bien, una tabla por componente. Se escoge implementar la segunda opción ya que a partir de ella se puede generar la primera tabla y además porque los datos quedan mejor predisuestos para su análisis.

Queda propuesto entonces implementar en el programa en Java una clase que pueda conectarse

a una base de datos SQL y guardar en cada instante de actualización los valores de un componente en su respectiva tabla.

2.3.15. Aplicación web como interfaz visual

¿Cómo se visualizarán los datos de eolian? Este problema es transversal tanto en las pantallas al interior de eolian como en el sistema de monitoreo externo. Por otro lado, debe ser afrontado en forma tal de que lo desarrollado para una pantalla no sea incompatible o tan diferente para otra, para no complicar el sistema como un todo dejandolo no-escalable ni flexible. Además lo desarrollado para la Raspberry Pi debe ser compatible con lo desarrollado para el sistema exterior. ¿Existe alguna herramienta tan transversal? Sí, el navegador web.

La mayoría de los dispositivos con conexión a internet soportan alguna forma de visualización web: smartphones, tablets, computadores y también claro, la Raspberry Pi (considerado como un pequeño computador). **Es por esto que nace la idea de desarrollar una aplicación web para visualizar los datos de eolian**, ya que entre otras cosas, se podría monitorear adicionalmente desde celulares y tablets [74]. Un ejemplo de aplicación web sacado de themeforest.net [76] se muestra en la Figura 31 a continuación:

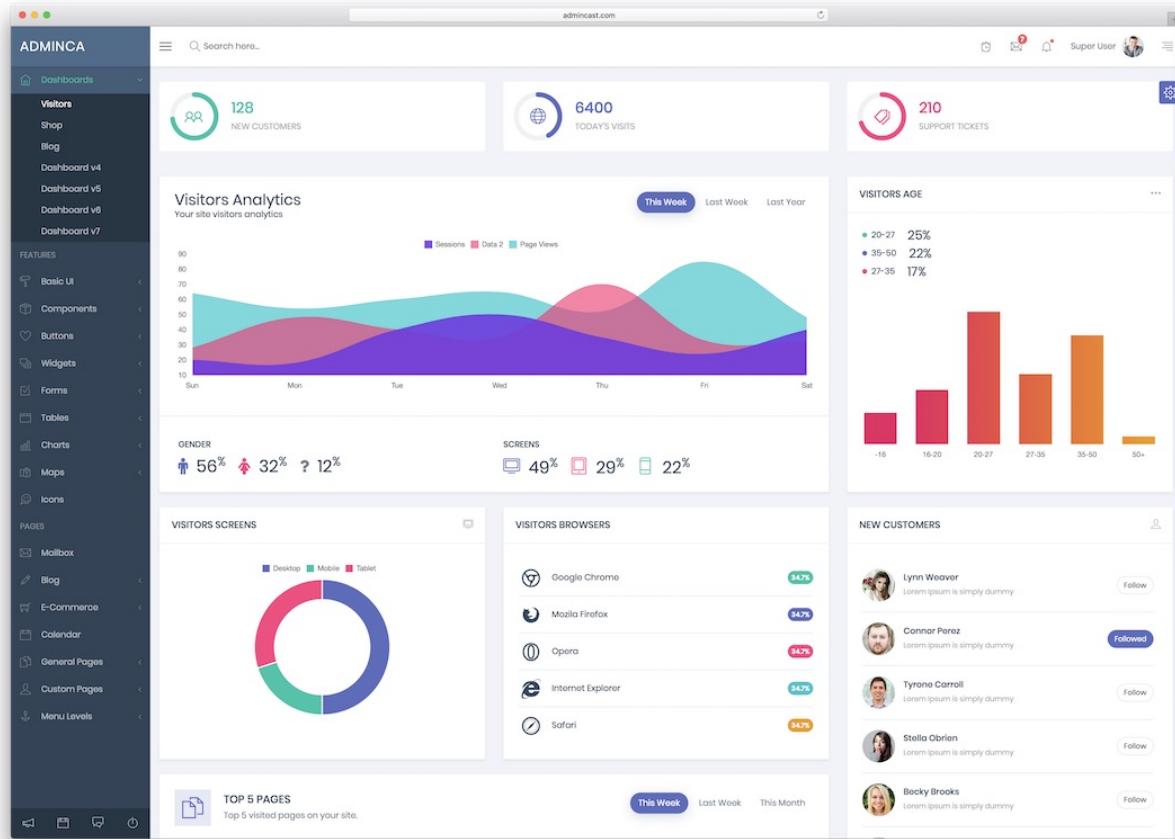


Figura 31: Ejemplo de aplicación web

Básicamente las aplicaciones webs constan de dos ejes de desarrollo: Back-end y Front-end. El primero se encarga de ‘todo lo que no se ve’, de la lógica, de levantar un servidor, de la conexión a una base de datos, de cómo se procesan las interacciones del usuario con el servidor y otras cosas. Por otro lado, Front-end es el desarrollo de ‘todo lo que se ve’, es decir, animaciones, distribuciones de tablas y gráficos en tiempo real, la organización visual de los elementos, etc.

Existen hoy en día cientos sino miles de herramientas o *frameworks* para desarrollar aplicaciones web [75]. Gracias al conocimiento de Analía Banura [77] [78], integrante del equipo desde 2019, se escoge utilizar VueJS [79] por su gran versatilidad, eficiencia, comunidad y buen aspecto. VueJS sirve para desarrollar la parte Front-end de la aplicación web, por lo tanto todo lo visual quedaría en manos de esta herramienta.

Para Back-end, se ha decidido ya trabajar con un programa en Java por el soporte de la librería de las Xbees y la flexibilidad de los patrones de diseño. Por otro lado, se podría montar VueJS sobre una Java Web Application [80], corriendo un servidor en Java y dejando todo el programa en este lenguaje, sin embargo, se utilizará NodeJS [81] para montar el servidor y a la vez montar VueJS sobre NodeJS.

Se escoge NodeJS porque es un entorno de ejecución de JavaScript optimizado para navegadores. Por otro lado, combinar código Java y JavaScript para la parte de servidor y Front-end parece no muy buena idea, ya que VueJS se desarrolla sobre JavaScript, HTML y CSS. Además no sería complicado enviar datos desde la aplicación en Java hacia el servidor con la interfaz web, ya que bastaría enviar los datos mediante HTTP POST [82], tener un *listener u observer* [83] y actualizar los datos en el Front-end.

2.3.16. Tercera línea de trabajo

Adicional a las líneas de trabajo anteriores, ahora se agrega:

1. Leer los componentes eolian mediante CANBUS con alguna de las dos opciones descritas y comunicar el programa de lectura con el programa en Java principal
2. Usar el GPS y acelerómetro descritos junto a la Raspberry Pi. Leer y enviar estos datos al programa en Java principal
3. Guardar los datos de eolian en una base de datos relacional ordenada SQL mediante la aplicación en Java
4. Utilizar una aplicación web desarrollada con VueJS y montada en NodeJS como interfaz visual al interior y exterior de eolian
5. Comunicar la aplicación en Java con la interfaz visual mediante HTTP POST

De esta forma, englobando todos los objetivos hasta este punto señalados, se obtiene un primer diagrama de componentes como se muestra en la Figura 32:

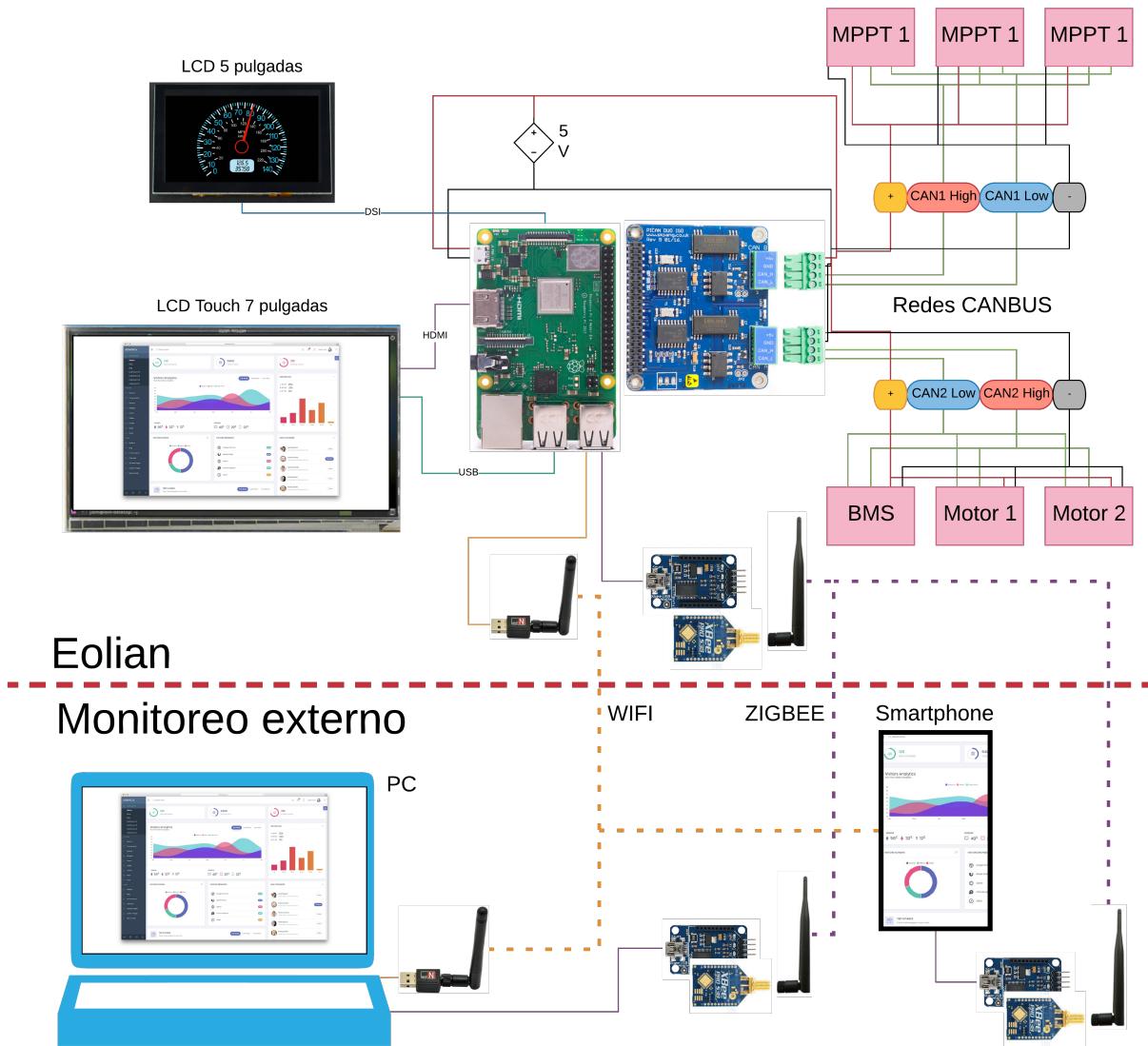


Figura 32: Esquema de conexiones del sistema completo

Como se muestra en la Figura 32, aún habiendo decidido usar ZigBee se da cabida a usar Wifi como plan B. Por otro lado, pueden verse dos redes CANBUS conectadas a las Raspberry Pi mediante la interfaz descrita en las secciones anteriores. Notar que podría conectarse una Xbee a un smartphone mediante un USB-OTG [29] y ejecutar la misma aplicación web desarrollada para el sistema completo. Esto haría posible el monitoreo de eolian desde un celular.

