ESCUELA POLITÉCNICA NACIONAL

DEPARTAMENTO DE AUTOMATIZACIÓN Y CONTROL INDUSTRIAL

ESTUDIO Y EVALUACIÓN DE ATAQUES A UNA RED SIMULADA DE SENSORES VEHICULARES BASADA EN TECNOLOGÍA CAN, EMPLEANDO EL VEHICLE NETWORK TOOLBOX DE MATLAB

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE MAGISTER EN ELECTRÓNICA Y AUTOMATIZACIÓN CON MENCIÓN EN REDES INDUSTRIALES

ING. EDWIN PATRICIO TOAPANTA GALLEGOS

DIRECTOR: ING. PATRICIO JAVIER CRUZ DÁVALOS, PhD.

Quito, agosto 2024

AVAL

Certifico que el presente trabajo fue desarrollado por Edwin Patricio Toapanta Gallegos, bajo mi supervisión.

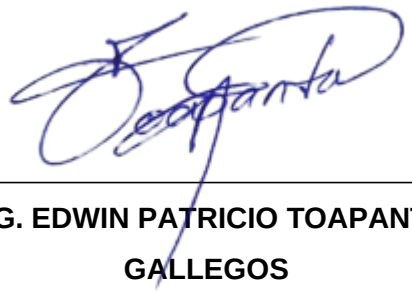
_____

Ing. Patricio Javier Cruz Dávalos, PhD.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Edwin Patricio Toapanta Gallegos, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.



ING. EDWIN PATRICIO TOAPANTA GALLEGOS

DEDICATORIA

A Dios, por darme la fortaleza y la sabiduría para seguir adelante en este camino. A mi padre, por su guía y enseñarme el valor del trabajo duro. A mi hermana, por su apoyo incondicional, por ser siempre una fuente de motivación y fortaleza en mi vida. A mi abuelita, aunque ya no esté conmigo, su amor y enseñanzas siguen presentes en cada paso que doy. Este logro es para ustedes.

Edwin Toapanta

AGRADECIMIENTO

Quiero expresar mi agradecimiento a todas las personas que, de alguna forma, han contribuido a la realización de este trabajo. A mi director de tesis, el Ing. Patricio Javier Cruz Dávalos, PhD., por su invaluable guía, paciencia y conocimientos que fueron fundamentales para la culminación de esta investigación. A la Escuela Politécnica Nacional, por brindarme las herramientas y el entorno académico necesario para desarrollar este proyecto. Finalmente, a todas aquellas personas que, aunque no mencionadas aquí, han sido parte de este proceso y cuyo apoyo ha sido crucial.

Edwin Toapanta

ÍNDICE DE CONTENIDO

RESUMEN

El presente trabajo de titulación se enfoca en el estudio y evaluación de ataques a una red simulada de sensores vehiculares basada en tecnología CAN, utilizando el Vehicle Network Toolbox de MATLAB. La creciente integración de dispositivos electrónicos en los automóviles modernos ha mejorado su eficiencia y seguridad, pero también ha introducido vulnerabilidades en su intercomunicación que pueden ser explotadas por atacantes malintencionados. En particular, el protocolo CAN, ampliamente utilizado en la comunicación Intra-vehiculares, presenta debilidades en términos de integridad, confidencialidad y disponibilidad, lo que lo hace susceptible a ataques como la intercepción de comunicación, la inyección de datos y la denegación de servicio.

En el presente trabajo se simulan dos sistemas vehiculares críticos: el sistema antibloqueo de frenos (ABS) y el sistema de encendido sin llave (Keyless); los mismos se implementan en MATLAB/Simulink, incorporando la comunicación CAN mediante el Vehicle Network Toolbox. Posteriormente, se simulan dos tipos de ataques: un ataque de denegación de servicio (DoS) al sistema ABS y un ataque de inyección de datos al sistema de encendido. Los resultados obtenidos permiten evaluar el impacto de estos ataques en el funcionamiento normal de los sistemas y extraer conclusiones relevantes sobre las vulnerabilidades del protocolo CAN y la importancia de implementar medidas de seguridad adecuadas en los vehículos modernos.

