UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Título del Proyecto:

Arquitectura Software para un Hyperloop

Autor:

Portilla Edo, Xavier

Director:

Pérez, Rubén

TESINA PARA LA OBTENCIÓN DEL TÍTULO DE:

Diploma de Especialización en Computación Móvil y Ubicua

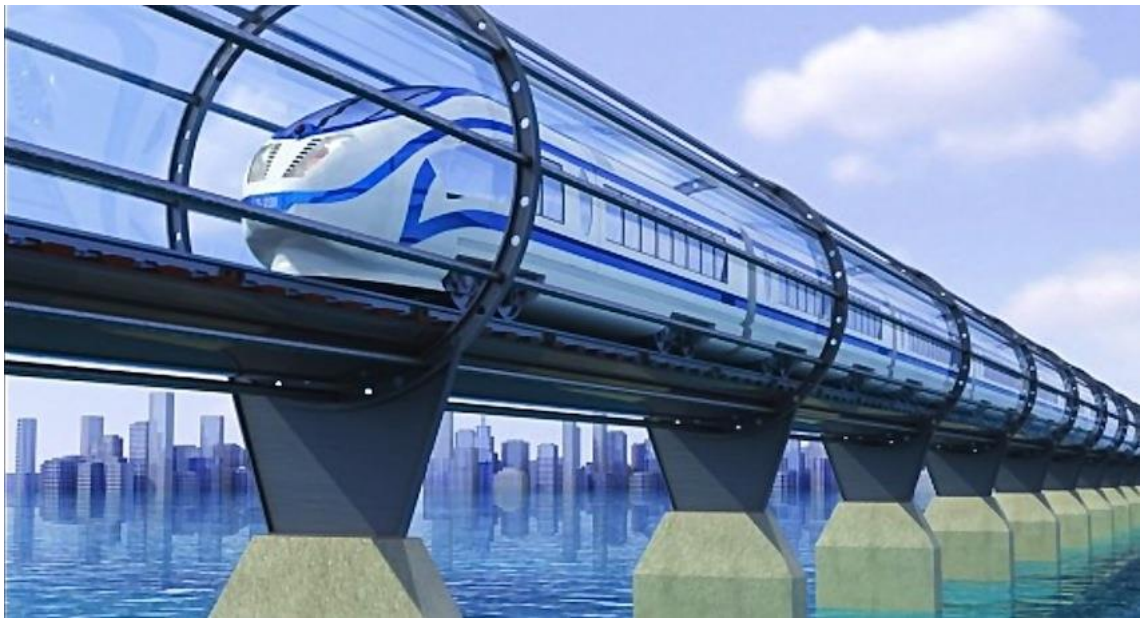
Septiembre de 2017

Contenido

Introducción

HYPERLOOP. El nombre del nuevo método de transporte que desarrolló Elon Musk (cofundador de PayPal y fundador de Tesla y Space X) con el que espera revolucionar la industria.

Se trata de un sistema de tubos de aluminio que conectarán una ciudad con otra utilizando paneles solares como fuente de energía. Estos tubos estarían montados en columnas de entre 40 y 90 metros de altura y permitirían viajar a una velocidad de 1.200 kilómetros por hora. Estos contenedores podrían transportar tanto personas como vehículos.



El proyecto surgió como una respuesta a los planes de crear un sistema de trenes de alta velocidad en California que costaría US$ 70 mil millones. Con Hyperloop se solucionan muchos de los problemas de esta iniciativa: el nuevo método costaría entre US$ 6 y 10 mil millones y sería cuatro veces más rápido que los trenes

El Hyperloop fue diseñado para unir ciudades que se encuentren a 1.600 kilómetros de distancia y que cuenten con tráfico muy pesado entre ambas.

Elon se arrepiente de haberse expresado públicamente sobre el proyecto ya que todavía se encarga de Tesla y Space X. El objetivo es que un tercero comience a trabajar con el proyecto. Si nadie trabaja en el Hyperloop las empresas privadas como Hyperloop One o las universidades como la Universidad Politécnica de Valencia, Purdue University, MIT o Stanford University entrarían en el desarrollo del proyecto.

Objetivos

El objetivo es el diseño completo de una arquitectura software completa, ambiciosa y futurista. Esta arquitectura engloba desde el diseño a bajo nivel como es la electrónica del prototipo, pasando por un middleware capaz de recolectar cantidades desmesuradas de información, transformarla, almacenarla y reenviarla a la capa arquitectónica más alta de la interfaz gráfica de usuario.

Junto a esta arquitectura se diseñan procesos externos que garantizan el correcto funcionamiento del prototipo, añadiendo así procesos de DevOps y explotación masiva de logs.

Motivación

Al poco tiempo que Elon Musk presentara el proyecto del Hyperloop, estudiantes de la Universidad Politécnica de Valencia entran en contacto gracias a la asociación Makers UPV.

Poco tiempo después SpaceX, empresa de Elon decide hacer una competición anual, sólo para estudiantes, para que diseñasen sus propios prototipos de Hyperloop para encontrar cual era el más rápido dentro del tubo.

HYPERLOOP UPV
Attract the future.

A los pocos días de haber recibido la noticia de la competición de SpaceX, los alumnos de Makers UPV que conocen el proyecto se juntan para formar un equipo, Hyperloop UPV. Hyperloop UPV es un equipo de 30 ingenieros de diferentes disciplinas con un único objetivo, crear un prototipo altamente diferenciador del resto de las universidades y ganar la competición.

En este equipo se me asigna el rol de Arquitecto Software en el subequipo de aviónica debido a la experiencia adquirida en el mundo de la consultoría de negocio.

Posteriormente y como consecuencia de que un estudiante y un profesor de la Universidad Politécnica de Valencia estaban estudiando y trabajando en Purdue University de Indiana, se llega a un acuerdo con su equipo de Hyperloop para poder mejorar el diseño del prototipo actual y poder ganar la *Hyperloop Pod Competition.*

Tecnologías utilizadas. Visión global

Este capítulo tiene como finalidad hacer una breve introducción a la arquitectura y explicar componente a componente que tecnología se ha utilizado para poder entender a posteriori el proyecto en su plenitud.

Comenzando por el nivel más bajo, la electrónica, se han utilizado tres microcontroladores Teensy los cuales han sido programados con el IDE de Arduino para cargar el código en las placas y Microsoft Visual Studio para estructurar y programar gran parte del código. Para las comunicaciones entre la capa de electrónica y el middleware se ha usado el protocolo UDP. A su vez, la interfaz gráfica de usuario y la capa de electrónica se comunican también vía UDP para poder enviar comandos directamente desde la web al prototipo.

Para la capa intermedia o middleware se han empleado herramientas que permitían hacer un streaming en tiempo real con cantidades semejantes a las del Big Data. Para ello se ha utilizado Mule como Enterprise Service Bus y Apache Kafka para el streaming Big Data en tiempo real.

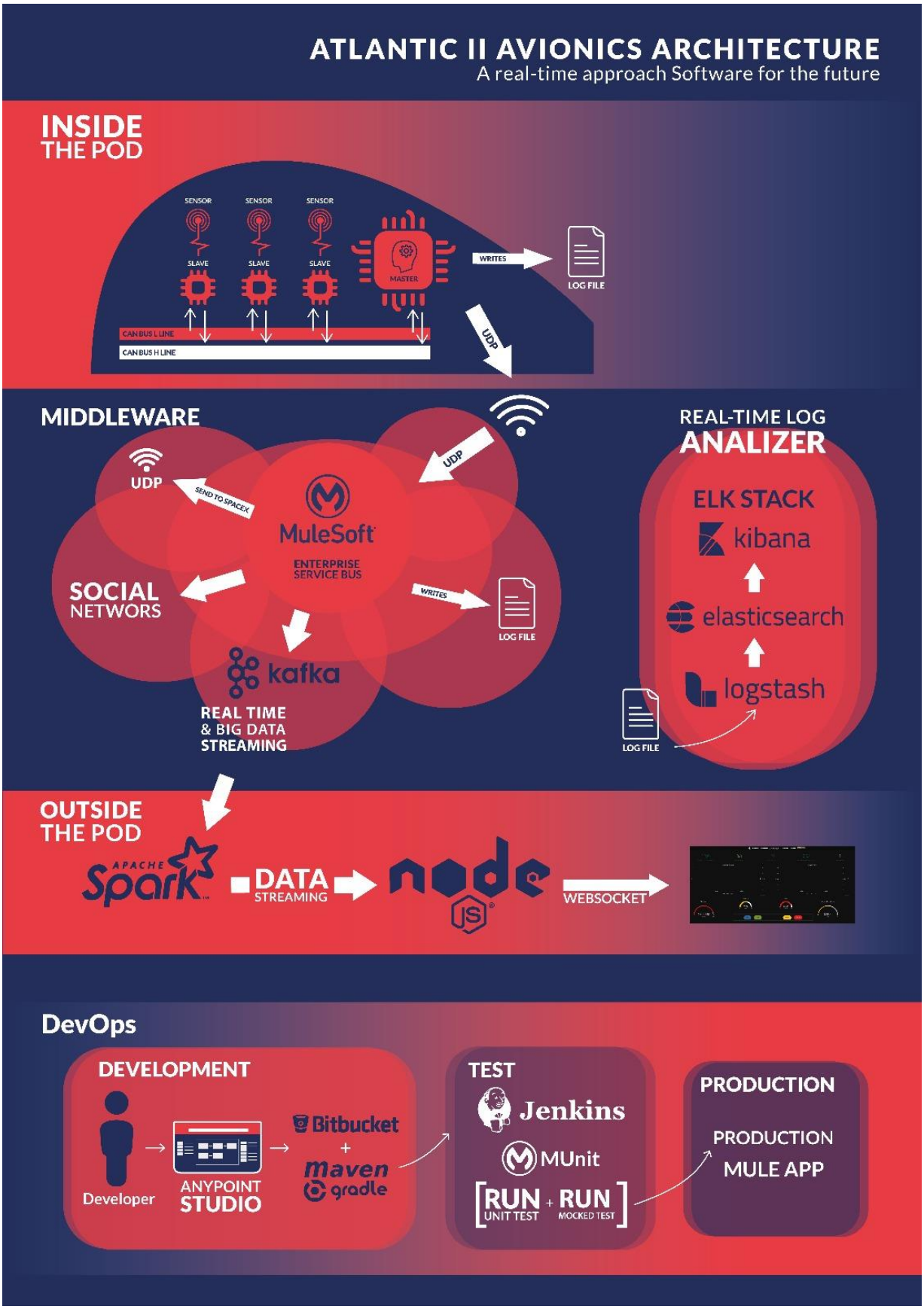
En la interfaz gráfica de usuario, para la recepción de toda la información de Kafka, se ha utilizado Apache Spark. La aplicación corre sobre NodeJS y se comunica entre el backend y frontend mediante websockets. La aplicación está hecha en su totalidad en AngularJS.

En La explotación de logs se ha utilizado el stack de ELK, el cual incluye Logstash, Elasticsearch y Kibana. En el siguiente capítulo se explica más detalladamente cada herramienta y que rol tiene en la explotación de los logs. Cabe destacar que este stack tecnológico es uno de los más importantes a nivel empresarial en este terreno.

Para DevOps se ha utilizado uno de los sistemas de control de versiones más importante dentro del mundo empresarial, Bitbucket. Como motor de integración continua se ha usado CircleCI, muy parecido a Jenkins. Finalmente, para recopilar todos los compilados se ha utilizado Artifactory.

Para esta arquitectura se han utilizado frameworks y librerías Open Source ya que era uno de los requisitos del proyecto. Aunque algunas partes de la arquitectura eran de pago, se han utilizado sus versiones gratuitas porque eran más que suficientes para las necesidades del equipo y del prototipo.