Por otro lado, el flujo de los datos a través de los diferentes programas a desarrollar queda muy bien explicado en la Figura 33:

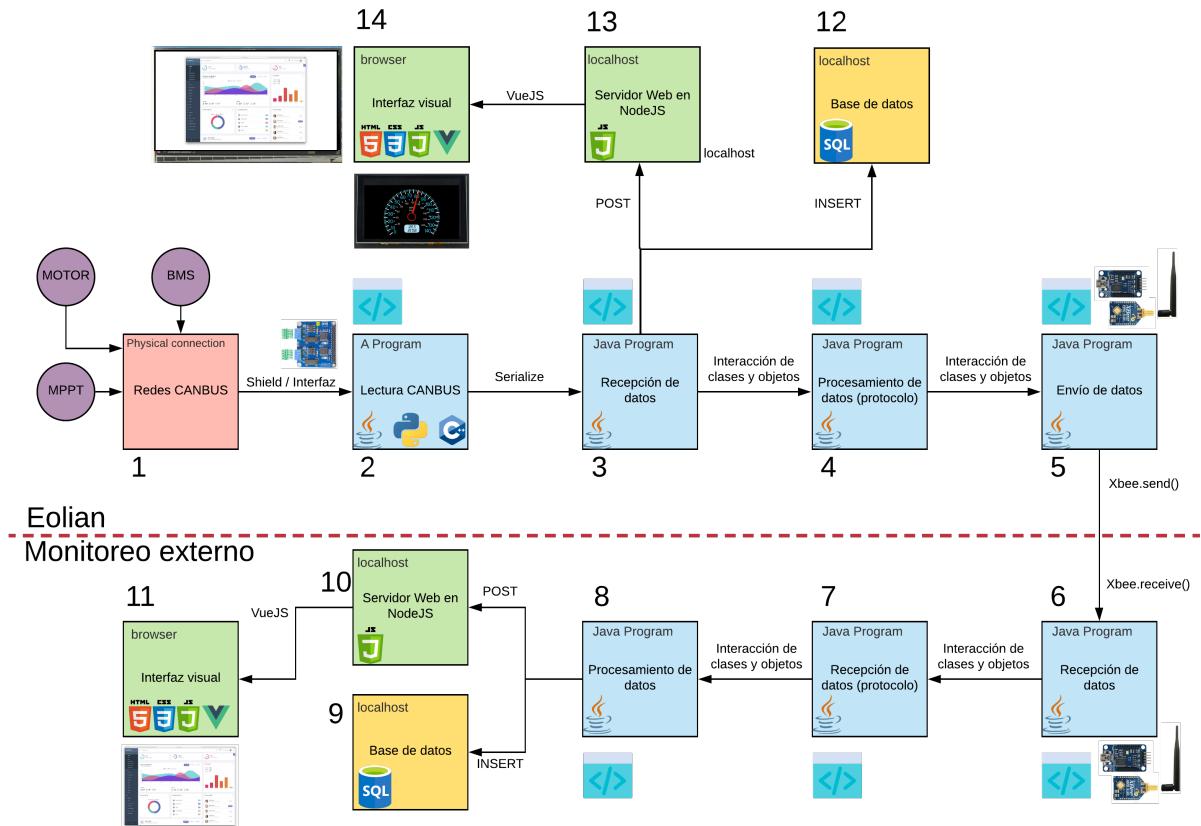


Figura 33: Flujo de datos del sistema completo

Como se muestra en la Figura 33, el paso 1 y 2 dependen del hardware específico para leer por CANBUS los componentes reales de eolian. Sin embargo desde el paso 3 en adelante el desarrollo es independiente, lo cual permite avanzar bastante en el sistema de telemetría sin tener que leer los valores reales de los componentes. Simplemente pueden emularse valores para desarrollar todo el sistema de telemetría.

Los pasos del 3 al 8 conforman el proceso que debe llevar a cabo el programa de telemetría principal en Java. Los pasos 9 y 12 corresponden al proceso de guardado en la base de datos. Los pasos 10 y 13, 11 y 14 corresponden a la visualización de datos a través de la aplicación web en VueJS. El paso 11 representa el monitoreo externo, mientras que el paso 14 la visualización al interior de eolian.

Habiendo diseñado en gran detalle el sistema completo de telemetría eolian y con la ventaja de poder desarrollar sin todo el hardware en la mano, se procede a desarrollar.

2.4. Desarrollo

2.4.1. Metodología de programación

Antes de comenzar a trabajar de lleno en las tareas definidas en la tercera línea de trabajo, se decide usar una metodología que busca evitar errores no previstos, ganando robustez en el programa desarrollado. Dicha metodología se denomina **Test Driven Development** [84]. Esta consiste en hacer *tests* sobre todas las funciones y métodos antes de desarrollarlos, definir contratos y documentarlos [85]. De esta forma se define el comportamiento esperado de un método antes de siquiera programar una línea de código. Al hacer tantos tests, se recorre la mayoría de los posibles valores (sino todos) que un método puede tomar como input, evitando resultados no esperados.

Adicional a esto, se documentará detalladamente cada clase y método desarrollado usando el formato JavaDoc [86] a fin de que generaciones posteriores puedan retomar y entender el trabajo realizado. Además, eventualmente se deberá usar la licencia GPL [18] y es mejor que el código sea entendible para el mundo.

2.4.2. Protocolo entre Xbees

El objetivo general del protocolo a desarrollar es reflejar valores de **Componentes** eolian desde un computador (la Raspberry Pi) hacia otro computador mediante **Mensajes**. Además es súper importante mencionar que ambos computadores saben de antemano la información que deben enviar/recibir lo cual reduce los datos a enviar. No es necesario enviar el detalle de lo que se envía en cada Mensaje, sino que basta de un identificador de Mensaje para que ambos computadores sepan de qué datos están intercambiando. Esto concluye en que sin importar cual sea el protocolo a implementar, si se ejecuta el mismo programa en ambos computadores al inicio, ambos quedarán configurados de igual forma.

En cuanto a responsabilidades, se realiza una separación o independencia de lo que son los **Componentes** de los **Mensajes** a enviar. **Ambas abstracciones quedan como las primeras clases a desarrollar en java**. Los componentes no tienen la responsabilidad de acomodarse entre ellos para ser enviados, sino que un ente externo tiene decirles de alguna forma en qué mensajes serán enviados. Por su parte, los Mensajes forman parte del protocolo y no se encargan de actualizar valores de componentes ni otras funciones. La delegación de responsabilidades es importante para desarrollar clases y objetos.

Cada componente se abstrae con un identificador y un vector o *array* de valores. Ahora bien, aplicando la compresión de datos discutida en las secciones anteriores, cada valor real de un componente debe ser codificado para poder ser enviado. Esta codificación usa el mínimo de bits necesarios para el envío. El usuario debe ingresar valor mínimo, máximo y cantidad de decimales que se quiere para cada valor y el programa calcula los bits necesarios y codifica cada valor del componente. Para decodificar, el programa aplica la función inversa a la codificación. Todo esto queda evidenciado en la Figura 34:

Componente	
Leído directamente	<code>double[] valores = {6.1, 3.11, 1.0}</code>
Predefinido por el usuario	<code>double[] mínimos = {1.0, -100.0, 0.0}</code> <code>double[] máximos = {10.0, 100.0, 1.0}</code> <code>int[] decimales = {1, 2, 0}</code> <code>int[] offset = {-10, 10000, 0}</code>
Deducido por el programa	$= \text{truncar} [-(mínimos \cdot 10^{decimales})]$ $= \text{truncar} [(máximos - mínimos) \cdot 10^{decimales} + 1]$ $= \lceil \log_2 (\deltaelta) \rceil$
Encode	<code>int[] valores_enviar = {51, 10311, 1}</code> $= \text{truncar} [\text{valores} \cdot 10^{decimales}] + \text{offset}$
Decode	<code>double[] valores_recibido = {6.1, 3.11, 1.0}</code> $= (\text{valores_enviar} - \text{offset}) \cdot 10^{-decimales}$

Figura 34: Valores de un componente

Dos cosas le conciernen a un Mensaje respecto de cada valor que tiene que acarrear: el valor a enviar y los bits significativos que usará por valor. Adicional a esto, de manera global debe tener un identificador (para saber qué mensaje es) y un CRC o *checksum* [87] que valide que el mensaje llegó íntegramente sin ser corrompido. Con esta abstracción de un mensaje y con los array de bits significativos y valores a enviar para cada componente, se desea un comportamiento como se muestra en la Figura 35:

	Componente #1	Componente #2	Componente #3						
	int[] bitSignificativos = {7, 15, 1} int[] valores_enviar = {51, 10311, 1}	int[] bitSignificativos = {9, 8, 7, 8, 7} int[] valores_enviar = {511, 255, 0, 33, 31}	int[] bitSignificativos = {4, 4, 4, 4, 15, 11} int[] valores_enviar = {15, 9, 11, 1, 989, 1579}						
Mensaje # 1	Header 0 0 1 0 0 0 0 1 0 8	51 0 1 1 0 0 1 1 0 8	10311 0 1 0 1 0 0 1 0 0 8	1 0 0 1 0 0 0 1 1 8	511 1 1 1 1 1 1 1 1 1 8	255 1 1 1 1 1 1 1 1 1 8	0 0 0 0 0 0 0 0 0 8	33 0 1 0 0 0 0 0 1 8	CRC 0 1 1 0 1 0 1 1 1 8
Mensaje # 2	Header 0 0 1 0 0 0 1 0 8	31 0 0 1 1 1 1 1 8	15 1 1 1 1 1 1 1 8	9 0 1 1 1 1 1 1 8	11 0 1 1 1 1 1 1 8	1 0 1 1 1 1 1 1 8	989 0 0 0 0 0 0 0 1 8	1579 1 0 1 1 1 1 1 1 8	CRC 0 0 0 0 0 0 0 0 8

Figura 35: Ejemplo práctico del protocolo

En este ejemplo se usan 8 bits para el header y 8 bits para el CRC. Esto permite hasta 256 mensajes diferentes y un checksum mínimo. Tampoco es necesario usar checksums más extensos, ya que el protocolo ZigBee incluye un sistema de checksum en la capa de transmisión. Además el mensaje tiene un largo fijo en bytes, en este caso 9 (como se verificó, la Xbee soporta hasta largo 144 bytes).

Los componentes van dispuestos en los mensajes en forma concatenada, al igual que los arrays de valores de cada uno. Notar que los valores van concatenados sin importar la numeración del byte que se encuentran, es decir, que un byte puede acarrear hasta 8 valores de un bit o puede ser parte de un valor de varios bytes. Esto insinúa la creación de un método que **inserte** bits en un array de bytes, y también de un método que de un array de bytes **extraiga** los bits y valores correspondientes. Finalmente notar que si el próximo valor de un componente no cabe al final del

mensaje, este simplemente no se envía. Podría haberse enviado particionado entre el final de un mensaje y el inicio de otro, pero esto hace más complejo al protocolo y dada la frecuencia de envío reflejada en los test de stress de las Xbee realmente no vale la pena darle prioridad a esto.

Existirá entonces un método de inicialización **genMessages()**, que establecerá qué mensajes y en qué espacio específico del mensaje van qué valores específicos de qué componentes y con cuantos bits se van a usar por valor. Esto relacionará los Componentes con los Mensajes y viceversa.

Se identifican y enumeran las diferentes tareas a realizar por el protocolo: lectura de sensores, actualización de valores, codificación de mensajes, envío por xbee, recepción por xbee, decodificación, actualización de valores, display de datos y almacenamiento de datos.

Para el proceso de codificación, los Componentes deberán **recorrer** todos sus Mensajes para actualizarse. Para el proceso de recepción y decodificación, los Mensajes deberán **recorrer** sus Componentes para actualizar sus valores. Este proceso de recorrer objetos se implementa como lista [88]. Por otro lado en la recepción sólo se tiene de un identificador de mensaje, y para llegar al objeto Mensaje en forma eficiente se utiliza un HashMap [89] para obtener el Mensaje correspondiente con costo $O(1)$.

Se decide emplear una estrategia de **multithread** [90] para paralelizar estas funciones. Por ejemplo, es mejor que exista un thread que se encargue sólo de recibir mensajes, mientras que otro de decodificarlos, etc. Para implementar esta estrategia y comunicar los threads se utilizan colas o **queues** [91]. Java tiene una implementación de Queues moderna para multithread denominada **Blocking queue** [92] a la cual se le puede dar un largo máximo y una vez llena, los threads se bloquean (esperan) hasta que existe un espacio en la cola: justo lo que se busca.

Un thread pone elementos en la queue y otro saca elementos de la queue: esto se denomina el patrón de productor-consumidor [93]. Este patrón calza justo con lo que se necesita: leer sensores y poner componentes en cola; tomar componente, codificarlo en mensaje y ponerlo en cola de envío; enviar. Por otro lado: receptionar mensajes y ponerlos en cola; decodificar mensajes y ponerlos en cola de visualización; visualizar valores y guardar en base de datos.