PALABRAS CLAVE: CAN, Seguridad Vehicular, DoS, Inyección de Datos, Vehicle Network Toolbox.

ABSTRACT

This work focuses on the study and the evaluation of attacks on a simulated sensor network based on the CAN protocol by using the Vehicle Network Toolbox from MATLAB. The increasing integration of electronics devices in moderns vehicles has improved their efficiency and safety, but it has also introduced vulnerabilities in their communication that can be exploited by malicious attackers. In particular, the CAN protocol, widely used in intra-vehicular communication, presents weaknesses in terms of integrity, confidentiality and availability, making it susceptible to attacks such as communication interception, data injection and denial of service.

In this study, two critical vehicular systems are simulated: the anti-lock system (ABS) and the keyless ignition system; both implemented in MATLAB/Simulink using CAN communication by means of the Vehicle Network Toolbox. Subsequently, two types of attacks are simulated: a denial-of-service (DoS) attack on the ABS system and a data injection attack on the ignition system. The obtained results allow for an evaluation of the impact of these attacks on the normal operation of the systems and provide relevant conclusions about the vulnerabilities of the CAN protocol and the importance of implementing security measures in modern vehicles.

KEYWORDS: CAN, Vehicular Security, DoS, Data Injection, Vehicle Network Toolbox.

1. INTRODUCCIÓN

Los avances tecnológicos en la industria automotriz se han enfocado cada vez más en la integración de dispositivos electrónicos dentro de los automóviles. Los vehículos modernos, lejos de ser solo medios de transporte, son complejas redes de subsistemas interconectados, donde la electrónica desempeña un papel central en el control y monitoreo del vehículo. Esta fusión de la mecánica y electrónica ha sido fundamental para mejorar la eficiencia, la seguridad y la experiencia del usuario en la conducción, pero con ella ha surgido un nuevo desafío: la seguridad ante ataques a estos subsistemas del vehículo. [1]

Esta integración requiere de una comunicación robusta entre los dispositivos involucrados, sobre todo considerando la importancia de intercambiar información para que el vehículo opere correctamente. Debido a esta necesidad, es fundamental una intercomunicación, pero reduciendo el posible exceso de cableado entre dispositivos; por lo cual se ha implementado el bus CAN (Controller Area Network).

El protocolo CAN fue establecido por Bosch como una red de trabajo para control industrial. CAN constituye un sistema electrónico sofisticado especialmente conveniente para tareas de networking [1] de dispositivos “inteligentes”, así como de sensores y actuadores dentro de un sistema o subsistema. Debido a su costo y eficiencia en la rapidez de transmisión de datos, las compañías automotrices vieron en CAN un protocolo robusto capaz de implementar un sistema de comunicación óptimo a bordo de los vehículos [2]. La tecnología CAN tiene como característica principal que se utilizan en la mayoría de los vehículos, además es una red multimaestro y estandarizada, esto quiere decir que en una red CAN, varios dispositivos pueden actuar como maestros o controladores, no existe un maestro único centralizado. Esta capacidad de comunicación distribuida y sin un maestro centralizado hace que el CAN sea muy adecuado para aplicaciones en las que la redundancia y la confiabilidad son fundamentales, como en la industria automotriz y en sistemas de control industrial. Es estandarizada porque permite asegurar la consistencia y confiabilidad del protocolo en una variedad de aplicaciones industriales y automotrices, en la cual se incluye aspectos como la velocidad de transmisión de datos, el formato de los mensajes, la topología de la red y la arquitectura eléctrica, mismas que garantizan y agilizan la transmisión de datos siendo importante ya que permite diagnosticar averías. Además, está estandarizada por que define las dos primeras capas del modelo OSI de interconexión de sistemas, siendo estas la capa física y la capa de enlace de datos.