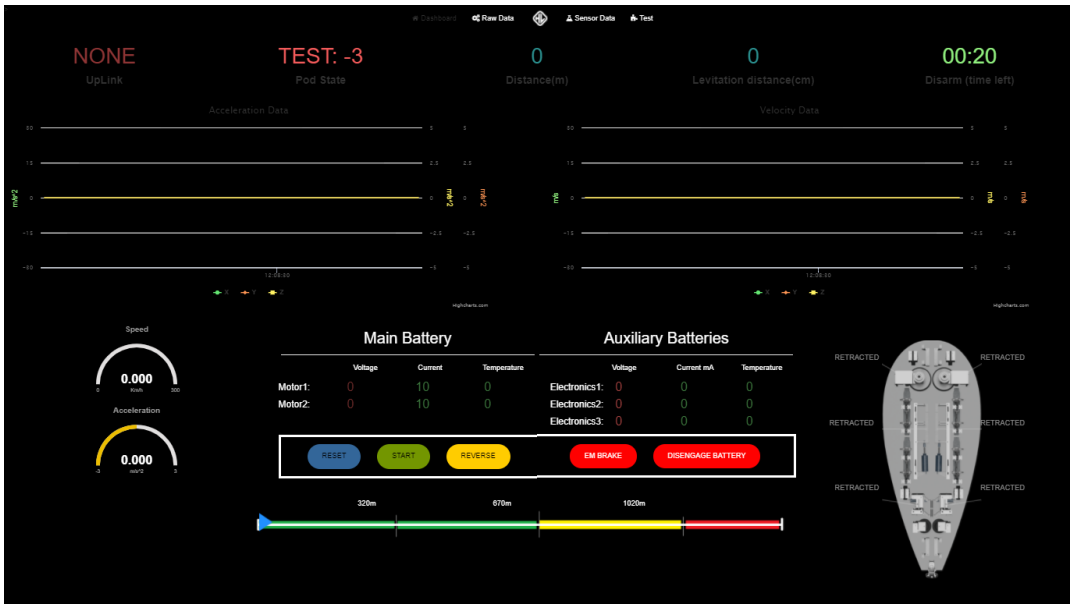
En la siguiente página se puede encontrar una visión completa de toda la arquitectura software diseñada y como se interconectan todos los componentes entre ellos.

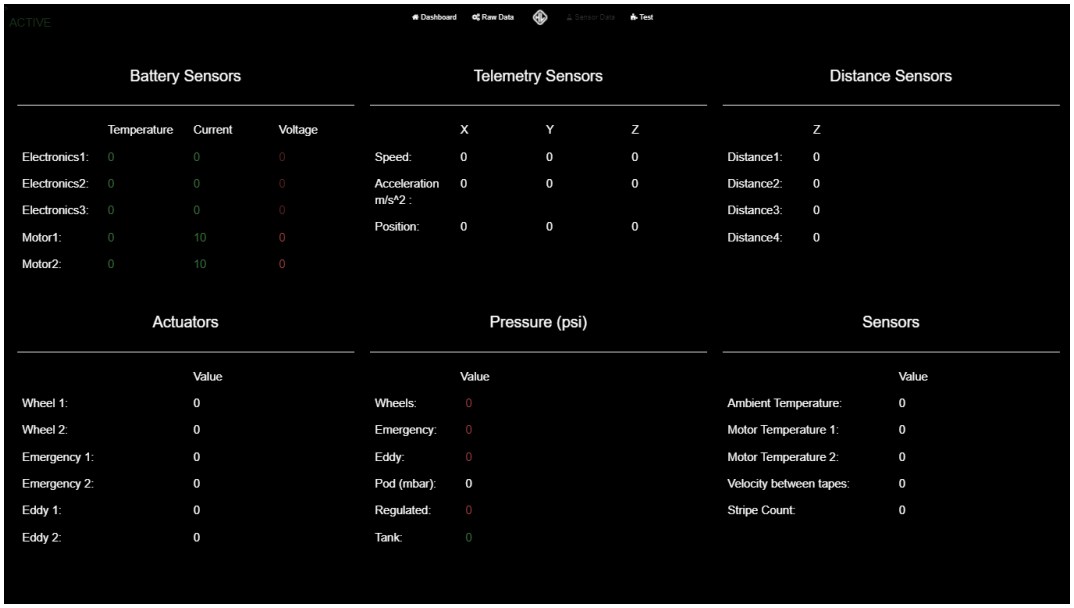

ATLANTIC II AVIONICS ARCHITECTURE
A real-time approach Software for the future
INSIDE THE POD
SENSOR
SENSOR
SENSOR
SLAVE
SLAVE
SLAVE
MASTER
WRITES
LOG FILE
UDP
CAN BUS L LINE
CAN BUS H LINE
MIDDLEWARE
UDP
SEND TO SPACEX
UDP
MuleSoft
ENTERPRISE SERVICE BUS
SOCIAL NETWORKS
WRITES
LOG FILE
kafka
REAL TIME & BIG DATA STREAMING
REAL-TIME LOG ANALIZER
ELK STACK
kibana
elasticsearch
logstash
LOG FILE
OUTSIDE THE POD
APACHE Spark
DATA STREAMING
node
WEBSOCKET
DevOps
DEVELOPMENT
Developer
ANYPOINT STUDIO
Bitbucket
+
Maven
gradle
TEST
Jenkins
MUnit
RUN UNIT TEST + RUN MOCKED TEST
PRODUCTION
PRODUCTION MULE APP

Arquitectura de la aplicación

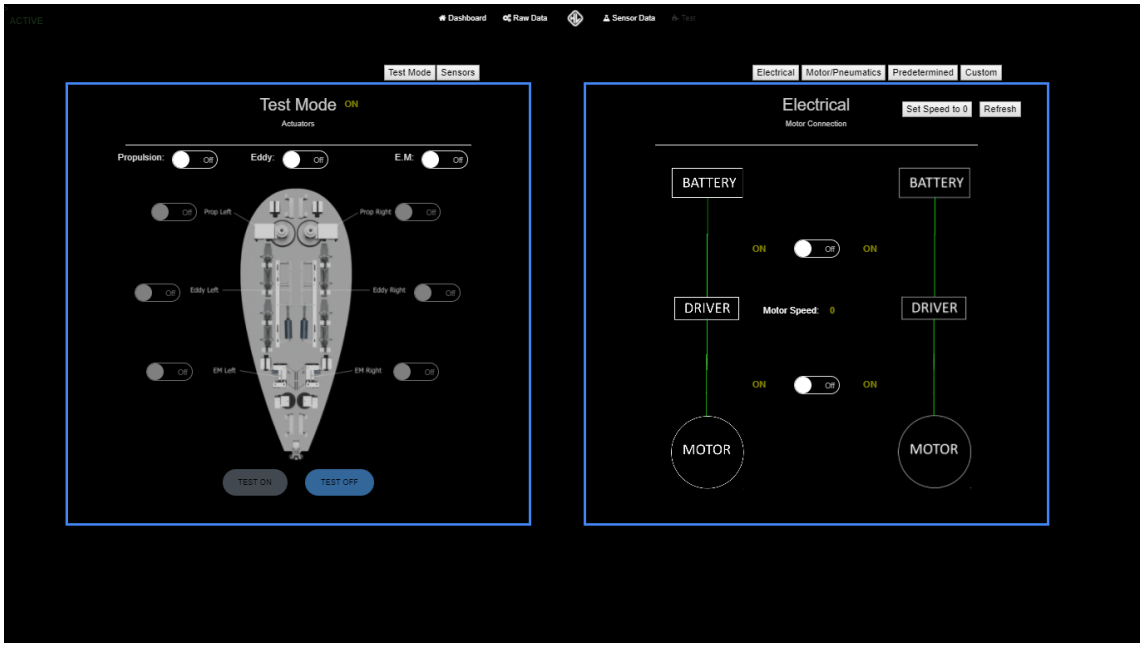
Electrónica

Uno de los requerimientos más importantes de SpaceX era la seguridad y la única manera de poder llegar a tener un prototipo seguro era sensorizar cada parte del mismo. Para ello se instalaron más de 35 sensores de todo tipo por toda la maquinaria. Había sensores de temperatura, corriente, voltaje, presión, aceleración, velocidad, altitud y lectores de una cinta 3M que estaba situada dentro del tubo y debíamos leerla para saber la posición exacta. Desde la interfaz gráfica de usuario éramos capaces de leer toda la información de los sensores en tiempo real:

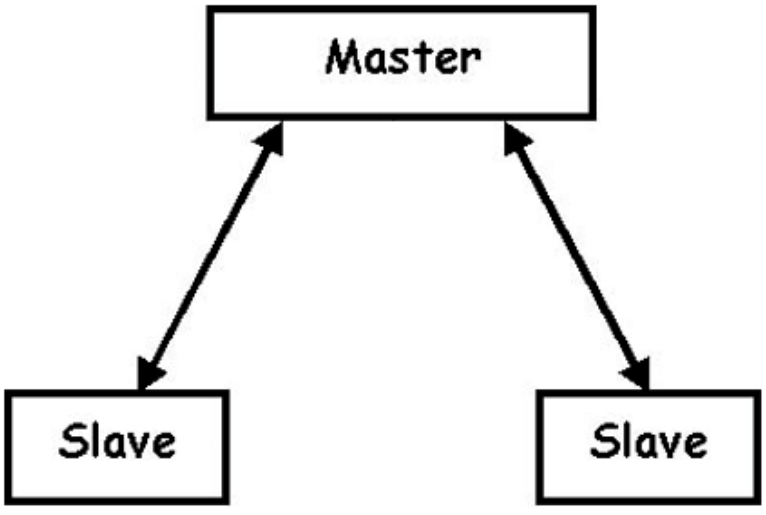




Además de los sensores teníamos ciertos actuadores que debíamos actuar dependiendo de la máquina de estados y todos los inputs de los sensores. Como actuadores teníamos los motores, los contractores que se encargaban de dejar pasar energía o no a cada motor y la neumática encargada de accionar los frenos. En la interfaz gráfica de usuario se podía actuar sobre cualquier cosa de lo mencionado anteriormente.



Como se puede deducir, leer tanta información y ejecutar tanto código por cada ciclo de reloj era demasiado para un solo microcontrolador. Por esto se tomó la decisión de realizar una arquitectura maestro/esclavo teniendo un maestro, que se encargaba de ejecutar la máquina de estados y tomar decisiones y dos esclavos cuya labor principal era la de recibir datos de los sensores y accionar los actuadores. Todos ellos estaban conectados mediante bus CAN. En los anexos se pude ver toda la lógica de la máquina de estados para así poder entender las decisiones que se tomaban en el microcontrolador máster.