Con todo esto en consideración se crean tres clases (y threads) para la lectura, codificación y envío de valores:

1. **SensorsReader:** Clase que se encarga de la lectura de los sensores reales de eolian, actualizar los valores de los componentes y desencadenar la codificación de los mensajes a enviar que le corresponden al componente. Luego pone los Mensajes en cola para el siguiente thread
2. **SenderAdmin:** Clase que se encarga de tomar Mensajes de la cola, calcular el CRC y poner el array final de bytes en cola para envío
3. **XbeeSender:** Clase que sólo se encarga de tomar arreglos de bytes de la cola y enviarlos con la librería Xbee

La interacción entre estas clases con los métodos principales de cada una queda reflejada en la Figura 36:

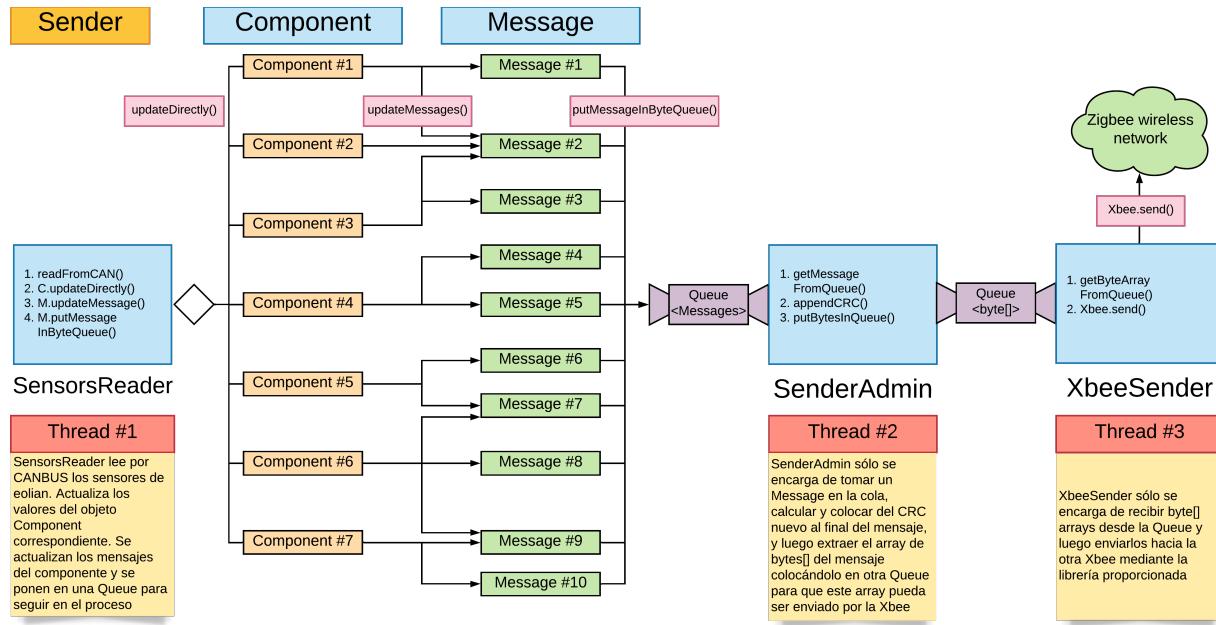


Figura 36: Diagrama de clases y flujo de datos al enviar mensaje

Para el lado de recepción, visualización y almacenamiento se desarrollan las siguientes cinco clases:

1. **XbeeReceiver:** Clase que usa la librería Xbee y sólo recepciona array de bytes y los pone en cola
2. **ReceiverAdmin:** Toma array de bytes de la cola anterior y busca el objeto Mensaje al cual corresponde el array. Lo encuentra con el HashMap, luego actualiza los bytes del objeto y recorre todos los Componentes que le conciernen al Mensaje. Se actualizan los valores de los Componentes y se ponen en cola
3. **LocalMasterAdmin:** Tiene un objeto tipo DataBaseAdmin y ServerAdmin. Toma Componentes de la cola y manda a guardar los valores del componente en la base de datos y a la vez manda el componente hacia la interfaz visual
4. **DataBaseAdmin:** Toma un componente y lo guarda en su tabla correspondiente en la base de datos
5. **ServerAdmin:** Toma un componente y lo envía por HTTP POST hacia el proceso que corre la interfaz visual en NodeJS/VueJS

La interacción entre estas clases con los métodos principales de cada una queda reflejada en la Figura 37:

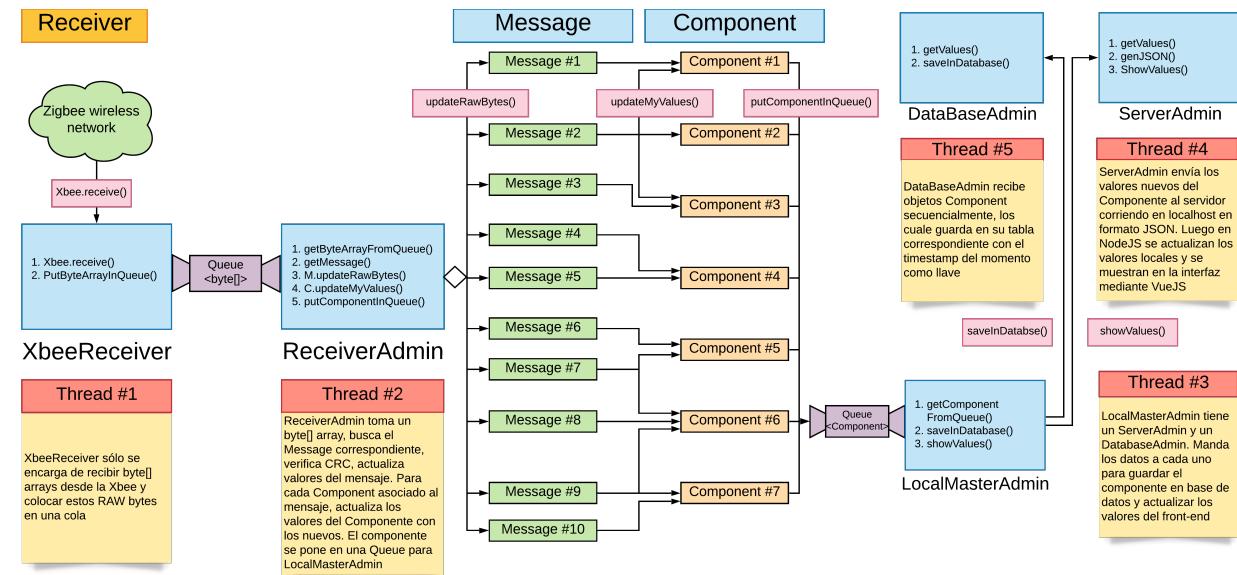


Figura 37: Diagrama de clases y flujo de datos al recibir mensaje

Diagramas en mejor tamaño quedan adjuntos en Anexo B.1 y Anexo B.2.

Definidas las clases y su interacción, se desarrollan los métodos de codificación y decodificación mediante la metodología TDD en ‘modo offline’: sin Xbees. Se crean diversos test bajo la librería de test unitarios JUnit en IntelliJ [94] y pasan exitosamente. Posterior a eso se integra el protocolo con las Xbees y la librería de envío. Se corren los threads y se logran enviar datos desde una Xbee a otra, usando el protocolo. Los datos se simulan con la clase RandomReader, hija de SensorsReader desarrollada justamente para esta función.

A esta altura se tiene el *core* listo del sistema de telemetría y se puede decir ya que SensorsReader (RandomReader), SenderAdmin, XbeeSender, XbeeReceiver, ReceiverAdmin y LocalMasterAdmin quedan totalmente implementadas.

Posteriormente se descargan librerías públicas de Apache y otros [95] [96] para enviar mediante HTTP POST objetos en formato JSON [97] hacia un servidor en NodeJS. Se deja la clase ServerAdmin operativa. Esto quiere decir que el programa en Java ya se comunica con el servidor en NodeJS.

Queda aún como tarea implementar la clase DataBaseAdmin con MySQL o MariaDB, ya que se priorizan las otras clases y se sabe que para conectar una clase con una base de datos local basta de una librería estándar.

2.4.3. Interfaz de visualización

Gracias al trabajo de Analía Banura y los requerimientos acordados por el equipo, se logra dar con una primera versión de la interfaz visual *responsive* [98] que se muestra en la Figura 38:

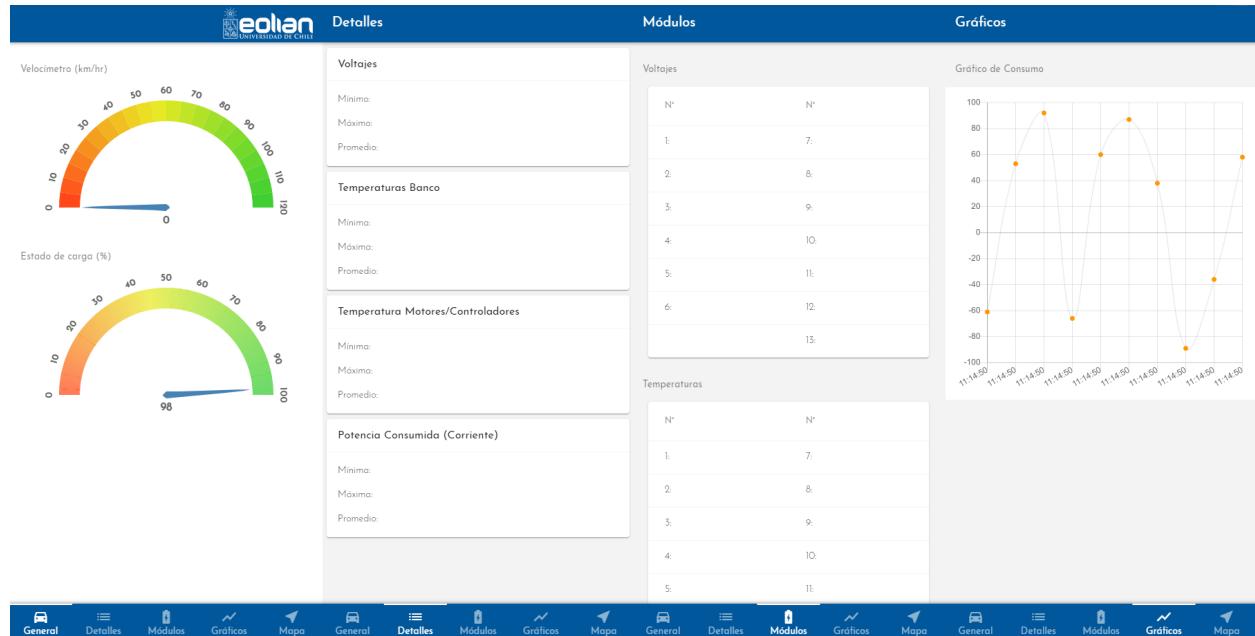


Figura 38: Primera iteración de interfaz gráfica eolian áuriga

Dicha visualización se crea mediante el uso de VueJS como estaba planteado durante el diseño. Lo bueno es que integra todos los componentes base que se necesitan en la visualización: velocímetros, tablas y gráficos. Además, se puede visualizar en pantallas de cualquier tamaño sin perder la estructura central.

Si bien contiene lo mínimo para probar la comunicación y el flujo de datos completo, aún faltan tareas por terminar a la fecha de entrega del presente informe. Dichas tareas se enumeran a continuación:

1. Conectar el servidor Front-end de VueJS con el servidor Back-end de NodeJS que ya se comunica con el programa en Java por HTTP POST
2. Redistribuir los elementos y cambiar los tamaños de acuerdo a un diseño estratégico. Además poner elementos faltantes
3. Diseñar interfaces específicas para las pantallas de 5 y 7 pulgadas
4. Añadir complementos a la interfaz como reproductores de música, mapas entre otros

Cabe mencionar que dichas tareas ya están avanzadas. La primera pretende solucionarse utilizando socket.io [99] que permite la comunicación Front-end Back-end mediante eventos. Para la

segunda y tercera se tienen diversos *mockups* [100] o borradores que deberán ser discutidos con el área de diseño. Para el cuarto se tienen en la mira diversos componentes de Vue para integrar, como por ejemplo un reproductor de Spotify [101].