En consecuencia, hoy en día, los vehículos ya no son simples dispositivos mecánicos móviles, sino que son supervisados y controlados de manera permanente por dispositivos digitales internos que son coordinados a través de redes vehiculares, principalmente empleando la tecnología del bus CAN. Por esto, la industria automotriz ha estado constantemente considerando a la seguridad como un problema crítico de ingeniería, sin embargo, el desarrollo de protocolos de comunicación, en primera instancia, no ha anticipado la manipulación no deseada de información dentro de la red vehicular. Por ejemplo, a causa de las vulnerabilidades de la red CAN podrían presentarse situaciones extremadamente peligrosas, como el secuestro remoto del vehículo, sabotajes de sistemas internos como de la dirección, frenos, airbag, etc.

En base a estos desafíos es fundamental realizar un estudio de la red CAN, que facilite el conocer y evaluar las vulnerabilidades de esta tecnología de red de sensores y actuadores vehiculares. Identificando posibles riesgos, mismos que ayuden a plantear medidas que protejan la integridad y confiabilidad de la seguridad de dicha red de comunicación. Una vez estudiadas dichas vulnerabilidades, se considerarán dos de ellas para ser simuladas con la ayuda del software MATLAB y la utilización de la librería Vehicle Network Toolbox. Esto permitirá analizar los efectos de las mismas y plantear recomendaciones tanto respecto a la seguridad de la red CAN como a la utilización de las herramientas de MATLAB asociadas a este bus de comunicación, como un soporte para futuros estudios.

Por esto, el presente trabajo de titulación se enfoca en la necesidad de comprender las potenciales vulnerabilidades de la tecnología CAN en la implementación de una red de sensores-actuadores dentro de un vehículo, estudiando los posibles ataques y debilidades que presenta dicho sistema. Particularmente se recopilará información referente a las definiciones, funcionamiento y característica que presenta la tecnología CAN, lo cual ayuda a identificar diferentes ataques que tengan dicha tecnología; permitiendo seleccionar dos posibles ataques que puedan ser utilizados de forma simulada. Esto servirá como apoyo para comprender de mejor manera dichos casos sin la necesidad de manipular un vehículo.

Para este fin, se utiliza el software Matlab con el Vehicle Network Toolbox como una herramienta para la simulación de la red y se procede a simular. Se seleccionarán dos de los diferentes ataques estudiados de la tecnología CAN que puedan ser simulados en el software Matlab. Esto servirá de soporte para analizar y evaluar las posibles causas que presentan estas y plantear posibles recomendaciones. La red simulada empleando el Toolbox Vehicle Network de Matlab, se basa en la estructura y funcionamiento de sensores y actuadores básicos en un vehículo por ejemplo, los sensores de velocidad y oxígeno,

mientras que en el caso de actuadores los del sistema ABS y el de inyección del combustible. Se desarrolla dos posibles escenarios de ataques los cuales están diseñados para que puedan representar posibles situaciones reales que se dan en los sensores y actuadores que pueden ser vulnerados. Dichos escenarios ayudan a analizar y evaluar el impacto que tienen estos ataques a la red CAN. Mediante la simulación se visualiza como los ataques afectan a la integridad y seguridad de la comunicación en la red CAN simulada. Una vez realizada la simulación se efectúa un análisis para evaluar el impacto que tiene los ataques a la red CAN y en base a los mismos se presentan recomendaciones que ayuden a futuros estudios que sean de apoyo tanto en el ámbito académico como en industrial.