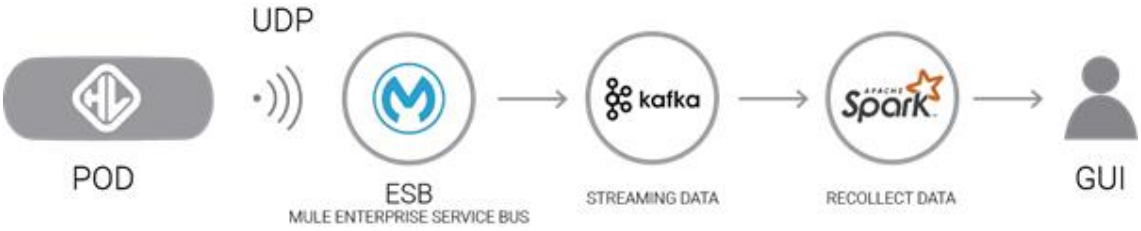


Las comunicaciones con el middleware se hacían mediante UDP gracias a que al microcontrolador master había una tarjeta Ethernet que era capaz de enviar y recibir datos. Una de las tareas más importantes a nivel arquitectónica fue la de estructurar toda la información que se capturaba en los microcontroladores y hacerla entendible por las siguientes capas de la arquitectura. Sabiendo que la interfaz gráfica de usuario estaba programada en Angular se decidió estructurar la información usando el formato JSON (acrónimo de JavaScript Object Notation)

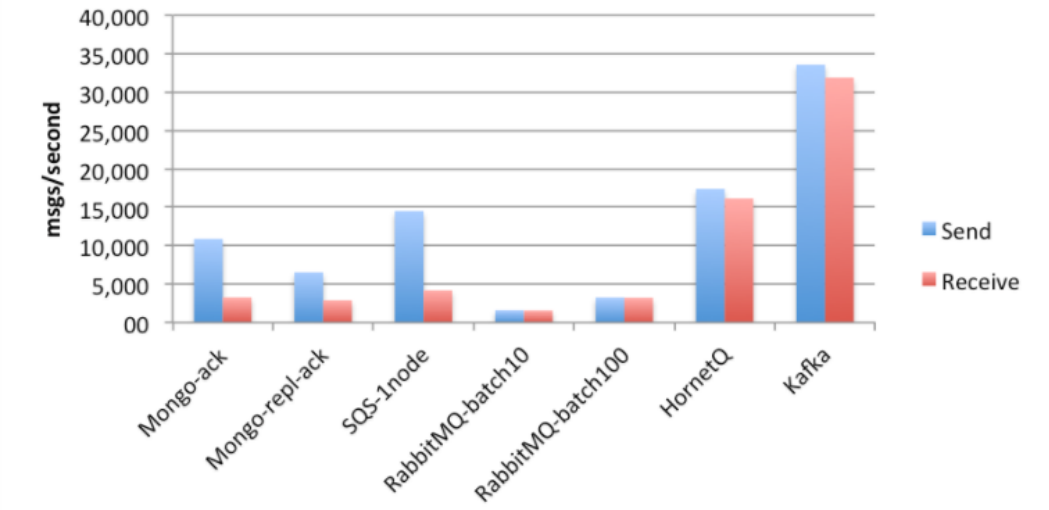
JSON es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript, aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente. En los anexos se puede encontrar un ejemplo de JSON y el JSON Schema.

Middleware

Entre la electrónica y la interfaz gráfica de usuario, habrá una capa intermedia donde se ubicarán tres componentes de software diferentes. El objetivo principal de esta capa es separar la interfaz gráfica de usuario del prototipo y recoger todos los datos para analizarlos, por ejemplo, con ELK. Los tres componentes de software se representan en la siguiente figura:



1. **Enterprise Service Bus (Mule ESB)**: este componente de software está conectado a la cápsula a través de la conexión UDP. Una de sus funciones es registrar constantemente todos los datos recibidos de la cápsula. A continuación, el ESB envía los datos a través de una gran cola de datos, Apache Kafka. Este componente se ha desarrollado con Mule Enterprise Service Bus de Mulesoft utilizando Anypoint Studio.
2. **Big data streaming (Apache Kafka)**: aquí los datos viajan a través de una cola hasta que se transmite a través de un flujo basado en suscripciones. Este componente puede conectar varios agentes a esta cola simplemente suscribiéndose a ella. Por ejemplo, cualquier aplicación o interfaz gráfica que se pueda conectar a él, puede hacer uso de la información recibida, por ejemplo, Microsoft Power BI, Apache Hadoop, Hazelcast, Apache Spark, etc.
   Se decidió utilizar Apache Kafka porque era la única cola que era capaz de transmitir cantidades enormes de datos y no bloquearse, cosa que sí pasaba con ActiveMQ o RabbitMQ. En la siguiente imagen se puede observar una comparación entre motores de cola donde se aprecia la potencia real de Apache Kafka:



3. **Recolección de datos (Apache Spark)**: todos los datos deben ser recolectados de esta cola basada en suscriptores. Este servicio de recolección es inteligente y sabrá cuándo y cuál fue el último paquete de datos recibido y podrá reiniciar la recolección desde ese punto a través de una reconexión. La API de Apache Spark se utiliza para suscribirse a la cola de Apache Kafka descrita en el punto 2. En definitiva, Apache Spark es un recolector inteligente que se puede enganchar a diferentes tipos de streaming como se puede observar en la siguiente figura:

Interfaz gráfica de usuario

Otro de los requisitos que exigía SpaceX era tener una interfaz gráfica de usuario capaz de recibir toda la información en tiempo real y que ésta se visualice de manera sencilla y clara para poder entender que está ocurriendo dentro del prototipo.

Por ello teniendo el inmenso stack tecnológico que hay hoy en día en internet, se tuvo que hacer una valoración y desde el punto de vista arquitectónica, decidir cuál era el más adecuado para las especificaciones del proyecto. Por ese motivo se decidió montar una aplicación NodeJS por la enorme cantidad de paquetes npm desarrollados por la comunidad y que han servido de gran utilidad.

Por lo tanto, se decide desarrollar una aplicación completamente en AngularJS con el backend en NodeJS utilizando numerosos paquetes npm. La comunicación entre el backend y el frontend se hace mediante websocket para que se actualice la información en tiempo real sin necesidad de actualizar la página.

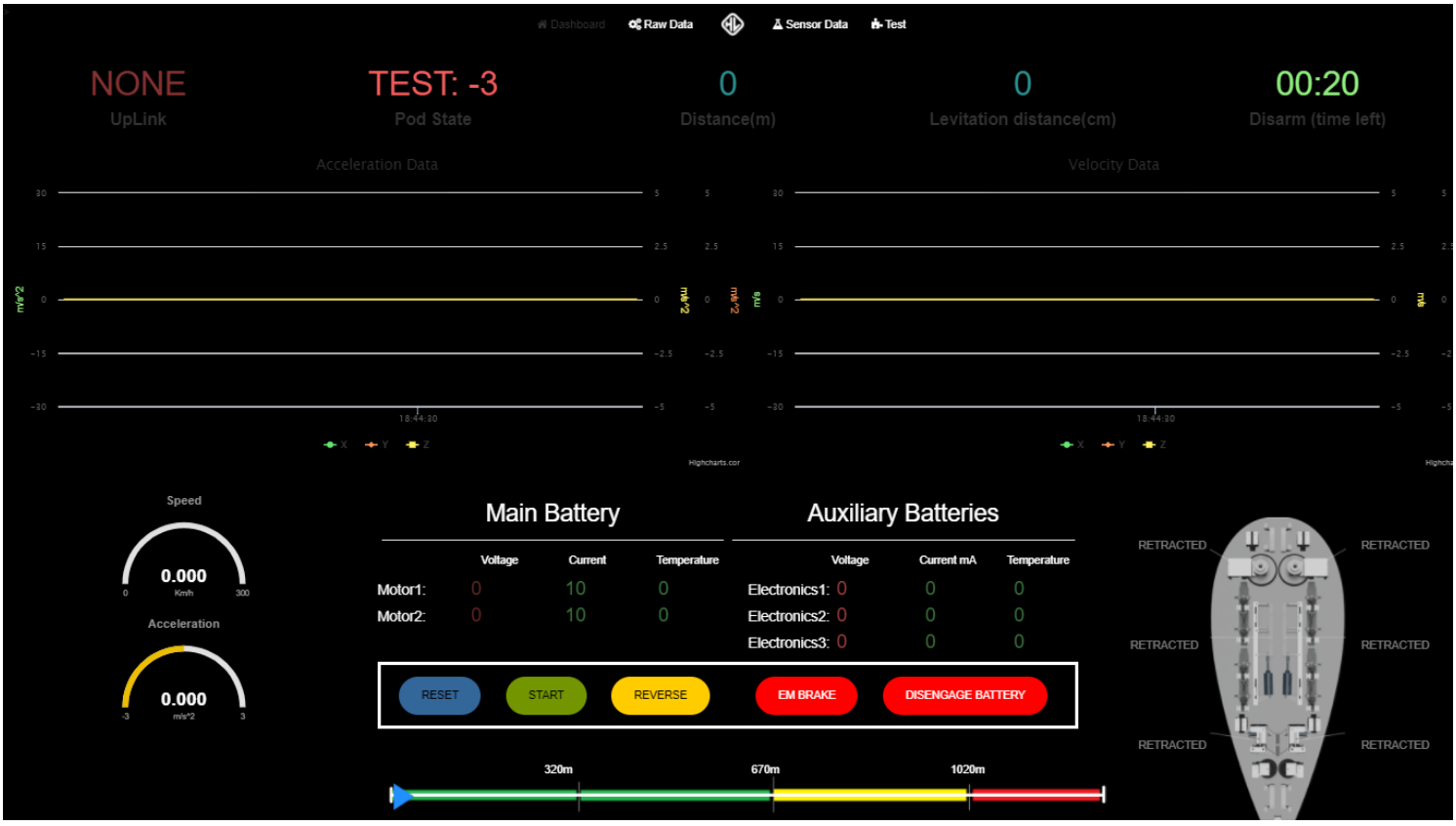
La interfaz gráfica se compone de 4 pantallas principales:

1. DASHBOARD

Antes de explicar esta parte de la aplicación hay que tener en cuenta que para que la interfaz gráfica de usuario reciba datos del middleware se ha utilizado el paquete npm Kafka-node que da la posibilidad de suscribirse a colas Kafka implementando Spark para poder recibir información. También se ha utilizado el paquete npm socket.io para poder crear el canal de comunicación vía websocket entre el frontend y el backend.