2.5. Desarrollo adicional

2.5.1. Integración de Eolian Fénix a nueva telemetría

A modo de validación de la nueva telemetría se pretende integrar ésta en eolian fénix mediante el uso de alguna interfaz o componente preexistente a falta de recursos (No es la idea colocar aún el mismo sistema de eolian áuriga dentro de eolian fénix).

Como eolian fénix trabaja con un Arduino MEGA principalmente, se intenta comunicar a éste con un computador que pueda conectarse a las Xbee y enviar los datos a través del protocolo implementado. Notar que el computador puede ser en un futuro una Raspberry Pi, o bien, se podría simplificar todo el sistema pero ahora lo que vale es generar el puente de comunicación.

Para esto se desarrolla en Java la clase ArduinoReader, clase hija de SensorsReader y que tenga como función principal leer el puerto serial conectado al computador desde el arduino. Se incurre en una forma de leer el puerto mediante una librería estándar [102] realizando diversas pruebas ligando leer los prints del arduino en Java.

Se pretende utilizar `Serial.write()` en vez de `Serial.print()` en el arduino para utilizar un mínimo de bits por dato (discusión en sección de compresión de datos). Sin embargo se encuentra el mismo problema del pasado: los datos llegan a Java irregularmente, teniendo que desarrollar un algoritmo caro que lea un buffer y extraiga los mensajes. Sin embargo, se nota que si los valores se envían con `Serial.print()` con valores separados por coma, los valores son fácilmente separados por Java con un splitter, pero más lento y menos compacto.

Por temas de practicidad, complejidad y después de muchas pruebas se decide seguir con el splitter pero para hacer más eficiente el sistema se buscan alternativas al método `Java.String.split()` que se usa por defecto [103]. Se encuentran estudios que comparan diversas formas de splitter, y todos apuntan que el método `indexOf()` es el más rápido [104]. Se deja entonces el método de splitter eficiente, y se logra leer en forma correcta los valores enviados del Arduino a la clase en Java.

Finalmente se acopla la clase ArduinoReader al sistema de telemetría completo, conectando el Arduino MEGA al computador, enviando con éxito mensajes desde el Arduino hasta el servidor en NodeJS, validando el flujo completo de datos y el protocolo de comunicación una vez más.

2.5.2. Navegación por mapa

Se pretende desarrollar un componente de navegación en la interfaz visual. Como se enviarán desde la Raspberry Pi los valores de latitud y longitud a cada instante gracias al GPS Shield, se podría entonces conectar alguna API que visualice estos valores en tiempo real en el Front-end a desarrollar.

Se da con dos opciones: Google Maps [105] y Mapbox [106]. La gran disyuntiva entre cual usar es que se necesita un modo offline, ya que se carecería de internet durante los desafíos en Australia y otros. Google Maps no permite el uso en modo offline mientras que Mapbox sí. Se comienza pues a desarrollar con Mapbox pero se encuentran muchos errores y trabas en el camino, no pudiendo mostrar una latitud y longitud en el Front-end. Por otro lado, al buscar una solución con Google Maps, se encuentra que es ilegal usarlo en modo offline [107] por los términos de servicio.

Se pretende también integrar este navegador con el modelo energético y la optimización de la velocidad en cada punto, ya que esta podría desplegarse en cada instante como 'velocidad objetivo'. Este problema sigue abierto hasta el día de hoy.

2.5.3. LCD base

Al igual que en eolian fénix, se pretende desarrollar un plan B para la visualización de los datos dentro del vehículo. Lo más simple es mostrar los datos más importante de velocidad y consumo mediante un LCD HD44780 comunicado por I2C con la Raspberry Pi (al igual que el acelerómetro).

Se investiga y se encuentra que una simple librería en Java [108] que podría hacer efectivo el uso de este aparato. Mejor aún, quedaría integrado en el programa principal de Java, simplificando la programación.

Se deja propuesto entonces el diseño de todo este pequeño sistema ya que el desarrollo real comienza con el LCD y la Raspberry en la mano.

2.5.4. Conexiones

Habiendo descrito el sistema completo de telemetría, incluído los desarrollos adicionales a la base se deja en la Figura 39 a continuación, un diagrama esquemático de todos los componentes físicos al interior de eolian auriga:

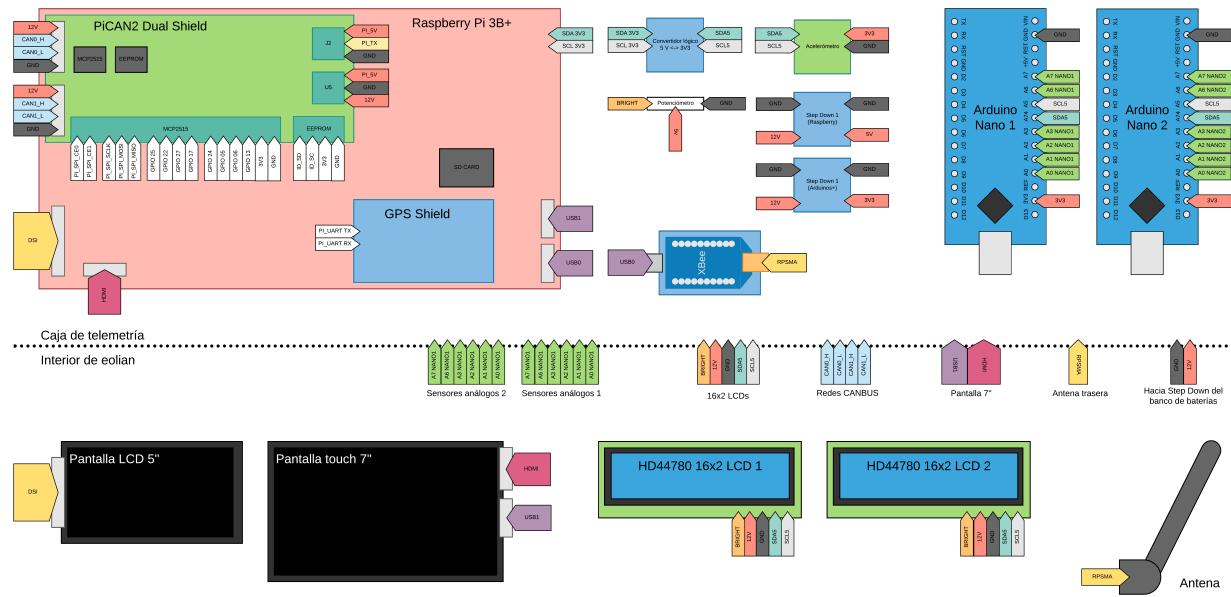


Figura 39: Esquemático de caja de telemetría e interior eolian

2.6. Trabajo futuro

2.6.1. Mejorar y terminar la interfaz

Como se dijo anteriormente, faltan tareas por completar en el desarrollo de la interfaz visual en VueJS. Se dejan estas tareas para el final, debido a que no tenía sentido avanzar en ello sin un protocolo de comunicación robusto.

Este problema tiene una alta prioridad a la fecha de redacción del presente informe.

2.6.2. Integrar radio a Raspberry Pi e interfaz

En una reunión reciente se propone el uso de radios en conjunto con la Raspberry Pi y la pantalla interactiva al interior eolian, reemplazando las radios físicas que se usaban en eolian fénix. Se menciona en la reunión que existen módulos acoplables a la Raspberry Pi.

Se deja todo este problema planteado y sujeto a investigación y revisión, no siendo prioridad aún en las tareas venideras.

2.6.3. Integración de modelo energético

Como se tendrán los datos ordenados en una base de datos SQL, es posible utilizar estos datos para validar un modelo energético elaborado en base a fórmulas y principios físicos de consumo, eficiencia, aerodinámica y condiciones climáticas.

Se delega esta investigación al equipo de modelo energético, no siendo una tarea principal del área de telemetría como tal.

2.6.4. Inteligencia artificial

Se pretende utilizar técnicas de inteligencia artificial como deep learning [109] y librerías como PyTorch [110] o TensorFlow [111] para dos tareas:

1. **Curva de aceleración:** Se quiere utilizar los datos de consumo y picks de corriente, en conjunto con grado de aceleración para estimar la mejor curva de aceleración del eolian. Esto supondría una aceleración eficiente al menos en la partida dejando de depender del piloto. El aprendizaje sería mejor mientras más datos se tengan y más pilotos diferentes manejen al eolian
2. **Cálculo y predicción de consumo:** Se quieren utilizar todas las variables eolian, junto a condiciones ambientales como radiación, velocidad y dirección del viento, inclinación, altura entre otros, para poder estimar el consumo del vehículo en función de todas estas variables. De esta forma se tendría una forma alternativa de calcular el consumo en cada punto con datos reales del auto, sin modelos predefinidos ideales. Serviría para el optimizador del modelo energético

Ahora bien, se dejan estas propuestas planteadas como proyectos para cursos relacionados a minería de datos, machine learning o deep learning, ya que no son tareas principales del área de telemetría. También podrían ser desafíos tomados por alguna nueva rama del equipo, pero se necesitarían nuevos integrantes interesados y motivados por esto.

2.6.5. Comandos para maniobrar a distancia

Al existir comunicación bidireccional eolian-sistema de monitoreo, sería posible enviar comandos hacia eolian para diversas funciones como encender luces, acelerar, frenar, mantener velocidad crucero, cambiar la música, etc. Sin embargo esto acomete un desarrollo extremadamente cuidadoso ya que por ejemplo, si eolian se sale de rango debería existir un mecanismo para que el conductor tome el control.

Esto da lugar, en conjunto con el modelo energético, a que eolian pueda tomar la velocidad óptima en cada punto del camino sin intervención del piloto y por lo tanto, podría recorrer cualquier ruta en forma óptima. Obviamente, sujeto a las intervenciones del piloto para maniobrar como adelantar autos, curvas entre otros.

Todo este desarrollo sí le compete a telemetría ya que las velocidades óptimas podrían simularse y los comandos se podrían testear con los pines GPIO de la Raspberry Pi. Sin embargo, estas propuestas no son prioridad al momento de redactar el presente informe.

2.7. Conclusiones

En primer lugar se puede destacar que la telemetría desarrollada para eolian áuriga sigue una metodología de trabajo bien definida, planificada y con un objetivo claro, a diferencia de la telemetría de eolian fénix donde se aprendía mientras se trabajaba bajo un marco de desarrollo bien rígido y predefinido anteriormente. La flexibilidad de diseñar el sistema de telemetría desde cero en eolian áuriga, en conjunto con el conocimiento adquirido de eolian fénix permite un desarrollo de mayor calidad en la segunda versión del sistema. Esto se refleja, por ejemplo, en el protocolo entre Xbees desarrollado en temas de flexibilidad, robustez, escalabilidad, simpleza y resiliencia.

En segundo lugar se concluye que la investigación exhaustiva de alternativas en conjunto a un diseño de detalles minucioso da frutos en los resultados obtenidos para el segundo sistema de telemetría. El uso de TDD fue un acierto para evitar errores. Por otro lado, es un aporte para cualquier equipo de autos solares ya que el proyecto entero queda bajo la licencia GPLv3.0. Se considera bueno dar bastante tiempo al aprendizaje e investigación ya que permite la obtención de mejores resultados.

En tercer lugar se rescata que el sistema de eolian fénix haya servido para ir de Santiago hasta Arica sin fracasar en el intento. Sin embargo, este sistema dejó abiertas muchas mejoras a realizar. Mejoras que fueron suplidas en el nuevo sistema de telemetría.

En cuarto lugar, se concluye que la cantidad de trabajo realizado en casi cuatro años es bastante para una sola persona, se necesitan al menos dos o tres personas en esta área del equipo ya que sólo realizar el presente informe tomó varias semanas de trabajo. Ver el correcto funcionamiento de todos los componentes físicos y lógicos de todo el sistema es una tarea ardua.

En quinto lugar, señalar que la decisión de implementar la interfaz visual como aplicación web brinda una gran flexibilidad para el futuro, ya que las aplicaciones web tienen mucha proyección como tales, por lo que si se desarrollase una herramienta de trabajo mejor que VueJS podría cambiarse todo el Front-end sin condonar todo el trabajo realizado en el Back-end y/o el programa en Java. Es por esto que el sistema completo también es flexible como tal, se podrían añadir funcionalidades nuevas sin afectar gran parte del programa.