1.1 Objetivo General

Estudiar y evaluar ataques de una Red Simulada de Sensores Vehiculares basada en Tecnología CAN, empleando el Vehicle Network Toolbox de MATLAB

1.2 Objetivos Específicos

- Realizar una revisión bibliográfica sobre el bus CAN respecto a la implementación de una red de sensores vehiculares; así como los ataques que se pueden presentar en este tipo de red, además del funcionamiento de la librería Vehicle Network Toolbox de MATLAB.
- Desarrollar y diseñar en el software MATLAB una simulación de una red de sensores-actuadores basados en el bus CAN
- Seleccionar y simular dos posibles ataques sobre la red CAN de sensores-actuadores implementada empleando MATLAB y su Vehicle Network Toolbox.
- Analizar los ataques seleccionados y sus consecuencias

1.3 Alcance

- Se recopilo información bibliográfica, funcionamiento y características generales del bus CAN además de identificar los distintos ataques que presenta la tecnología CAN de una red de sensores vehiculares.
- Se eligieron dos ataques que sean representativos y relevantes para el entorno de redes de sensores vehiculares. Estos ataques se seleccionaron con base en su impacto potencial y su frecuencia de aparición.

- Se realizó una investigación sobre el funcionamiento y uso de la librería Vehicle Network Toolbox, disponible en el entorno MATLAB. Esto permite comprender cómo simular una red de sensores-actuadores basada en tecnología CAN y cómo incorporar escenarios de ataque.
- Considerando al menos dos sensores (el de velocidad y de presencia) y al menos dos actuadores (válvulas de presión y el motor de arranque) se implementó una red CAN simulada en MATLAB utilizando la librería Vehicle Network Toolbox.
- Empleando la red CAN de sensores/actuadores virtuales creada en MATLAB, se procede a simular los ataques seleccionados, recopilando la información necesaria para su posterior análisis.
- Del análisis de los resultados de los dos ataques simulados se presentan recomendaciones para mitigar los efectos de estos, así como para futuros estudios referentes a la simulación del bus CAN.

1.4 Marco Teórico

1.4.1. Redes Intra-vehiculares

Las redes Intra-vehiculares en sistemas automotrices implican la integración de dispositivos de seguridad, asistencia al conductor y entretenimiento utilizando diversos protocolos de red. Estas redes se están volviendo más complejas con el aumento del número de sensores, actuadores y requisitos de ancho de banda para aplicaciones de asistencia al conductor.

Los dispositivos automotrices suelen ser clasificados en dominios funcionales, agrupando aplicaciones y sensores según su funcionalidad o ubicación física. Las redes existentes muestran una heterogeneidad significativa, con diferentes protocolos como CAN para datos de control del chasis, FlexRay para aplicaciones críticas de seguridad y LIN para mensajes de control serial. Estas tecnologías pertenecen a la capa física de la red de comunicación automotriz [3], véase Figura 1.1.