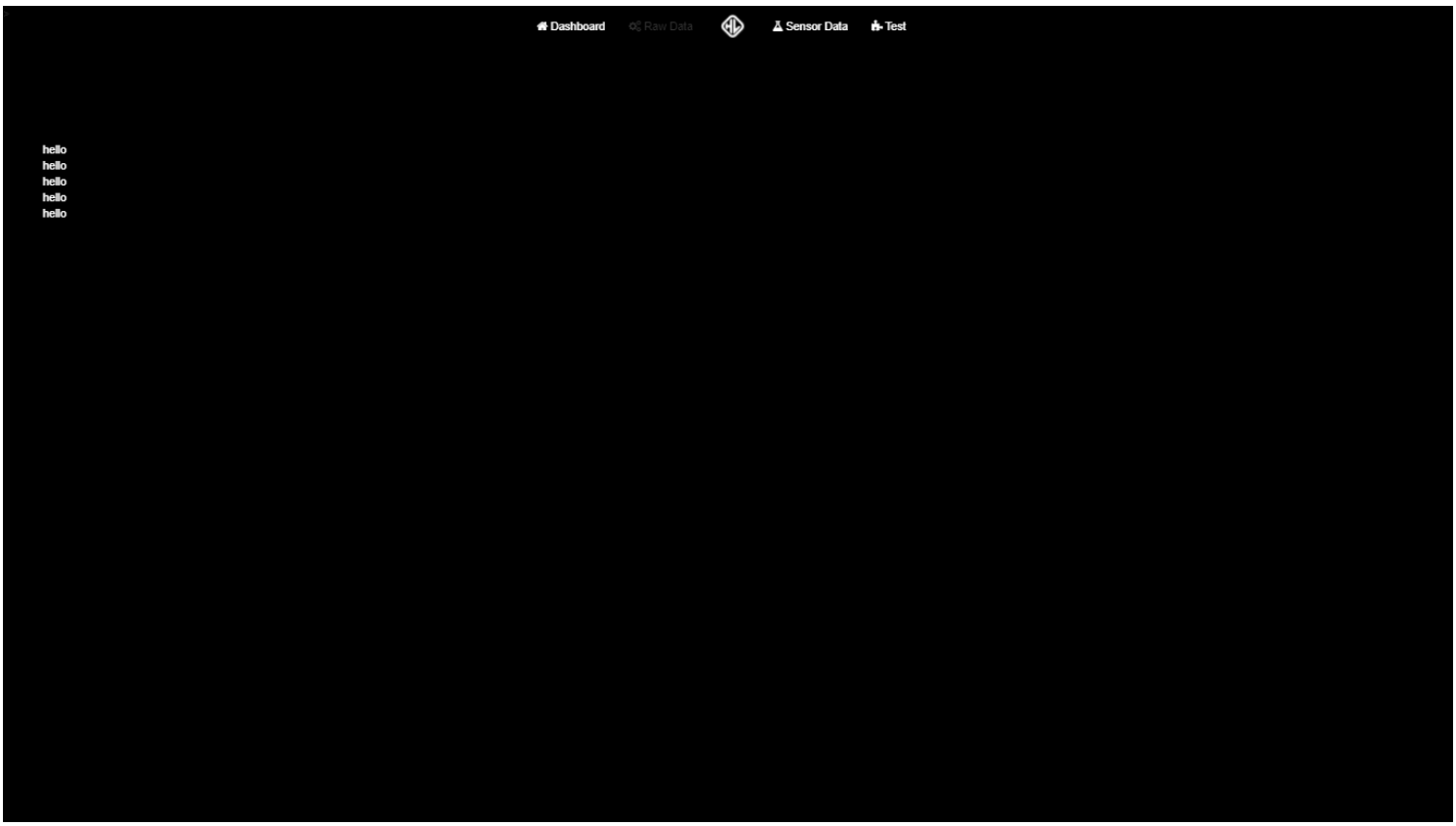
Una vez explicado esto y, sabiendo cómo se comporta por detrás la aplicación, se puede continuar con esta parte. En el Dashboard se visualiza toda la información crítica del prototipo como pueden ser las temperaturas, corrientes y voltajes de las baterías, el estado de la máquina de estados, cuanto tiempo queda para activar el freno, distancia recorrida, etc.

A su vez se pueden encontrar dos gráficas que visualizan los valores de la velocidad y la aceleración respecto del tiempo. Para estas gráficas se ha usado el paquete npm dgram junto con los componentes angular.



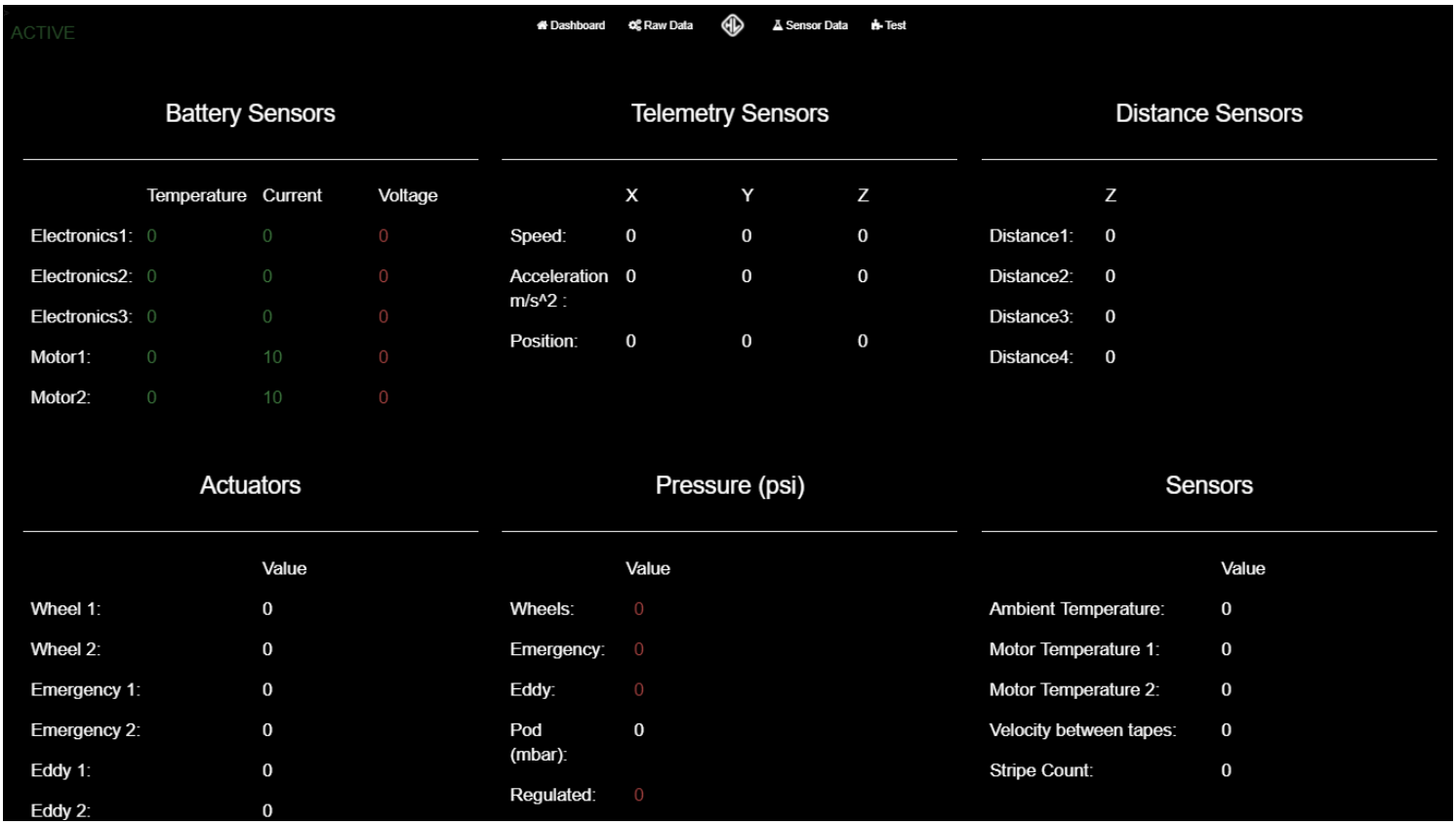
2. RAW DATA

Esta pantalla es muy simple, la única función que tiene es la de visualizar el campo raw_data que se recibe en el JSON del middleware. Este campo contiene toda la traza interna de los microcontroladores para saber que se está ejecutando en ellos y poder detectar posibles errores sin tener que estar conectados vía USB a éstos.



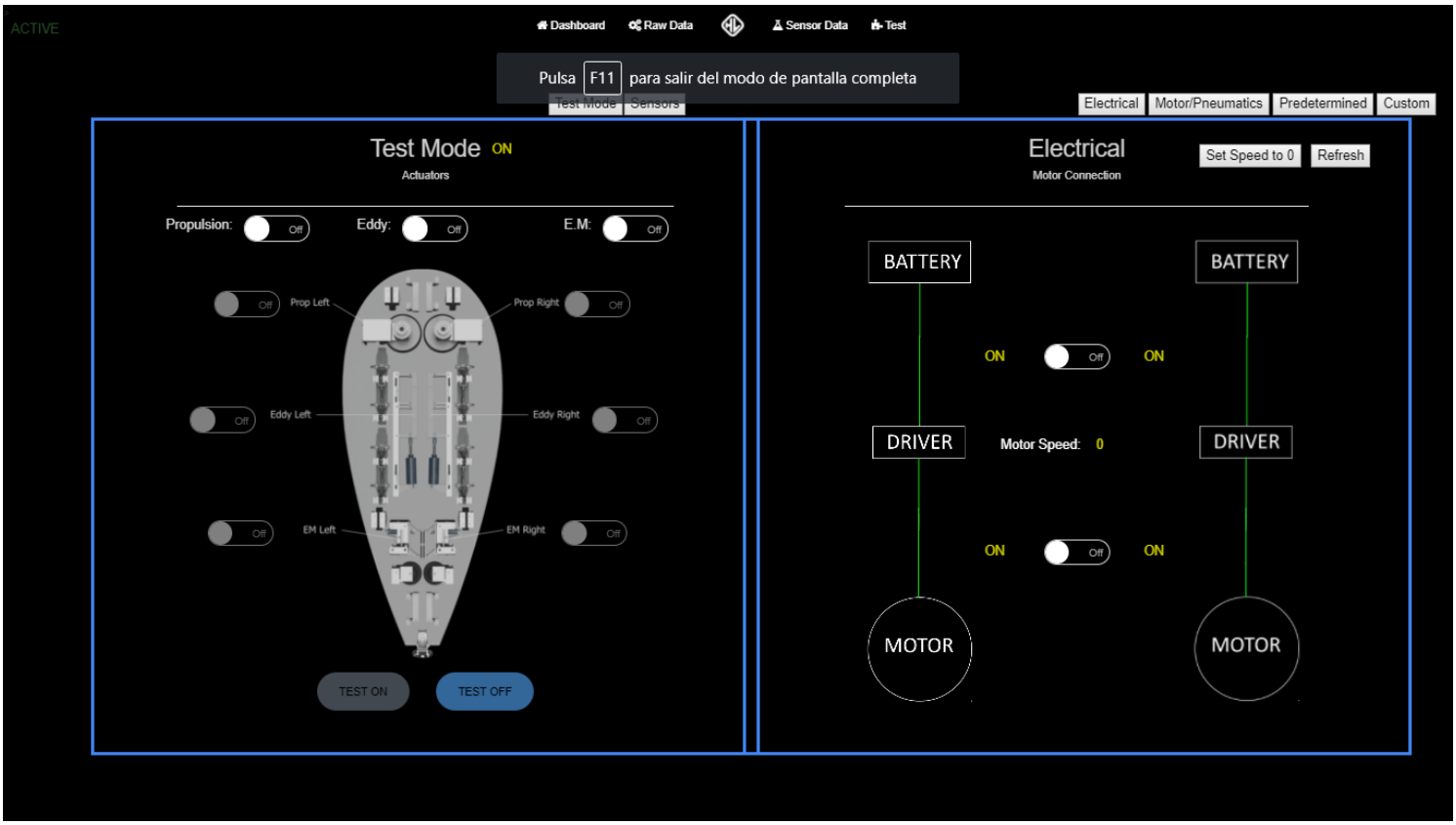
3. SENSOR DATA

En el Dashboard es imposible visualizar la información de los 35 sensores. Es por esta razón por la que se crea una pantalla específica para visualizar toda la información de los sensores agrupando la información por el origen de donde proviene: distancia, baterías, presión, etc.:



4. TEST

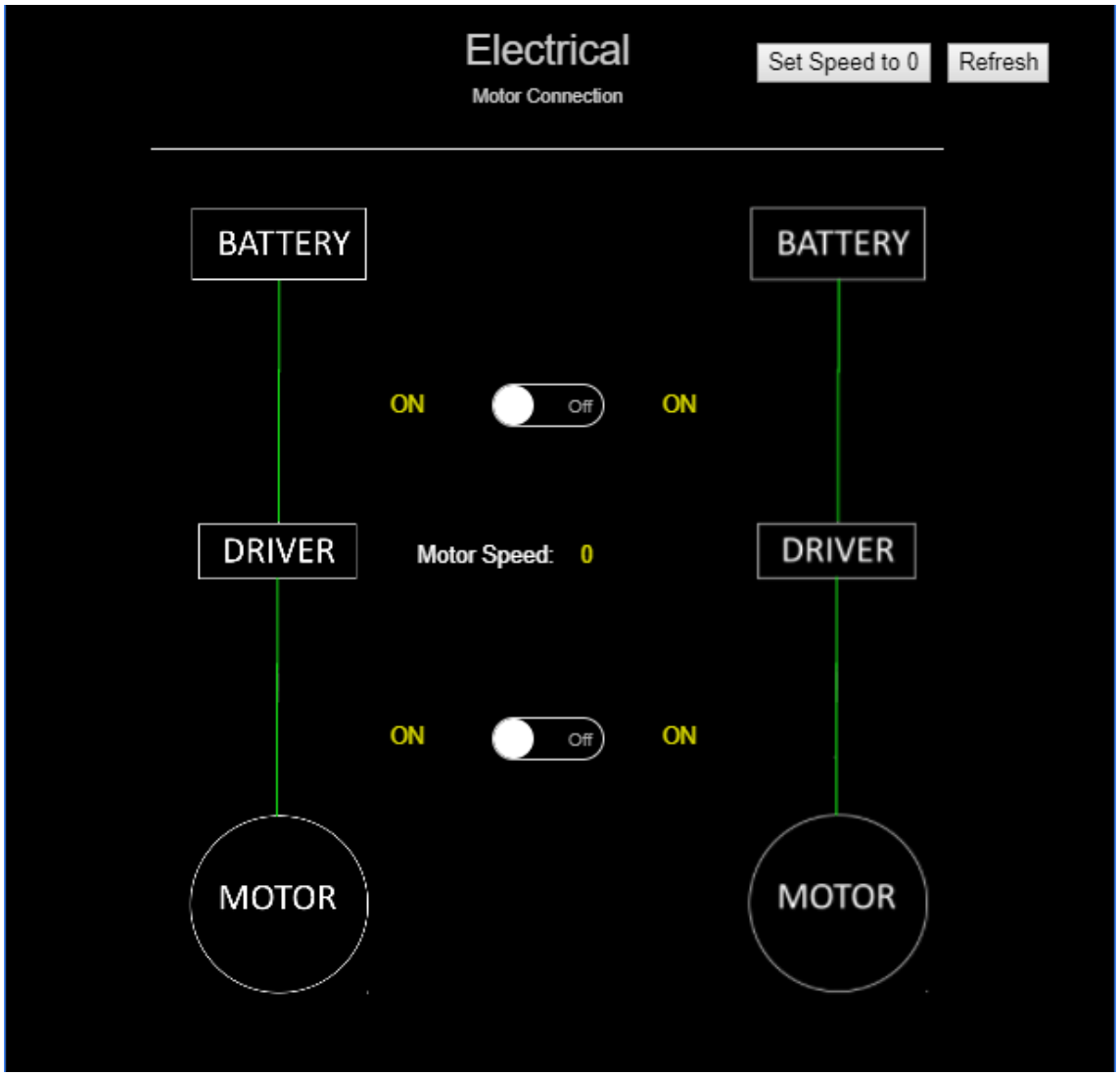
Esta parte es, junto con el Dashboard, la más importante de toda la aplicación ya que es la que permite manejar de manera remota todas las funcionalidades del prototipo, algo necesario para poder pasar todos los test a que obliga SpaceX antes de entrar al Hyperloop.



Se divide en 5 subpantallas:

Electrical

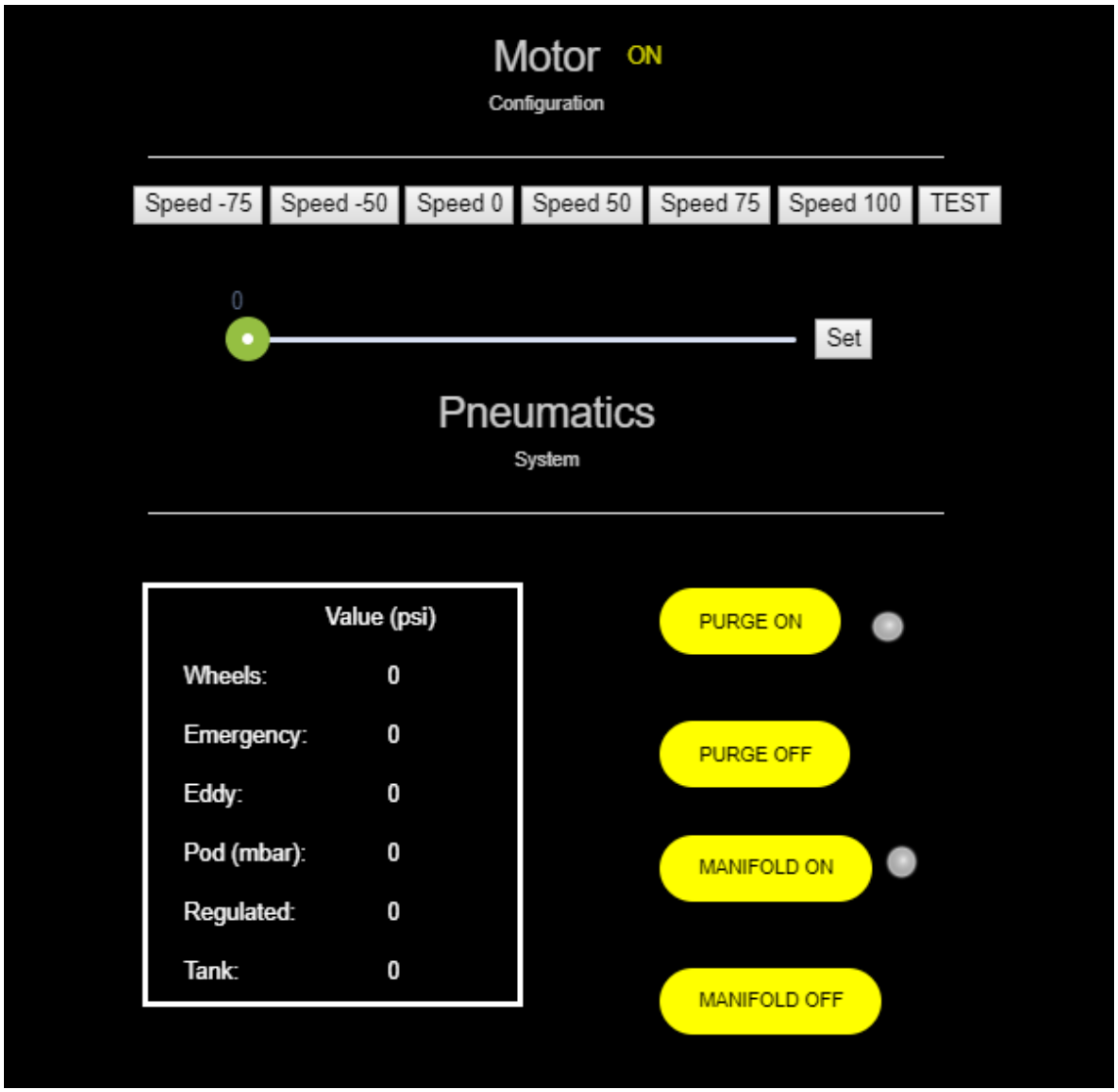
Esta parte se utiliza para poder abrir y cerrar los contractores. Permite pasar la energía de las baterías a los motores. A su vez también permite activar los motores. También se visualiza la velocidad del motor:



Motor/pneumatics

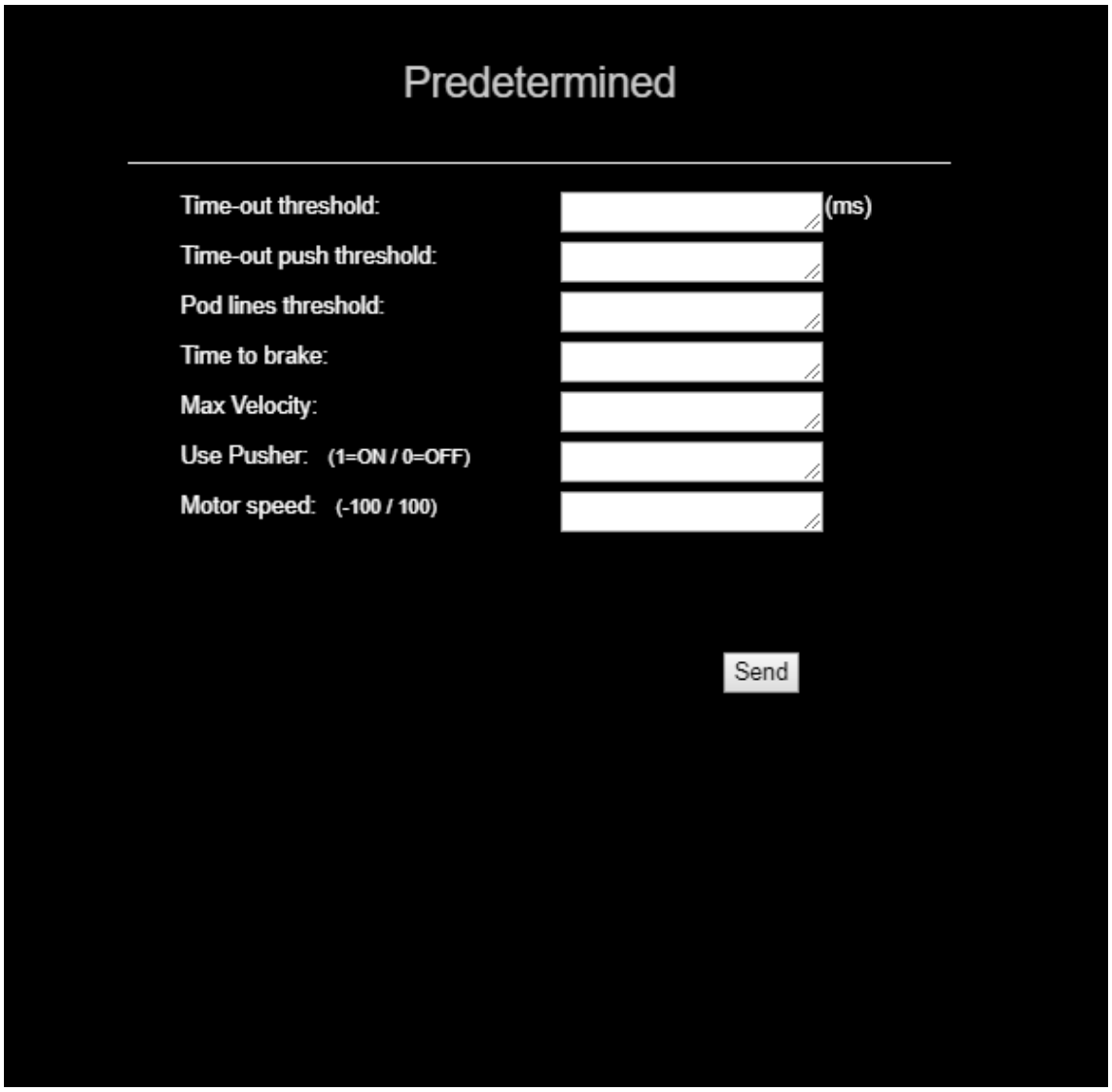
En esta parte se puede encontrar todo lo relacionado con la neumática del prototipo y también se puede especificar la velocidad del motor mediante botones o mediante una slider.

En la parte de la neumática se pueden encontrar todos los actuadores de este subsistema y la información de los sensores colocados en las válvulas del prototipo.