En sexto lugar se recalca que el sistema desarrollado para eolian áuriga es altamente escalable. Agregar más componentes a eolian sólo consta de agregar un nuevo objeto componente en el programa en Java. Añadir funcionalidades extra significaría sólo agregar nuevas clases en Java y que estas se comuniquen con el programa principal, o si las funcionalidades son de Front-end, bastaría con agregar un nuevo componente en VueJS. Podría concluirse que el sistema desarrollado es un sistema base para la integración de nuevas funcionalidades.

En séptimo lugar, señalar que el sistema de eolian áuriga aún está sujeto a su finalización y validación. Para esto se deberá validar con eolian fénix y desde ahí incluir mejoras visuales y de otras índoles como usar la telemetría desde un celular.

Finalmente señalar que es importante tener una constante comunicación con el equipo, ya que telemetría es un área que interactúa con electrónica y modelo energético principalmente y se necesita coordinación para ciertas tareas. Además, telemetría deja un proyección para crear nuevos equipos de trabajo encargados de áreas como inteligencia artificial, minería de datos o diseño digital.

Anexo A. Resultados pruebas de stress Xbees

Tamaño mensaje [Bytes]	Baudrate								
	1200	2400	4800	9600	19200	38400	57600	115200	230400
1	4.94	8.97	15.98	29.20	41.09	62.07	63.63	97.70	76.31
2	9.39	17.03	31.45	57.59	73.49	123.71	130.17	192.96	150.99
3	13.71	25.35	46.42	79.69	104.17	185.38	174.59	280.48	223.93
4	17.41	31.76	60.42	94.07	139.00	244.96	224.67	380.77	302.34
5	21.29	38.41	80.00	111.84	172.93	308.55	287.77	467.81	370.95
6	23.93	44.83	90.76	130.82	207.40	355.51	361.25	572.74	558.66
7	27.49	50.55	98.46	147.96	241.44	419.16	473.26	655.06	595.34
8	30.23	56.16	104.47	164.80	275.64	475.43	528.51	767.61	652.00
9	33.35	61.42	115.20	193.81	308.17	519.42	601.68	835.89	676.59
10	35.65	65.75	128.00	214.25	343.19	581.70	693.58	934.93	841.18
11	38.60	71.30	140.80	233.23	372.88	612.88	722.16	1019.84	994.58
12	40.68	75.80	153.57	251.98	403.10	656.06	768.39	1102.43	907.92
13	42.85	79.33	164.26	267.40	433.13	737.21	810.22	1202.70	989.04
14	44.54	83.97	164.95	286.71	462.84	773.01	859.53	1264.79	1056.05
15	46.61	86.95	164.22	296.24	488.55	791.26	982.83	1065.11	1089.32
16	48.43	90.44	170.67	314.68	505.39	819.67	1056.59	1151.08	1176.99
17	50.14	93.55	181.33	310.93	546.01	826.29	1114.32	1186.99	1227.79
18	52.30	97.79	191.97	307.05	568.02	882.79	1159.94	1246.36	1317.91
19	53.29	100.42	202.66	316.17	592.73	883.43	1228.42	1322.75	1388.08
20	55.17	103.06	213.12	319.98	615.67	890.67	1218.47	1390.24	1431.54
21	56.25	106.85	210.55	329.00	648.91	909.56	1229.72	1481.48	1520.31
22	58.55	109.83	211.85	348.39	675.95	980.52	1310.62	1493.55	1576.72
23	59.21	112.02	210.47	365.07	691.00	1007.71	1353.26	1687.21	1665.46
24	60.89	113.99	219.40	386.64	703.17	1055.64	1406.55	1763.54	1827.18
25	61.89	117.45	228.57	393.59	706.43	1076.24	1483.59	1884.94	2238.34
26	63.23	118.99	237.71	406.75	640.11	1107.99	1521.00	1974.63	1928.93
27	64.25	121.45	246.86	416.06	639.72	1080.43	1619.58	2148.31	1903.15
28	65.25	123.72	249.37	433.09	645.97	1096.79	1676.75	2150.70	1968.64
29	66.44	125.72	244.25	445.09	675.36	1131.80	1714.15	2252.08	2149.10
30	67.20	127.45	234.38	454.44	703.70	1202.26	1724.83	2264.32	2146.23
31	68.48	129.69	238.26	465.42	703.97	1271.59	1701.89	2359.21	2194.38
32	69.07	131.64	242.00	456.95	722.43	1315.14	1762.50	2331.51	2254.31
33	70.38	132.98	246.11	452.81	762.27	1342.55	1799.44	2408.58	2239.57
34	70.69	135.62	254.78	463.95	792.49	1388.32	1841.42	2482.11	2332.92
35	72.17	136.43	257.00	459.38	805.67	1430.50	1884.76	2494.83	2471.05
36	72.15	138.46	256.76	456.57	810.21	1378.57	1792.29	2549.94	2412.38
37	72.59	139.57	258.35	463.78	835.86	1403.48	1813.01	2589.77	2669.55
38	73.04	141.31	262.01	482.42	861.11	1447.29	1809.70	2724.01	2931.42

39	73.83	142.97	261.96	495.06	866.55	1500.12	1809.41	2756.18	2922.88
40	74.52	144.14	264.49	501.32	888.10	1523.81	1826.82	2863.69	3147.62
41	75.17	145.26	272.17	510.74	903.52	1572.87	1862.87	2928.15	3371.16
42	75.96	146.93	274.97	522.98	913.82	1566.11	1882.73	2948.20	3141.83
43	76.66	149.89	278.87	530.77	927.70	1588.18	1968.59	2995.26	3113.91
44	77.33	149.97	276.38	539.76	934.16	1604.55	2029.43	3106.91	3405.31
45	78.46	150.51	279.33	538.59	950.05	1621.04	2036.38	3119.58	3465.00
46	79.48	151.91	282.13	535.97	960.59	1669.57	2132.99	3108.32	3304.60
47	79.74	153.15	284.21	535.25	978.64	1668.68	2180.27	3193.59	3621.79
48	80.76	154.78	292.10	538.77	992.64	1703.52	2222.22	3228.19	3570.63
49	80.71	155.44	295.10	537.99	1015.75	1719.66	2166.13	3289.25	4125.27
50	81.99	156.74	295.44	538.96	1022.85	1730.34	2171.08	3340.91	4057.78
51	81.91	158.40	296.00	539.63	1028.93	1731.87	2176.05	3316.21	4147.69
52	82.83	158.55	293.29	552.35	1043.95	1773.05	2102.37	3400.03	4186.46
53	82.91	159.22	295.05	559.85	1038.44	1809.55	2174.63	3451.42	4173.23
54	83.48	161.96	298.22	570.58	1013.10	1846.97	2240.57	3494.47	4295.60
55	84.04	162.33	302.72	579.73	973.24	1878.67	2309.47	3515.05	4034.33
56	83.63	162.13	308.17	585.65	972.90	1836.31	2374.39	3504.16	4516.13
57	83.87	163.50	310.49	590.96	989.82	1852.45	2343.27	3343.30	3879.93
58	84.62	164.35	310.89	598.63	1000.00	1893.14	2370.25	3379.36	4037.87
59	85.83	164.98	311.84	595.29	1020.09	1920.64	2355.20	3411.79	4249.50
60	85.98	165.41	313.81	599.95	1036.39	1921.54	2346.41	3453.04	4354.14
61	86.72	166.72	313.24	586.67	1043.82	1974.17	2416.03	3504.34	4474.11
62	86.17	168.61	313.82	588.13	1055.19	1955.71	2444.41	3508.77	4540.46
63	86.51	168.57	315.18	589.45	1063.54	1934.65	2489.72	3523.69	4667.36
64	86.70	169.56	320.65	590.49	1074.19	1949.61	2576.70	3541.59	4739.69
65	87.26	170.22	323.29	599.30	1087.41	1988.07	2532.73	3551.33	4769.24
66	87.45	171.18	323.26	602.24	1103.05	2008.03	2509.22	3533.00	4848.66
67	87.71	171.23	324.30	603.71	1111.28	2049.74	2467.59	3842.84	4903.40
68	88.18	171.99	326.29	616.00	1120.71	2075.20	2473.09	3953.26	4915.78
69	88.54	172.63	327.50	621.67	1125.92	2094.65	2473.65	3990.52	5070.92
70	88.67	174.84	330.76	631.59	1132.41	2126.30	2515.09	4077.12	5072.83
71	88.90	174.43	332.78	638.35	1144.61	2130.98	2583.13	4111.65	5157.63
72	89.58	175.04	333.99	652.22	1147.81	2096.92	2594.31	4027.97	5257.01
73	89.82	175.91	332.10	658.04	1154.97	1970.63	2661.42	3876.59	5415.43
74	90.01	176.51	333.36	672.05	1161.95	1977.29	2709.13	3756.35	5373.61
75	90.24	176.85	337.42	680.66	1174.22	1996.54	2768.24	3877.77	5356.00
76	90.55	177.14	335.42	673.67	1184.52	1966.26	2672.20	3528.48	5392.75
77	90.76	178.42	338.70	651.69	1183.32	1983.87	2698.44	3512.29	5440.16
78	91.03	178.29	340.86	662.50	1191.11	2014.67	2760.57	3436.43	5591.40
79	91.31	178.71	339.78	658.50	1206.25	2018.55	2778.66	3434.19	5596.49
80	91.72	179.87	340.42	652.31	1204.17	2051.07	2802.79	3539.04	5599.89
81	91.84	180.58	343.98	652.75	1188.50	2077.72	2841.11	3526.34	5721.96
82	92.09	180.53	344.56	659.88	1184.61	2085.98	2832.27	3574.23	5862.59

83	92.38	180.78	344.33	667.53	1173.53	2103.24	2884.05	3689.87	5807.44
84	92.47	182.02	346.14	672.22	1165.81	2102.89	2899.15	3589.90	5840.63
85	92.98	182.74	345.53	680.11	1170.27	2115.59	2975.46	3740.21	5735.49
86	93.00	182.76	347.58	687.22	1178.05	2140.37	3012.68	3833.81	5842.39
87	93.54	182.82	348.83	695.39	1184.37	2180.18	3009.13	3801.28	5998.35
88	93.49	184.63	352.00	703.39	1195.25	2205.96	2999.93	3852.38	5957.22
89	93.98	184.29	355.61	698.88	1211.59	2227.45	3014.70	3852.65	6097.98
90	94.00	184.54	355.25	690.40	1213.35	2249.72	2975.60	3909.13	6030.15
91	94.40	183.84	353.70	684.50	1223.96	2234.94	3025.37	3944.52	6110.66
92	94.38	184.92	355.60	678.62	1231.63	2255.45	3039.41	4003.48	6095.54
93	94.67	186.12	355.11	677.34	1219.19	2262.88	3092.58	4029.29	6167.52
94	94.26	187.81	359.05	673.14	1252.97	2274.05	3132.39	3999.66	6217.75
95	95.00	187.55	359.45	679.69	1253.73	2285.19	3164.56	3981.39	6231.96
96	95.99	187.70	358.46	691.78	1245.69	2213.87	3081.57	3990.02	6224.07
97	95.74	187.74	359.40	696.82	1254.30	2263.92	3088.39	4008.10	6290.94
98	95.76	187.76	358.84	697.88	1252.81	2284.17	3131.59	4029.94	6330.34
99	95.97	188.75	359.75	704.14	1273.05	2293.26	3140.66	4121.57	6431.08
100	96.08	189.15	360.60	710.77	1288.81	2302.45	3123.24	4684.50	6527.84
101	96.26	189.38	362.44	718.12	1290.06	2322.69	3174.60	4201.33	6480.17
102	96.45	189.45	364.00	724.75	1292.92	2323.62	3213.00	4162.93	6545.17
103	96.61	190.29	363.70	726.46	1295.45	2347.04	3228.23	4241.48	6589.05
104	96.70	190.67	364.12	716.76	1304.44	2304.66	3243.51	4281.24	6618.30
105	97.04	190.76	367.28	710.36	1300.44	2308.00	3228.88	4375.36	6648.52
106	97.21	191.05	370.24	707.37	1292.12	2324.31	3283.67	4577.05	6794.00
107	97.40	191.88	374.11	703.80	1304.66	2325.23	3283.42	4716.15	6784.60
108	97.55	191.78	368.90	697.88	1311.87	2327.08	3279.88	4804.91	6820.77
109	97.80	191.74	370.82	700.67	1314.82	2346.05	3299.23	4815.98	6854.48
110	97.87	192.50	370.51	709.02	1297.78	2338.24	3276.25	4905.24	6968.64
111	98.06	193.57	370.31	711.10	1295.91	2324.51	3281.79	4920.87	7049.86
112	98.12	193.19	371.14	714.29	1285.80	2337.67	3294.02	4934.36	7132.40
113	98.37	193.43	371.68	726.60	1292.05	2350.25	3230.79	4974.03	7051.48
114	98.42	194.33	370.49	733.83	1297.67	2361.96	3211.99	5009.45	7212.91
Tiempo total test [s]	77400	39958	20986	11146	6322	3594	2749	1986	1591