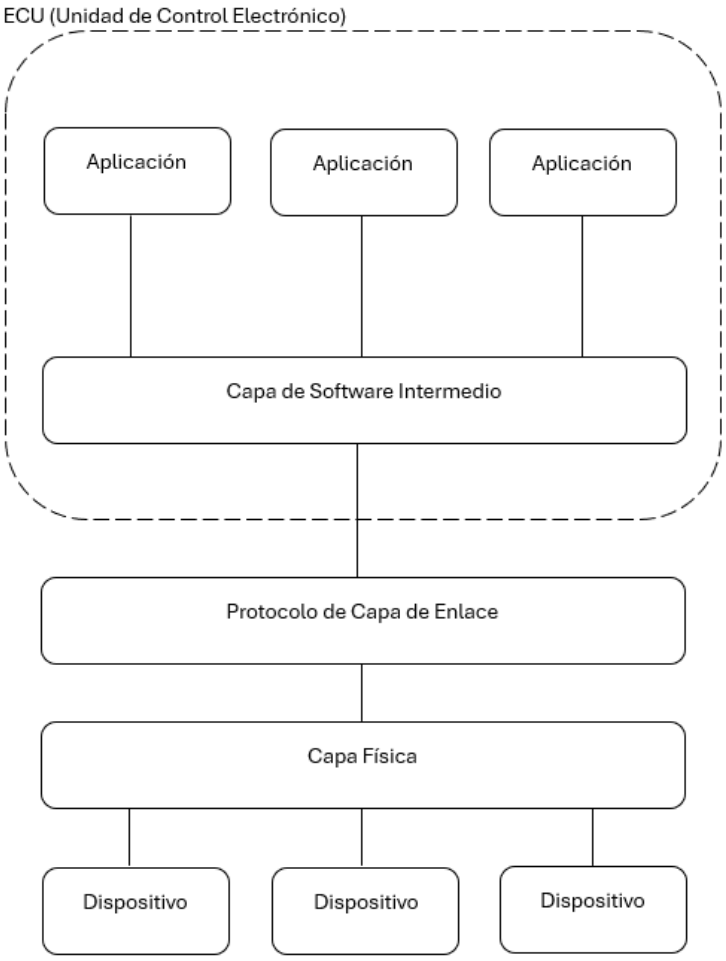


Figura 1.1 Capas de la red comunicación vehicular.[3]

La Tabla 1.1 presenta información general sobre la velocidad máxima de bits, medio de comunicación y protocolo de transmisión de cada una de estas tecnologías.

Tabla 1.1. Tecnologías de capa física automotriz [3].

Tecnología	Bitrate	Medio	Protocolo	Alcance Máximo	Máximo de dispositivos
LIN	20 Kbps	Cable Simple	Serial, maestro-esclavo	40 metros	16
CAN	1 Mbps	Par Trenzado	Serial, broadcast, multimaestro	40 metros (alta velocidad), 1 km (baja velocidad)	Teóricamente ilimitado, limitado por el hardware
CAN FD	8 Mbps	Par Trenzado	Serial, broadcast, multimaestro	40 metros (alta velocidad), 1 km (baja velocidad)	Teóricamente ilimitado, limitado por el hardware
FlexRay	20 Mbps	Par Trenzado	Serial, TDMA (Acceso Múltiple por división de Tiempo	24 metros (10 Mbps)	64

1.4.1.1. LIN

Es un bus de comunicación serial maestro-esclavo económico, desarrollado a finales de la década de 1990 por el consorcio LIN. Surgió como respuesta a la necesidad de una alternativa más asequible al protocolo CAN para elementos de menor importancia dentro de la red vehicular [4].

El protocolo facilita la transferencia de datos entre un maestro y hasta 16 esclavos a través de un cable simple sin blindaje. Su funcionamiento se basa en tres elementos fundamentales:

- Interrupción: El nodo maestro comienza la comunicación enviando una señal de interrupción (break) que sincroniza todos los nodos esclavos y les indica el comienzo de una nueva trama de datos.
- Sincronización: Después de la interrupción, el maestro envía un campo de sincronización que permite a los esclavos ajustar sus relojes internos y prepararse para recibir los datos.

- Identificación: Cada trama de datos contiene un identificador único que especifica el tipo de mensaje y el nodo esclavo al que va dirigido. Esto ayuda que los esclavos filtren los mensajes y solo procesen aquellos que les corresponden.

Su aplicación se limita por la velocidad de los mensajes que puede transmitir. Además, la tasa de transferencia de datos se encuentra en el orden de los 20 Kbps y su alcance no supera los 40 metros.

1.4.1.2. CAN

El Controller Área Network (CAN) representa un sistema altamente fiable que facilita una comunicación eficiente entre unidades de control electrónico. Su propósito fundamental es prevenir y mitigar problemas durante la transmisión de información entre los puntos de comunicación, garantizando una transferencia sin errores. Esta tecnología opera mediante la creación de canales de comunicación entre los puntos mencionados anteriormente.

La comunicación CAN se realiza mediante un protocolo específico. La transmisión de datos se lleva a cabo mediante impulsos eléctricos, cuya señal se presenta en forma de onda cuadrada o, comúnmente, como modulación por ancho de pulso (PWM). Estas señales atraviesan un bus da datos compuesto por dos cables entrelazados. La unidad de control está equipada con un transceptor que convierte los impulsos eléctricos en mensajes binarios. Para su interpretación, se requiere un controlador y un microprocesador.