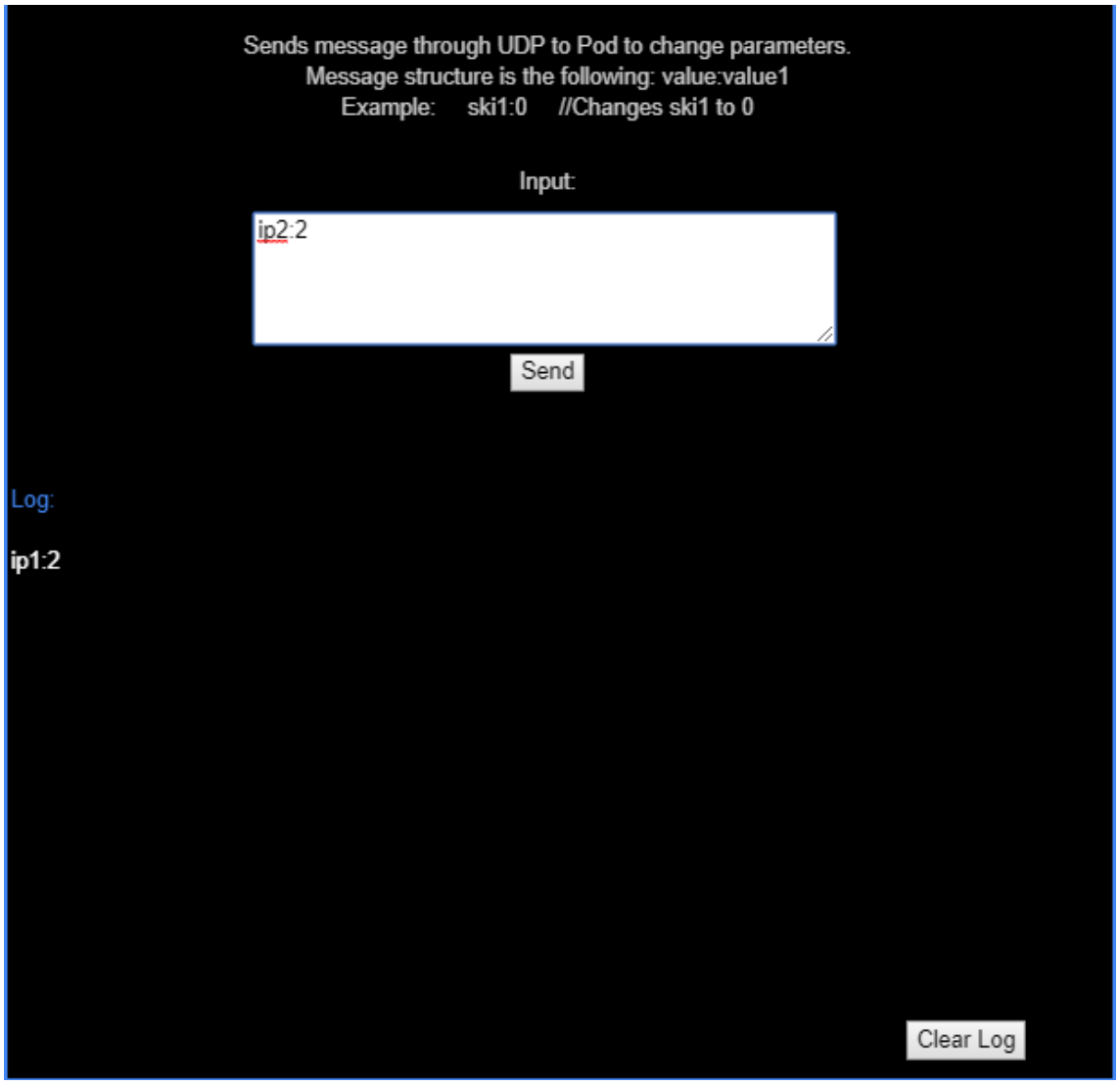
Predetermined

Esta pantalla permite enviar información a los microcontroladores de manera directa usando UDP. La información que se puede modificar desde esta pantalla es toda aquella que es bastante susceptible de ser cambiada, como por ejemplo cualquier threshold. Se hizo por la necesidad de no estar flasheando el código de los microcontroladores cada vez que se quiera cambiar el valor de una constante. Para el envío UDP se ha utilizado el paquete npm UDP4



Custom

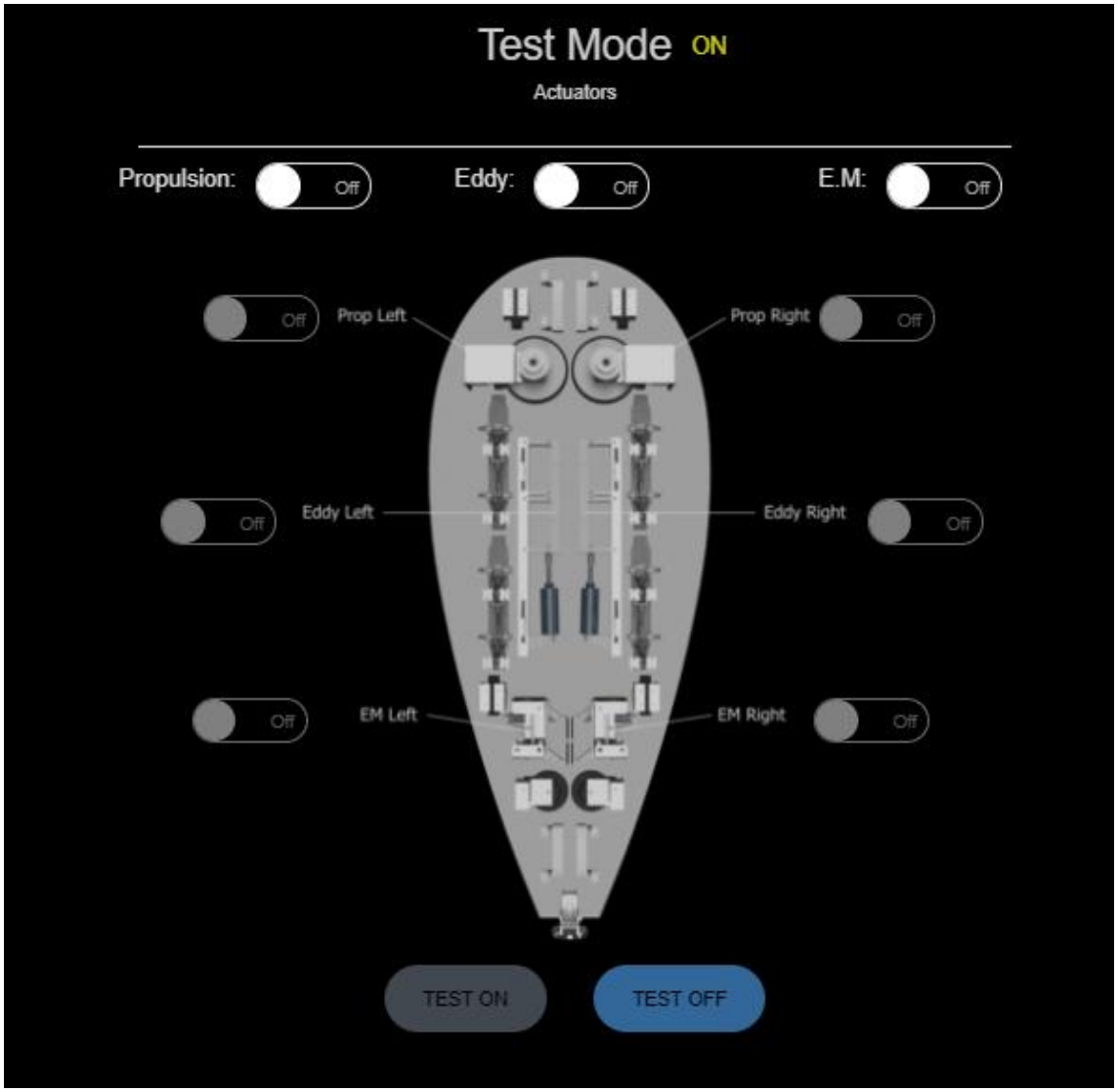
Esta pantalla a diferencia de la anterior, te permite enviar cualquier tipo de información al microcontrolador. La pantalla anterior cuando se rellenan los campos y se le da a enviar, manda la información en un cierto formato, en esta pantalla se debe escribir en ese formato antes de enviarlo al microcontrolador.



Test Mode

Esta parte de la aplicación permite saber si está activado el modo test y si no lo está, activarlo.

También permite activar los frenos magnéticos, los frenos de fricción y la propulsión del prototipo:

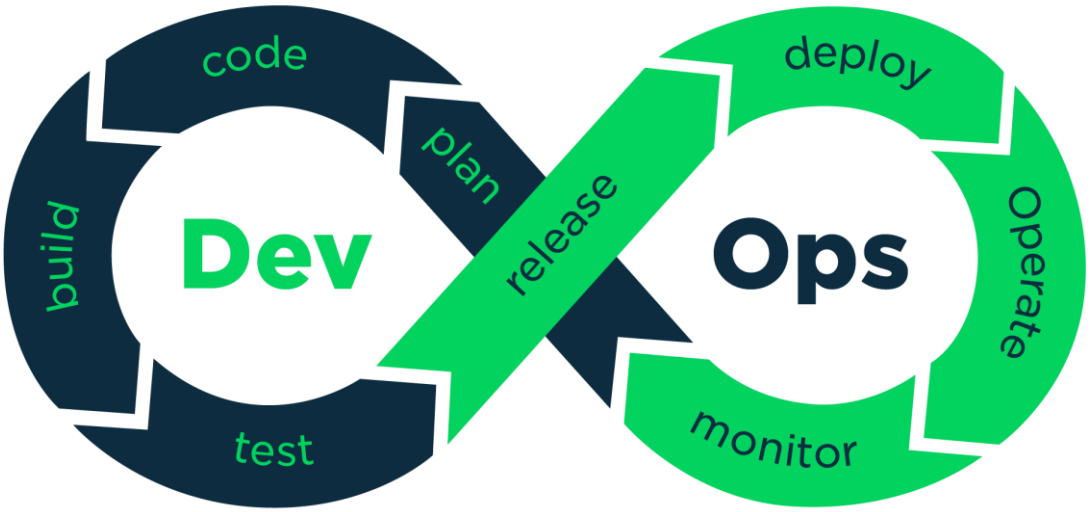


Sensors

Esta pantalla visualiza información de ciertos sensores que se encuentran en el Dashboard y en la pantalla de Sensor Data y no en la de test. Para no ir cambiando de pestaña, se creó esta subpantalla, así se tiene en una sola pantalla toda la información necesaria para visualizarla en los test:

DevOps

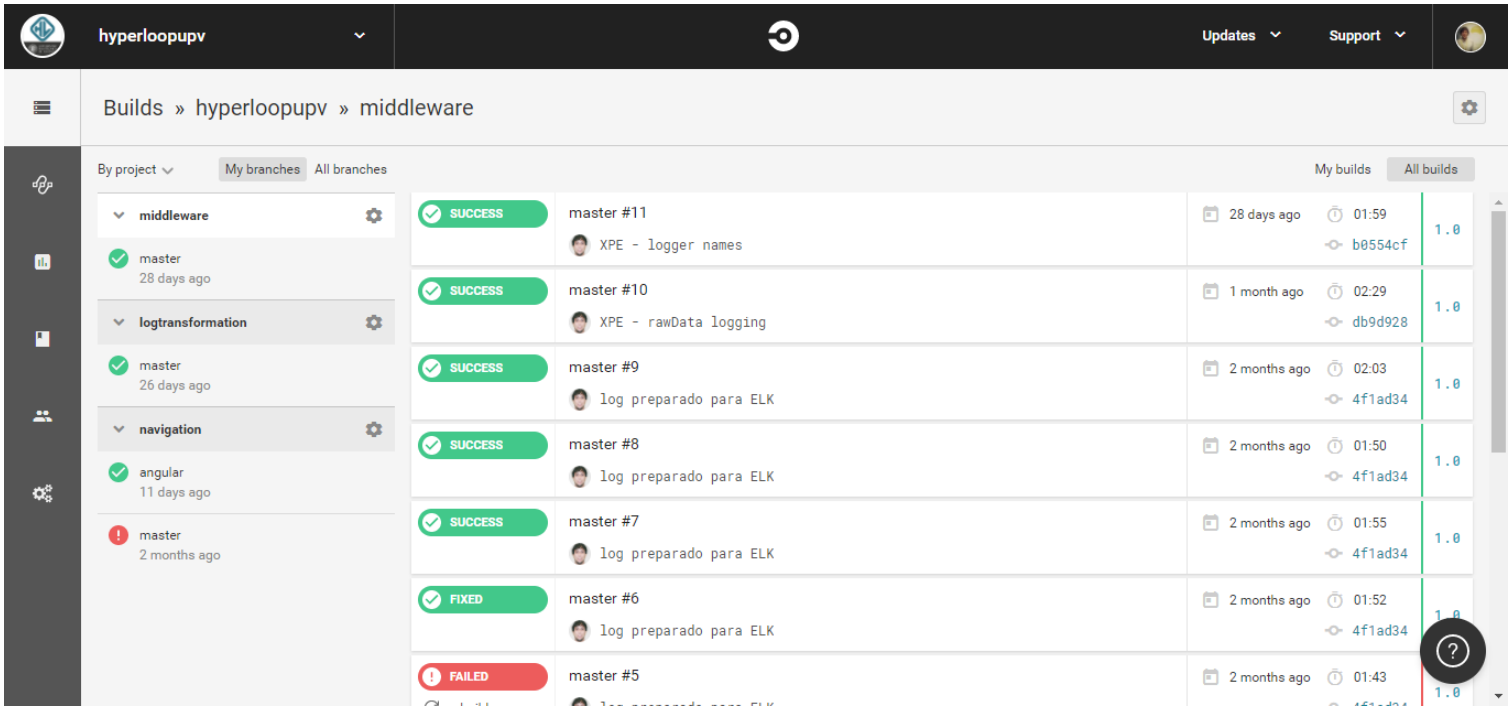
Para garantizar la integridad de todo el software del Hyperloop se decidió introducir procesos de DevOps que es una metodología de desarrollo software basada en la integración entre desarrolladores y administradores de sistemas, que permite que los desarrolladores puedan enfocarse sólo en desarrollar y puedan desplegar su código en segundos. En la siguiente figura se muestra todos los procesos que engloba DevOps:



DevOps es especialmente útil en el nuevo entorno de la transformación digital y el desarrollo de productos digitales, para los que el usuario final y/o el cliente interno de negocio demanda TTM (time-to-market), más flexibilidad, más calidad, menos coste y una altísima frecuencia de releases.

Dentro de Hyperloop se ha utilizado DevOps especialmente para la capa del intermedia o middleware y para la interfaz gráfica de usuario. Cuando se hace un *push* a Bitbucket, nuestra herramienta de control de versiones, CircleCI detectaba que se había realizado un cambio y rápidamente montaba un contenedor Docker en el cloud con lo necesario para poder compilar el código. Cuando se compilaba de manera correcta se ejecutaban todos los test unitarios. Para los test hemos utilizado JUnit para el middleware y Java y Selenium para la interfaz gráfica de usuario. Una vez todos los test estaban pasados, CircleCI se encargaba de poner los archivos compilados en Artifactory, herramienta de gestión de compilados. Si algo de todo este proceso fallaba, se recibían alertas por varios canales de comunicación. Uno de estos canales era vía Mail y otro era vía Slack. En Slack teníamos monitorizado todo lo que ocurría una vez se hace *push* a Bitbucket y se podía tener acceso a cualquier información desde cualquier parte con tu Smartphone.

En la siguiente imagen se puede ver una captura de pantalla del estado del middleware en CircleCI:



Explotación de logs

Para la explotación de logs producidos por el middleware se utilizó el stack ELK para poder hacer un intenso post procesado de toda la información.

El stack ELK es un paquete de tres herramientas Open Source de la empresa Elastic. Las herramientas son Elasticsearch, Logstash y Kibana. Estas tres herramientas son proyectos independientes y pueden ser usadas por separado, pero juntas forman un gran equipo. Entre ellas se van a encargar de leer y almacenar toda la información que necesitamos para posteriormente consultarla y monitorizarla:

1. **Logstash**: herramienta para la administración de logs. Se encarga de recolectar, parsear y filtrar los logs para posteriormente darles alguna salida como, almacenarlos en MongoDB, enviarlos por correo electrónico o guardarlos en Elasticsearch. Estos logs le pueden llegar a Losgstash desde el mismo servidor o desde un servidor externo, por lo que se podría tener un servidor exclusivo para el stack ELK. La aplicación se encuentra basada en jRuby y requiere de Java Virtual Machine para correr.
2. **Elasticsearch**: un servidor de búsqueda basado en Lucene. Provee un motor de búsqueda de texto completo (full-text), a través de una interfaz web RESTful. Básicamente nos ofrece un servicio de búsquedas a través de una API RESTful. Mediante peticiones HTTP podemos almacenar información de forma estructurada en Elasticsearch para que éste la indexe, y posteriormente poder realizar búsquedas sobre ella.
3. **Kibana**: herramienta analítica open Source (licencia Apache) que va a permitir interactuar con la información almacenada (por Logstash) en Elasticsearch y monitorizarla:

Conclusiones

Primero, hay que destacar que la aplicación responde a las expectativas y requerimientos recogidos en la especificación de requisitos. El trabajo realizado ofrece un alto nivel de cumplimiento de objetivo y es un punto de partida a partir del cual se pueden hacer mejoras.

Desde el punto de vista personal, decir que ha resultado una experiencia muy interesante haber llevado a cabo este proyecto, ya que para terminarlo ha sido necesario aplicar gran parte de los conocimientos adquiridos durante los estudios y, a su vez, es muy gratificante ver la utilidad de los mismos a la hora de desarrollar un proyecto real y completo, empezando por la toma de requisitos, planificación y estudio del proyecto, implementación del mismo, fase de implementación y hasta la fase de pruebas.

Seguidamente, el desarrollo de este proyecto también ha supuesto todo un reto ya que, en gran parte, ha implicado la investigación de nuevas tecnologías que no se habían tratado a lo largo del máster. Tecnologías tales como el uso de frameworks JavaScript punteros, el trabajo con librerías nuevas, el desarrollo de aplicaciones Big Data (Kafka) y la mejora en la calidad de código gracias a las exigencias de SpaceX.

Para finalizar, solo cabe mencionar que la realización de éste trabajo de final de máster ha sido de un inestimable valor, tanto a nivel de conocimientos, como a nivel de vida, ya que aparte de aprender todo el conocimiento tecnológico e informático, gracias a Hyperloop, he podido conocer el mundo industrial y más en concreto el mundo aeroespacial, un mundo que está en continuo cambio para poder proporcionar al mundo las últimas tecnologías disponibles en el mercado.

Bibliografía

- Wikipedia Hyperloop.
  - https://es.wikipedia.org/wiki/Hyperloop
- MuleSoft Documentation.
  - https://docs.mulesoft.com/
- Kafka Dev Zone.
  - https://kafka.apache.org/documentation/
- Documentation for getting started, migration, installation, administration, configuration, optimization, and programming interfaces for the CircleCI 2.0 platform.
  - https://circleci.com/docs/
- Wikipedia DevOps.
  - https://es.wikipedia.org/wiki/DevOps
- Elastic Stack and Product Documentation.
  - https://www.elastic.co/guide/index.html
- AngularJS API Docs.
  - https://docs.angularjs.org/api
- Node.js v8.5.0 Documentation.
  - https://nodejs.org/es/docs/
- NPM Documentation.
  - https://docs.npmjs.com/
- jQuery API
  - https://api.jquery.com/
- PJRC (Teensy) Forum
  - https://forum.pjrc.com/

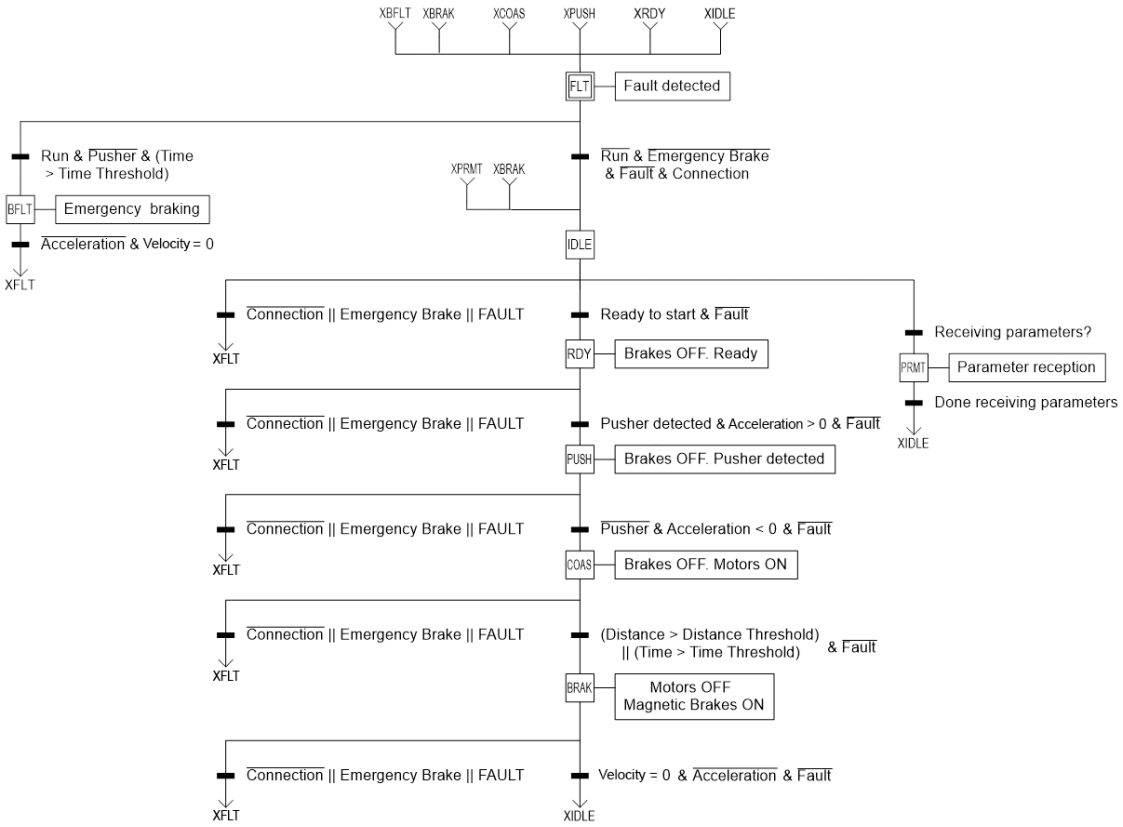
Anexos

Máquina de estados del prototipo

El sistema de aviónica asegura que el prototipo continúa a través del recorrido y se comunica con la interfaz gráfica de usuario. El prototipo contendrá tres controladores Teensy 3.6 en una configuración Master-Multi Slave. El protocolo de bus CAN se utilizará para las comunicaciones entre los microcontroladores maestro y esclavo. Todos los nodos están conectados entre sí a través de un bus de dos hilos. Para el CAN de alta velocidad, el bus está terminado con resistencias de 120 ohmios en cada extremo.