Anexo B. Diagrama de clases y flujo de datos del protocolo

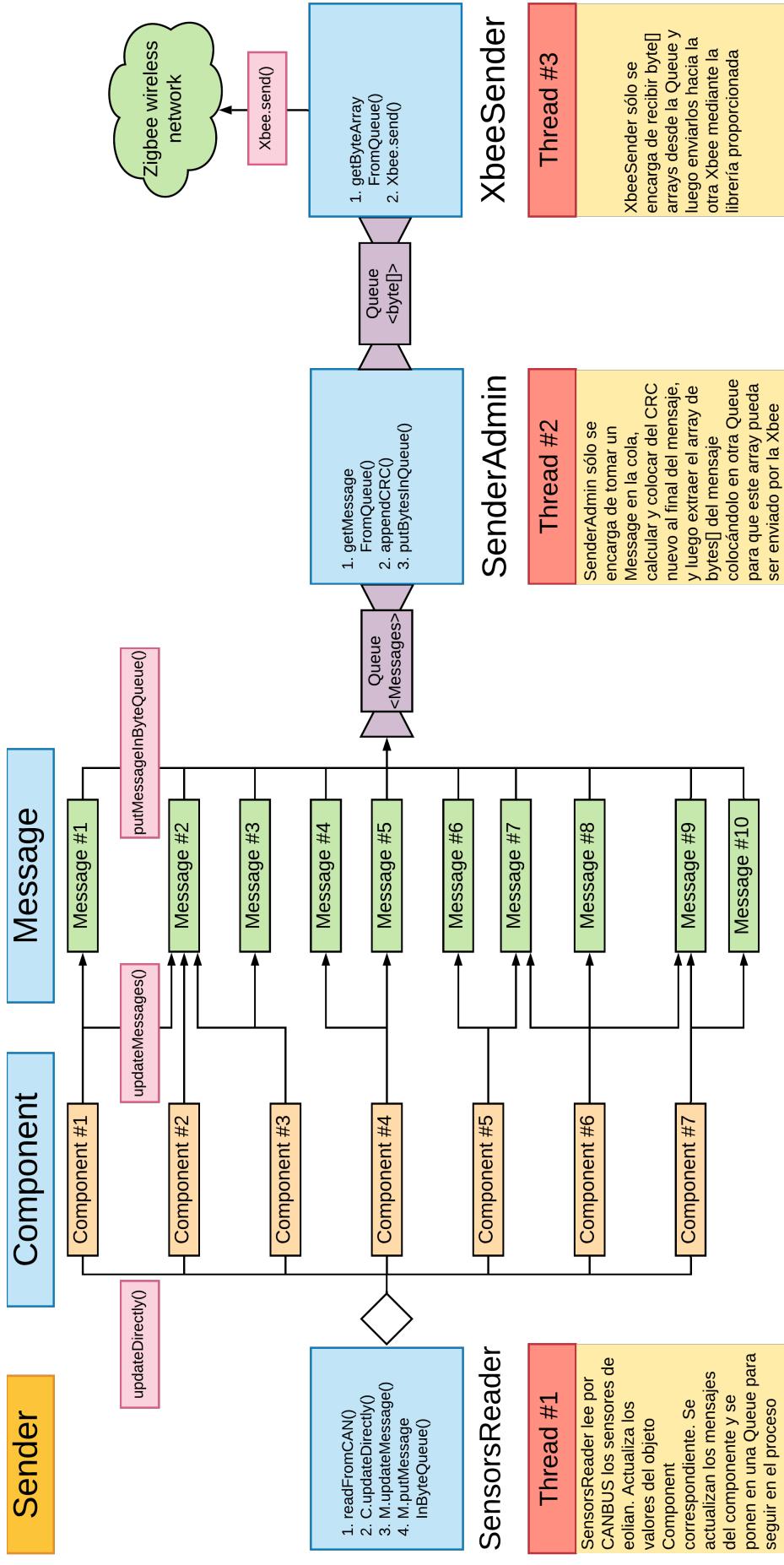


Figura B.1: Diagrama de clases y flujo de datos al enviar mensaje

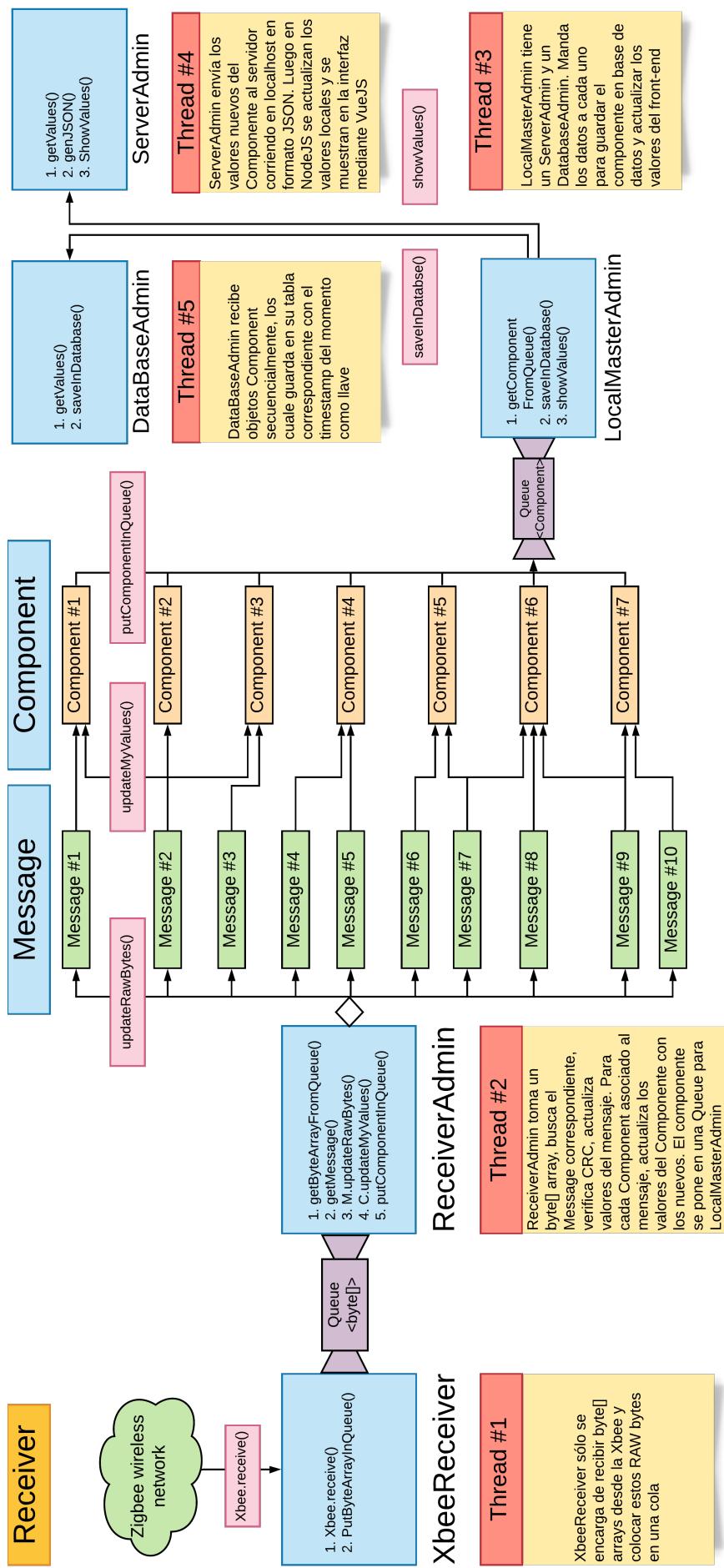


Figura B.2: Diagrama de clases y flujo de datos al recibir mensaje

Anexo C. Esquemático de conexiones

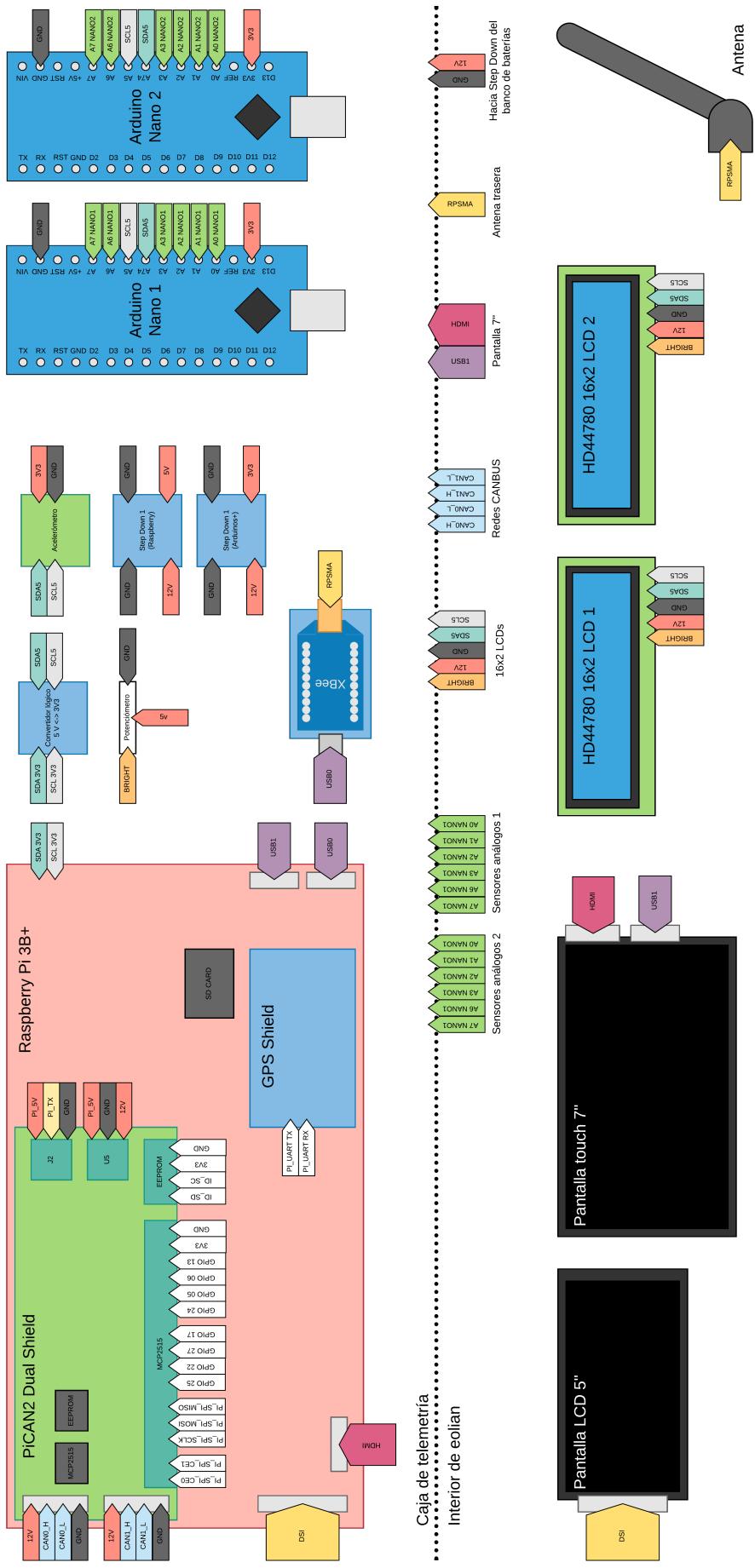


Figura C.1: Esquemático de caja de telemetría e interior eolian áuriga

Anexo D. Esquemático de conexiones eolian fénix

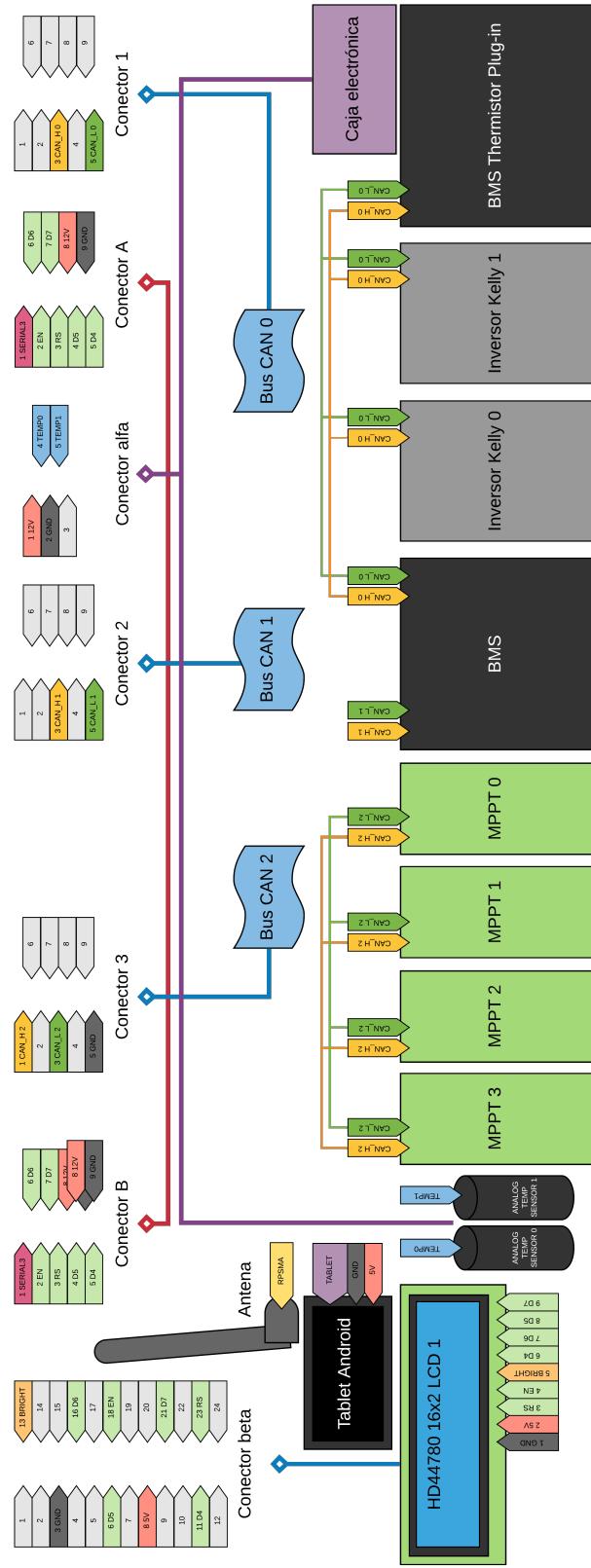
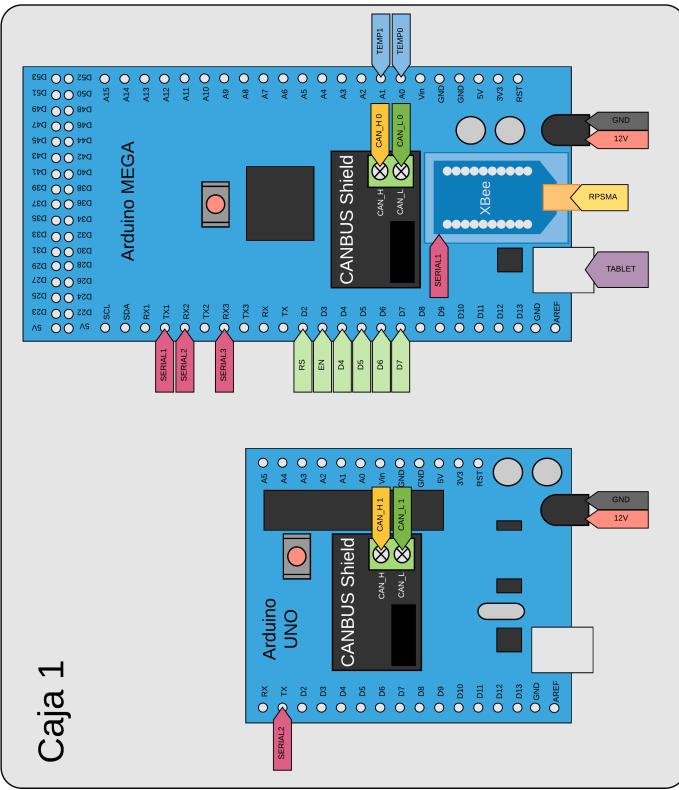
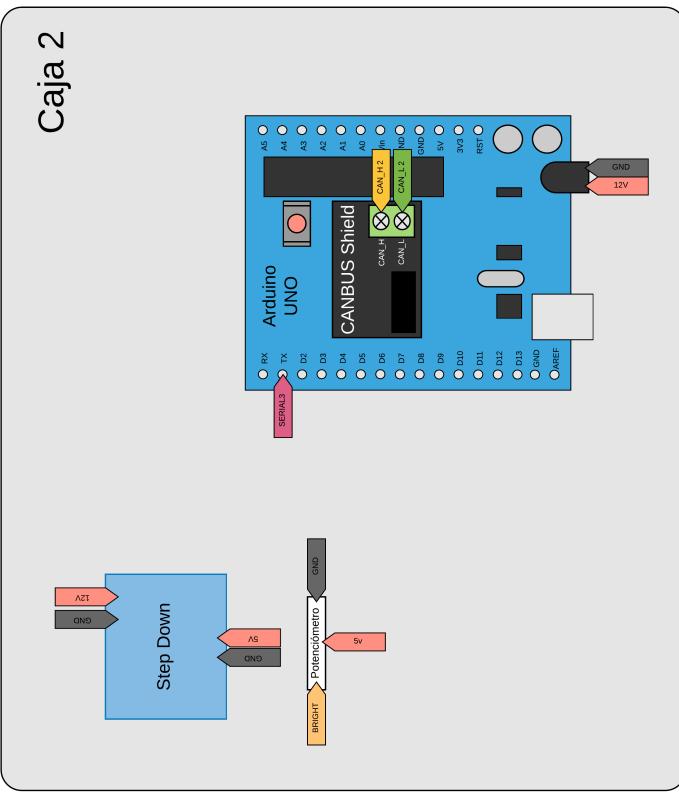


Figura D.1: Esquemático de caja de telemetría e interior eolian fénix

Referencias

- [1] LinkedIn Bryan Bizarro “Bryan Bizarro”. [En línea]. Disponible en: <https://www.linkedin.com/in/bryan-bizarro>
- [2] LinkedIn Sebastián Díaz “Sebastián Díaz”. [En línea]. Disponible en: <https://www.linkedin.com/in/sebasti%C3%A1n-alexis-d%C3%ADaz-vega-a2096826/>
- [3] Orion BMS “Orion BMS”. [En línea]. Disponible en: <https://www.orionbms.com/>
- [4] Orion BMS “Thermistor Expansion Module”. [En línea]. Disponible en: <https://www.orionbms.com/products/thermistor-expansion-module/>
- [5] Kelly “KHB - High Power Opto-Isolated Brushless Motor Controller With Regen (72V-144V (150A-1000A)”. [En línea]. Disponible en: <https://www.kellycontroller.com/shop/khb/>
- [6] Drivetek “USER MANUAL MPPT RACE VERSION 4”. [En línea]. Disponible en: <https://fdocuments.in/document/user-manual-mppt-race-version-4.html>
- [7] Arduino Store “Arduino Uno Rev3”. [En línea]. Disponible en: <https://store.Arduino.cc/Arduino-uno-rev3>
- [8] Seeed Wiki “CAN-BUS Shield V1.2”. [En línea]. Disponible en: http://wiki.seeedstudio.com/CAN-BUS_Shield_V1.2/
- [9] Wikipedia “CAN bus”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/CAN_bus
- [10] DiGi “Xbee Ecosystem”. [En línea]. Disponible en: <https://www.digi.com/xbee>
- [11] Qt Company “Develop Fluid, High-Performance UIs and Applications”. [En línea]. Disponible en: <https://www.qt.io/product>
- [12] Arduino Store “Arduino MEGA 2560 REV3”. [En línea]. Disponible en: <https://store.Arduino.cc/Arduino-mega-2560-rev3>
- [13] Arduino Reference “SoftwareSerial”. [En línea]. Disponible en: <https://www.Arduino.cc/en/Reference/SoftwareSerial>
- [14] DiGi “Next Generation Configuration Platform for XBee/RF Solutions”. [En línea]. Disponible en: <https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu>
- [15] Sparkfun “SparkFun XBee Shield”. [En línea]. Disponible en: <https://www.sparkfun.com/products/12847>