Dada la relevancia del protocolo CAN en el presente trabajo de titulación se brindará más detalle en la sección 1.4.2.

SAE J1939 es un conjunto de estándares basados en CAN que define la comunicación en los vehículos comerciales y maquinaria pesada. Proporciona un marco común para la comunicación entre diferentes sistemas y componentes, facilitando la interoperabilidad y el diagnostico. J1939 utiliza identificadores de 29 bits para direccionar mensaje y define parámetros específicos para diversas aplicaciones, como motores, trasmisiones, frenos y sistemas de control de emisiones.

XCP (Universal Measurement and Calibration Protocol) es un protocolo de comunicación utilizando para la medición, calibración y programación de ECUs en tiempo real. XCP se basa en CAN como capa de transporte y proporciona un conjunto de comandos y servicios para acceder a los datos internos de la ECU, modificar parámetros y cargar nuevo software. XCP es ampliamente utilizado en el desarrollo y prueba de sistemas automotrices, así como en aplicaciones de ajuste y optimización del rendimiento.

1.4.1.3. CAN FD

En 2012, Bosch presento el protocolo CAN FD (Flexible Data-Rate), que ofrece una tasa de datos efectiva de 5 Mbps y soporta una carga útil mayor de hasta 64 bytes, en contraste con los 8 bytes del CAN clásico. El protocolo CAN FD es retro compatible y puede coexistir con los nodos CAN clásicos, y ambos están estandarizados bajo la norma ISO 11898-1:2015

CAN FD conserva la robustez y simplicidad de CAN, pero introduciendo mejoras significativas en la transmisión y la longitud de datos. Esto se logra mediante la implementación de un esquema de dos velocidades: una velocidad de bit más baja para el arbitraje y una velocidad de bit más alta para la transmisión de datos, que puede alcanzar hasta los 8 Mbps [5].

1.4.1.4. FlexRay

Las principales ventajas de FlexRay sobre CAN son su flexibilidad, mayor tasa máxima de datos, 10 Mbps, y su comportamiento determinista de acceso múltiple por división de tiempo (TDMA). Sin embargo, los nodos FlexRay son más costosos que los nodos CAN, lo cual puede no ser atractivo para la fabricación en alta escala [4]. Está basado en la norma ISO 17458, por ello es empleado en sistemas de seguridad activa y de precisión en el vehículo.

FlexRay proporciona latencia y fluctuación constantes a través de la sincronización de reloj; estas características significan que se utiliza frecuentemente en aplicaciones de ‘drive-by-wire’. Las aplicaciones “drive-by-wire” reemplazan a los sistemas mecánicos tradicionales de control de vehículos, como la dirección, el acelerador y los frenos con sistemas electrónicos

1.4.2. CAN

El Controller Área Network es un protocolo de comunicación en bus serie desarrollado por Bosch en la década de 1980. Este protocolo establece un estándar para la comunicación eficiente y confiable entre sensores, actuadores, controladores y otros nodos de aplicaciones en tiempo real [4]. La transmisión de datos tiene una secuencia lógica, que involucra tres componentes principales, véase Figura 1.2.