El software del prototipo utiliza una máquina de estado que alterna entre los estados exigidos por SpaceX que luego se enviará en el JSON a través del NAP. Las transiciones de estado se realizan basándose en criterios y umbrales predefinidos que pueden agregarse, eliminarse y modificarse fácilmente, dependiendo de las diferentes necesidades, sin reprogramar el microcontrolador (intercambio en caliente). Estos criterios son controlados por retroalimentación del sensor, tales como IMU, codificadores y otros sensores. También hay un conjunto de criterios persistentes que comprueban continuamente el estado de conexión del prototipo y si la señal de frenado de emergencia se envió o no desde la interfaz gráfica de usuario. El diagrama muestra las transiciones de estado a estado y las condiciones involucradas. Los Estados se enumeran como sigue:

0. **(0) FLT:** Conexión perdida, freno de emergencia o cualquier otro fallo detectado. Anule la carrera del tubo.
1. **(-1) BFLT**: Fallo + empujador separado. Activar frenos de emergencia.
2. **(1) IDLE**: Prototipo activado, pero no listo para ser empujado.
3. **(-2) PRMT**Recibe parámetros de la interfaz gráfica de usuario (umbrales y activación/desactivación manual de los actuadores).
4. **(-3) TEST:** Modo de prueba
5. **(2) RDY:** Prototipo listo para ser empujado.
6. **(3) PUSH**: Prototipo está siendo empujado.
7. **(4) COAS**: Prototipo separado del vehículo empujador. Modo autónomo (motores enganchados).
8. **(5) BRAK**: Frenado normal (no de emergencia).



El estado inicial del prototipo es **0: FAULT (FLT)**

- **FLT a IDLE**: si la ejecución no ha comenzado Y no hay ningún fallo detectado Y la conexión con la estación base está activa Y la estación base no ordena frenar (comando de frenado de emergencia).
- **FLT a BFLT**: si la carrera ha comenzado Y el empujador no está tocando el prototipo Y el temporizador de empuje está apagado (indicador de tiempo vía software).

1: IDLE

- **IDLE a RDY**: si el receptor recibe un 'r' de la estación base Y no se detecta ningún fallo.
- **IDLE a PRMT**: si el prototipo recibe un 'p' de la estación base.
- **IDLE a FLT**: si no hay conexión con la estación base O se ha detectado un fallo O la estación base no ordena frenar (comando de frenado de emergencia).
- **IDLE a TEST**: si el prototipo recibe un 't' de la estación base.

2: LISTO (RDY)

- **RDY a PUSH**: si se detecta el empujador Y Aceleración > 0 Y no se detecta ningún fallo.
- **RDY a FLT**: si no hay conexión con la estación base O fallo detectado O la estación base no ordena frenar (comando de frenado de emergencia).

3: PUSHING (PUSH)

- **PUSH a COAS**: si no se detecta el empujador Y aceleración < 0 (acelerómetro) Y no se detecta ningún fallo.
- **PUSH a FLT**: si no hay conexión con la estación base O fallo detectado O la estación base no ordena frenar (comando de frenado de emergencia).

4: COASTING (COAS)

- **COAS a BRAK**: si el prototipo ha alcanzado la distancia de frenado (encoders + lectores de cinta) O el temporizador ha alcanzado el tiempo de frenado (temporizador vía software) Y no se detecta ningún fallo.
- **COAS a FLT**: Si no hay conexión con la estación base O error detectado O la estación base no ordena frenar (comando de frenado de emergencia).

5: BREAKNG (BRAK)

- **BRAK a IDLE**: si velocidad por debajo de 0 Y aceleración por debajo de 0 Y no se detecta ningún fallo.

Cuando la aceleración y la velocidad llegan a 0 durante 5 segundos el prototipo sabe que tiene que ir al estado inicial después del recorrido efectuado (estado IDLE).

- **BRAK a FLT**: si no hay conexión con la estación base O se ha detectado un fallo O la estación base no ordena frenar (comando de frenado de emergencia).

-1: BRAKE + FAULT (BFLT)

- **BFLT a IDLE**: si velocidad por debajo de XX y aceleración por debajo de XXX.

Los frenos magnéticos actúan normalmente en el estado 5: FRENADO, cuando el prototipo ha llegado al final, y en -1. BRAKE + FAULT, cuando se produce un fallo y se ordena al prototipo que se detenga. Los frenos de fricción sólo se activan en -1. BRAKE + FAULT como frenos de emergencia.

-2: PARAMETROS (PRMT)

- **PRMT a IDLE**: Si el prototipo recibe un 'i' de la estación base.

-3: TEST

- **TEST a IDLE**: Si el prototipo recibe otro 't' de la estación base.

Criterios de anulación

Durante el estado de COAST, hay muchos criterios de anulación que harán que el prototipo se detenga. Pueden agruparse en:

Estado de conexión

En caso de pérdida de conexión con la estación base, el prototipo activará el mecanismo de frenado (estado FLT a BFLT). Se comprobará enviando cada 50 ms una señal keep-alive a la estación base, y si el microcontrolador máster no recibe ninguna señal en 1 segundo entrará en estado de fallo FLT para frenar inmediatamente.

Estado de temperatura

La temperatura ambiente, la temperatura de los motores y la temperatura de las baterías se van a supervisar para proporcionar un criterio de aborto para la ejecución.

1. La temperatura del motor debe ser inferior a 150 ºC.
2. La temperatura de la batería auxiliar debe ser inferior a 60 ºC.
3. La temperatura de la batería principal debe ser inferior a 72 ºC.

Estado de corriente / tensión

La corriente y el voltaje de cada batería van a ser monitorizados para proporcionar una visión de todo el sistema energético completo (temperatura, corriente y voltaje).

1. La corriente de la batería auxiliar debe ser inferior a 10 A.
2. El voltaje de la batería auxiliar debe estar entre 11.4 V y 15 V.
3. La corriente de la batería principal debe ser inferior a 350 A.
4. El voltaje de la batería principal debe estar entre 42,5 V y 72 V.

Estado del sistema de presión

La presión de todos los sistemas presurizados será monitorizada para asegurar el funcionamiento apropiado de componentes críticos tales como frenos o el tanque de aire principal.

1. El sistema de alta presión debe estar entre 250 psi y 3000 psi.
2. El sistema de baja presión debe estar entre 100 psi y 200 psi.

Estado de levitación

La distancia de la levitación y la distancia entre el prototipo y el rail se van a medir para asegurar que esta distancia sea siempre inferior a 2/10 ''

El incumplimiento de cualquiera de estos requisitos hará que el prototipo ingrese en estado FLT y frene con seguridad hasta que se detenga por completo.

Ejemplo JSON

Este anexo pretende mostrar un ejemplo de JSON que enviaban los microcontroladores a través del middleware hasta la interfaz gráfica de usuario:

{ "status": -3, "team_id": 315, "encoder_distance": 0, "timestamp": 0, "stripe_count": 0, "rawdata": "hello", "pusher_distance": 0, "switch_motor": 1, "switch_batteries": 1, "motorSpeed": 0, "acceleration": { "x": 0, "y": 0, "z": 0 }, "position": { "x": 0, "y": 0, "z": 0 }, "velocity": { "x": 0, "y": 0, "z": 0 }, "batteryMotor1": { "temperature": 0, "current": 0, "voltage": 0 }, "batteryMotor2": { "temperature": 0, "current": 0, "voltage": 0 }, "batteryElectronics1": { "temperature": 0, "current": 0, "voltage": 0 }, "batteryElectronics2": { "temperature": 0, "current": 0, "voltage": 0 }, "batteryElectronics3": { "temperature": 0, "current": 0, "voltage": 0 }, "temperatures": { "ambient_temperature": 0, "motor1_temperature": 0, "motor2_temperature": 0 }, "distance": { "z1": 0, "z2": 0, "z3": 0, "z4": 0 }, "actuator": { "wheel1": 0, "wheel2": 0, "emergency1": 0, "emergency2": 0, "eddy1": 0, "eddy2": 0 }, "angles": { "yaw": 0, "pitch": 0, "roll": 0 }, "pressure": { "wheels": 0, "emergency": 0, "eddy": 0, "pod": 0, "regulated": 0, "tank": 0 } }

JSON Schema

Una vez mostrado un ejemplo de JSON se especifica el JSON Schema que cumplen todos los JSON generados por los microcontroladores:

{"$schema":"http://json-schema.org/draft-04/schema#","definitions":{},"properties":{"acceleration":{"properties":{"x":{"type":"integer"},"y":{"type":"integer"},"z":{"type":"integer"}},"type":"object"},"actuator":{"properties":{"eddy1":{"type":"integer"},"eddy2":{"type":"integer"},"emergency1":{"type":"integer"},"emergency2":{"type":"integer"},"wheel1":{"type":"integer"},"wheel2":{"type":"integer"}},"type":"object"},"angles":{"properties":{"pitch":{"type":"integer"},"roll":{"type":"integer"},"yaw":{"type":"integer"}},"type":"object"},"batteryElectronics1":{"properties":{"current":{"type":"integer"},"temperature":{"type":"integer"},"voltage":{"type":"integer"}},"type":"object"},"batteryElectronics2":{"properties":{"current":{"type":"integer"},"temperature":{"type":"integer"},"voltage":{"type":"integer"}},"type":"object"},"batteryElectronics3":{"properties":{"current":{"type":"integer"},"temperature":{"type":"integer"},"voltage":{"type":"integer"}},"type":"object"},"batteryMotor1":{"properties":{"current":{"type":"integer"},"temperature":{"type":"integer"},"voltage":{"type":"integer"}},"type":"object"},"batteryMotor2":{"properties":{"current":{"type":"integer"},"temperature":{"type":"integer"},"voltage":{"type":"integer"}},"type":"object"},"distance":{"properties":{"z1":{"type":"integer"},"z2":{"type":"integer"},"z3":{"type":"integer"},"z4":{"type":"integer"}},"type":"object"},"encoder_distance":{"type":"integer"},"motorSpeed":{"type":"integer"},"position":{"properties":{"x":{"type":"integer"},"y":{"type":"integer"},"z":{"type":"integer"}},"type":"object"},"pressure":{"properties":{"eddy":{"type":"integer"},"emergency":{"type":"integer"},"pod":{"type":"integer"},"regulated":{"type":"integer"},"tank":{"type":"integer"},"wheels":{"type":"integer"}},"type":"object"},"pusher_distance":{"type":"integer"},"rawdata":{"type":"string"},"status":{"type":"integer"},"stripe_count":{"type":"integer"},"switch_batteries":{"type":"integer"},"switch_motor":{"type":"integer"},"team_id":{"type":"integer"},"temperatures":{"properties":{"ambient_temperature":{"type":"integer"},"motor1_temperature":{"type":"integer"},"motor2_temperature":{"type":"integer"}},"type":"object"},"timestamp":{"type":"integer"},"velocity":{"properties":{"x":{"type":"integer"},"y":{"type":"integer"},"z":{"type":"integer"}},"type":"object"}},"type":"object"}

Código fuente entregado

Se entregan tres ficheros comprimidos en formato ZIP:

1. codigoElectronica.zip: este fichero contiene todo el código de los microcontroladores y pruebas y test hechos en el laboratorio de Hyperloop
2. codigoMiddleware.zip: este comprimido contiene el proyecto y se puede importar directamente a Anypoint Studio para revisar todo el código del middleware.
3. codigoGUI.zip: contiene el proyecto de NodeJS con toda la aplicación web. Para ejecutar la aplicación hay que seguir los siguientes pasos:
    a. Situarse en la carpeta del proyecto
    b. Ejecutar el comando npm install
    c. Ejecutar el comando node server.js
    d. Abrir en el navegador http://localhost:5999