- [16] github.com/vannevar-morgan/ “Qt-Temperature-Sensor”. [En línea]. Disponible en: <https://github.com/vannevar-morgan/Qt-Temperature-Sensor>
- [17] youtube.com Vannevar Morgan “Qt & Arduino - Reading a Temperature Sensor - Part 1: Arduino”. [En línea]. Disponible en: <https://www.youtube.com/watch?v=1PNn63P793Y>
- [18] GNU.org “GNU General Public License”. [En línea]. Disponible en: <https://www.gnu.org/>

- [licenses/gpl-3.0.html](http://www.gnu.org/licenses/gpl-3.0.html)
- [19] Wikipedia “WYSIWYG”. [En línea]. Disponible en: <https://es.wikipedia.org/wiki/WYSIWYG>
- [20] Arduino Reference “Serial.print()”. [En línea]. Disponible en: <https://www.Arduino.cc/reference/en/language/functions/communication/serial/print/>
- [21] Wikipedia “Data buffer”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Data_buffer
- [22] Wikipedia “Analysis of algorithms”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Analysis_of_algorithms
- [23] Arduino Reference “Serial.write()”. [En línea]. Disponible en: <https://www.Arduino.cc/reference/en/language/functions/communication/serial/write/>
- [24] qcustomplot.com “Qt Plotting Widget QCustomPlot”. [En línea]. Disponible en: <https://www.qcustomplot.com/>
- [25] dfrobot.com “SD Module (Arduino Compatible)”. [En línea]. Disponible en: <https://www.dfrobot.com/product-163.html/>
- [26] create.arduino.cc “SD Card Module with Arduino: How to Read/Write Data”. [En línea]. Disponible en: <https://create.Arduino.cc/projecthub/electropeak/sd-card-module-with-Arduino-how-to-read-write-data-37f390>
- [27] developer.android.com “Download Android Studio and SDK”. [En línea]. Disponible en: <https://developer.android.com/studio>
- [28] Oracle “java Programming Language”. [En línea]. Disponible en: <https://docs.oracle.com/javase/7/docs/technotes/guides/language/>
- [29] androidayuda.com “¿Qué es el USB OTG? Gracias a él, tu móvil es mucho más ordenador de lo que parece”. [En línea]. Disponible en: <https://androidayuda.com/android/que-es/usb-otg/>
- [30] LinkedIn Danilo Sánchez “Danilo Sánchez”. [En línea]. Disponible en: <https://www.linkedin.com/in/danilo-sanchez-c/>
- [31] Arduino “Hello World with LCD”. [En línea]. Disponible en: <https://www.Arduino.cc/en/Tutorial>HelloWorld>
- [32] Arduino Reference “analogRead()”. [En línea]. Disponible en: <https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/>
- [33] Wikipedia “Datasheet”. [En línea]. Disponible en: <https://en.wikipedia.org/wiki/Datasheet>
- [34] Sparkfun “GPS Shield Hookup Guide”. [En línea]. Disponible en: <https://learn.sparkfun.com/tutorials/gps-shield-hookup-guide/all>

- [35] Microsoft y Github Inc. “Github.com”. [En línea]. Disponible en: <https://github.com>
- [36] Wikipedia “Object-oriented programming”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Object-oriented_programming
- [37] predictabledesigns.com “Comparison of Wireless Technologies”. [En línea]. Disponible en: <https://predictabledesigns.com/>
- [38] DigiKey “Comparing Low-Power Wireless Technologies (Part 1)”. [En línea]. Disponible en: <https://www.digikey.com/en/articles/techzone/2017/oct/comparing-low-power-wireless-technologies>
- [39] Raspberry Pi Organization “Setting up a Raspberry Pi as a Wireless Access Point”. [En línea]. Disponible en: <https://www.Raspberrypi.org/documentation/configuration/wireless/access-point.md>
- [40] Asaduzzaman, Abu & Chidella, Kishore konda & Ph. D., M.. (2015). “A time and energy efficient parking system using Zigbee communication protocol”. 2015. 10.1109/SECON.2015.7132927. Disponible en: https://www.researchgate.net/publication/283633635_A_time_and_energy_efficient_parking_system_using_Zigbee_communication_protocol
- [41] Wikipedia “Acknowledgement (data networks)”. Disponible en: [https://en.wikipedia.org/wiki/Acknowledgement_\(data_networks\)](https://en.wikipedia.org/wiki/Acknowledgement_(data_networks))
- [42] brainboxes.com “What is RTS / CTS Hardware Flow Control?l”. Disponible en: <http://www.brainboxes.com/faq/items/what-is-rts--cts-hardware-flow-control->
- [43] JetBrains “IntelliJ IDEA”. Disponible en: <https://www.jetbrains.com/es-es/idea/>
- [44] Wikipedia “Data type”. Disponible en: https://en.wikipedia.org/wiki/Data_type
- [45] Raspberry Pi Organization “Raspberry Pi”. [En línea]. Disponible en: <https://www.Raspberrypi.org/>
- [46] Raspberry Pi 3B+ “Raspberry Pi 3 Model B+”. [En línea]. Disponible en: <https://www.Raspberrypi.org/products/Raspberry-pi-3-model-b-plus/>
- [47] androidcentral.com “Can you use multiple monitors with the Raspberry Pi 3 B+?”. [En línea]. Disponible en: <https://www.androidcentral.com/can-you-use-multiple-monitors-Raspberry-pi-3-b>
- [48] amgkits.com “Pantalla HDMI 7 pulgadas Touch para Raspberry Pi”. [En línea]. Disponible en: <https://amgkits.com/Raspberry-pi/289-pantalla-hdmi-7-pulgadas-touch-para-Raspberry-pi.html>
- [49] dfrobot.com “5” 800x480 TFT Raspberry Pi DSI Touchscreen(Compatible with Raspberry Pi 3B/3B+)”. [En línea]. Disponible en: <https://www.dfrobot.com/product-1784.html>
- [50] Gordon Hollingworth, Raspberry Pi Blog “The Eagerly Awaited Raspberry Pi Display”. Disponible en : <https://www.Raspberrypi.org/blog/>

- the-eagerly-awaited-Raspberry-pi-display/
- [51] DiGi “Do more with API mode: XBee libraries”. [En línea]. Disponible en: https://www.digi.com/resources/documentation/Digidocs/90001456-13/concepts/c_xbee_libraries_api_mode.htm?tocpath=XBee%20API%20mode%7C_____6
 - [52] DiGi “Xbee Python Library”. [En línea]. Disponible en: <https://xbplib.readthedocs.io/en/latest/>
 - [53] educba.com “Java Performance vs Python”. [En línea]. Disponible en: <https://www.educba.com/java-performance-vs-python/>
 - [54] Geeksforgeeks “IEEE Standard 754 Floating Point Numbers”. [En línea]. Disponible en: <https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/>
 - [55] Wikipedia “Data compression”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Data_compression
 - [56] techiedelight.com “Huffman Coding Compression Algorithm”. [En línea]. Disponible en: <https://www.techiedelight.com/huffman-coding/>
 - [57] Wikiwand.com “Universal code (data compression)”. [En línea]. Disponible en: [https://www.wikiwand.com/en/Universal_code_\(data_compression\)](https://www.wikiwand.com/en/Universal_code_(data_compression))
 - [58] Geeksforgeeks “Online Algorithm”. [En línea]. Disponible en: <https://www.geeksforgeeks.org/online-algorithm/>
 - [59] Wikipedia “Elias gamma coding”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Elias_gamma_coding
 - [60] Copperhill Technologies “PiCAN2 Duo CAN-Bus Board for Raspberry Pi”. [En línea]. Disponible en: <https://copperhilltech.com/pican2-duo-can-bus-board-for-Raspberry-pi/>
 - [61] Seeed “Serial CAN-BUS Module based on MCP2551 and MCP2515”. [En línea]. Disponible en: <https://www.seeedstudio.com/Serial-CAN-BUS-Module-based-on-MCP2551-and-MCP2515.html>
 - [62] Arduino Organization “Arduino MEGA 2560 REV3”. [En línea]. Disponible en: <https://store.Arduino.cc/Arduino-mega-2560-rev3>
 - [63] Adafruit “Adafruit Ultimate GPS HAT for Raspberry Pi A+/B+/Pi 2/Pi 3 - Mini Kit”. [En línea]. Disponible en: <https://www.adafruit.com/product/2324>
 - [64] pimylifeup.com “Raspberry Pi Accelerometer using the ADXL345”. [En línea]. Disponible en: <https://pimylifeup.com/Raspberry-pi-accelerometer-adxl345/>
 - [65] I2C Protocol “I2C Bus, Interface and Protocol”. [En línea]. Disponible en: <https://i2c.info/>
 - [66] Geeksforgeeks “Difference between SQL and NoSQL”. [En línea]. Disponible en: <https://www.geeksforgeeks.org/difference-between-sql-and-nosql/>
 - [67] MariaDB Foundation “MariaDB Server: The open source relational database”. [En línea]. Dis-

- ponible en: <https://mariadb.org/>
- [68] MySQL “The world’s most popular open source database”. [En línea]. Disponible en: <https://www.mysql.com/>
- [69] mongodb.com “La base de datos líder para aplicaciones modernas”. [En línea]. Disponible en: <https://www.mongodb.com/>
- [70] w3schools.com “SQL PRIMARY KEY Constraint”. [En línea]. Disponible en: https://www.w3schools.com/sql/sql_primarykey.asp
- [71] Wikipedia “Timestamp”. [En línea]. Disponible en: <https://en.wikipedia.org/wiki/Timestamp>
- [72] Wikipedia “Pipeline (Unix)”. [En línea]. Disponible en: [https://en.wikipedia.org/wiki/Pipeline_\(Unix\)](https://en.wikipedia.org/wiki/Pipeline_(Unix))
- [73] pypi.org “Module for serializing and de-serializing Java objects”. [En línea]. Disponible en: <https://pypi.org/project/javaobj-py3/>
- [74] Wikipedia “Web Application”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Web_application
- [75] Wikipedia “Web Application Development”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Web_application_development
- [76] themeforest.net “Web Application Development”. [En línea]. Disponible en: <https://bit.ly/2EA4Lv7>
- [77] Github.com “Analía Banura”. [En línea]. Disponible en: <https://github.com/analiabs>
- [78] LinkedIn Analía Banura “Analía Banura”. [En línea]. Disponible en: <https://www.linkedin.com/in/anal%C3%A1Da-bannura-schultz-bb2070186/>
- [79] Vue.js “The Progressive JavaScript Framework”. [En línea]. Disponible en: <https://vuejs.org>
- [80] edureka.co “What Is a Java Web Application?”. [En línea]. Disponible en: <https://www.edureka.co/blog/java-web-application/>
- [81] NodeJS Organization “Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome”. [En línea]. Disponible en: <https://nodejs.org/es/>
- [82] developer.mozilla.org “POST”. [En línea]. Disponible en: <https://developer.mozilla.org/es/docs/Web/HTTP/Methods/POST>
- [83] Wikipedia “Observer pattern”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Observer_pattern
- [84] agiledata.org “Introduction to Test Driven Development”. [En línea]. Disponible en: <http://agiledata.org/essays/tdd.html>
- [85] Geeksforgeeks “Differences between Testing and Debugging”. [En línea]. Disponible en: [https:](https://)

- //www.geeksforgeeks.org/differences-between-testing-and-debugging/
- [86] Oracle “How to Write Doc Comments for the Javadoc Tool”. [En línea]. Disponible en: <https://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>
- [87] Wikipedia “Cyclic redundancy check”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Cyclic_redundancy_check
- [88] Wikipedia “List (abstract data type)”. [En línea]. Disponible en: [https://en.wikipedia.org/wiki/List_\(abstract_data_type\)](https://en.wikipedia.org/wiki/List_(abstract_data_type))
- [89] dzone.com “Java HashMap Implementation in a Nutshell”. [En línea]. Disponible en: <https://dzone.com/articles/custom-hashmap-implementation-in-java>
- [90] Wikipedia “Multithreading (computer architecture)”. [En línea]. Disponible en: [https://en.wikipedia.org/wiki/Multithreading_\(computer_architecture\)](https://en.wikipedia.org/wiki/Multithreading_(computer_architecture))
- [91] Wikipedia “Queue (abstract data type)”. [En línea]. Disponible en: [https://en.wikipedia.org/wiki/Queue_\(abstract_data_type\)](https://en.wikipedia.org/wiki/Queue_(abstract_data_type))
- [92] Geeksforgeeks “BlockingQueue Interface in Java”. [En línea]. Disponible en: <https://www.geeksforgeeks.org/blockingqueue-interface-in-java/>
- [93] Wikipedia “Producer-consumer problem”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem
- [94] jetbrains.com “Test Driven Development: Creating Your First Test”. [En línea]. Disponible en: <https://www.jetbrains.com/help/idea/tdd-with-intellij-idea.html>
- [95] Apache Software Fundation “HttpCore API”. [En línea]. Disponible en: <https://hc.apache.org/httpcomponents-core-ga/>
- [96] JSON Simple “JSON Simple”. [En línea]. Disponible en: <https://code.google.com/archive/p/json-simple/>
- [97] w3schools.com “JSON - Introduction”. [En línea]. Disponible en: https://www.w3schools.com/js/js_json_intro.asp
- [98] learn.shayhowe.com “Responsive Web Design”. [En línea]. Disponible en: <https://learn.shayhowe.com/advanced-html-css/responsive-web-design/>
- [99] socket.io “Socket.IO enables real-time, bidirectional and event-based communication.”. [En línea]. Disponible en: socket.io
- [100] Wikipedia “Mockup”. [En línea]. Disponible en: <https://en.wikipedia.org/wiki/Mockup>
- [101] github.com/gk4m/ “vue-spotify”. [En línea]. Disponible en: <https://github.com/gk4m/vue-spotify>
- [102] programcreek.com “Java Code Examples for gnu.io.SerialPort”. [En línea]. Disponible en: <https://www.programcreek.com/java-api-examples/?api=gnu.io.SerialPort>

- [103] baeldung.com “Java String.split()”. [En línea]. Disponible en: <https://www.baeldung.com/string/split>
- [104] demeranville.com “Battle of the tokenizers – delimited text parser performance”. [En línea]. Disponible en: <http://demeranville.com/battle-of-the-tokenizers-delimited-text-parser-performance/>
- [105] Google “Maps Javascript API”. [En línea]. Disponible en: <https://developers.google.com/maps/documentation/javascript/tutorial>
- [106] Mapbox “Maps and location for developers”. [En línea]. Disponible en: <https://docs.mapbox.com/api/>
- [107] support.google.com “Is it illegal to download google maps to offline gps navigators like navicomputer?”. [En línea]. Disponible en: <https://support.google.com/maps/thread/AAAAQuUrST8BvgSPni-XrM/?hl=sl>
- [108] Youtube.com WeArGenius “16x2 LCD using I2C | MCP23017| JAVA | Pi4J | Raspberry Pi #23”. [En línea]. Disponible en: <https://www.youtube.com/watch?v=0aK9us8-2iU>
- [109] Wikipedia “Deep Learning”. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Deep_learning
- [110] pytorch.org “An open source machine learning framework that accelerates the path from research prototyping to production deployment”. [En línea]. Disponible en: <https://pytorch.org/>
- [111] tensorflow.org “An end-to-end open source machine learning platform”. [En línea]. Disponible en: <https://www.tensorflow.org/>