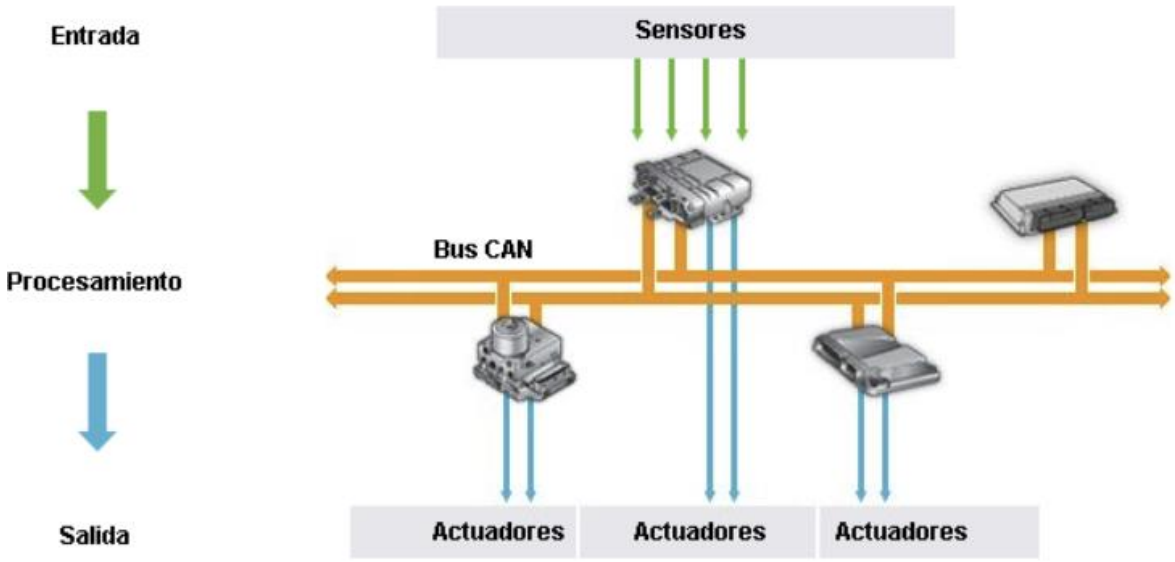


Figura 1.2. Secuencia Lógica [6].

Sensor (Entrada): Son dispositivos que recopilan información del entorno, como velocidad, posición, presión o temperatura. Estos datos se convierten en señales eléctricas que se transmiten a través de la comunicación CAN.

Unidad de control electrónico (ECU) (Procesamiento): Las ECU son los responsables de procesar los datos recibidos de los sensores, tomar decisiones basadas en esa información y generar comandos de control que se envían a los actuadores. En muchos sistemas automotrices modernos, una ECU central, conocida como ECU Gateway, juega un papel crucial en la coordinación de la comunicación entre diferentes ECUs. Actúa como un punto de interconexión que filtra, enruta y traduce los datos a través de diferentes redes de comunicación como CAN, LIN, FlexRay y Ethernet, asegurando que solo la información relevante y segura sea transmitida entre sistemas críticos y no críticos.

Actuadores (Salida): Son dispositivos que realizan acciones físicas en respuesta a los comandos recibidos de la ECU.

La red CAN en los automóviles opera a diferentes velocidades según la aplicación y la prioridad de los datos transmitidos. La norma ISO 11898 establece dos velocidades principales: 125 Kb/s y 1 Mb/s. Sin embargo, en la industria automotriz es común encontrar implementaciones a velocidades intermedias de 250 Kb/s y 500 Kb/s [7]. La elección de velocidad depende de la importancia de la función. Las funciones de seguridad, como el control del motor, el sistema de frenos antibloqueo (ABS) y los airbags, requieren una comunicación rápida y confiable. Las funciones menos críticas, como el control de las

luces, los espejos retrovisores, el sistema de audio y la navegación, pueden operar a velocidades bajas.

Es importante destacar que, la red CAN puede operar a diferentes velocidades, todos los nodos de la red durante la inicialización deben negociar y acordar una única velocidad de comunicación. La velocidad de la red se mantiene constante durante la operación normal para asegurar una comunicación efectiva y sincronizada entre todos los dispositivos conectados. Adicionalmente, la velocidad de comunicación en una red CAN depende de las distancias físicas entre los nodos; si la distancia es menor a 40 metros, la taza de transmisión puede alcanzar hasta 1 Mbps.

1.4.2.1. Modelo OSI

El protocolo CAN estandariza la capa física y de enlace de datos, siendo estas las dos capas inferiores del modelo OSI de comunicación, Figura 1.3. La ISO 11898-1 cubre la capa de enlace de datos, la ISO 11898-2 capa física para altas velocidades e ISO 11898-3 para baja velocidad [8].

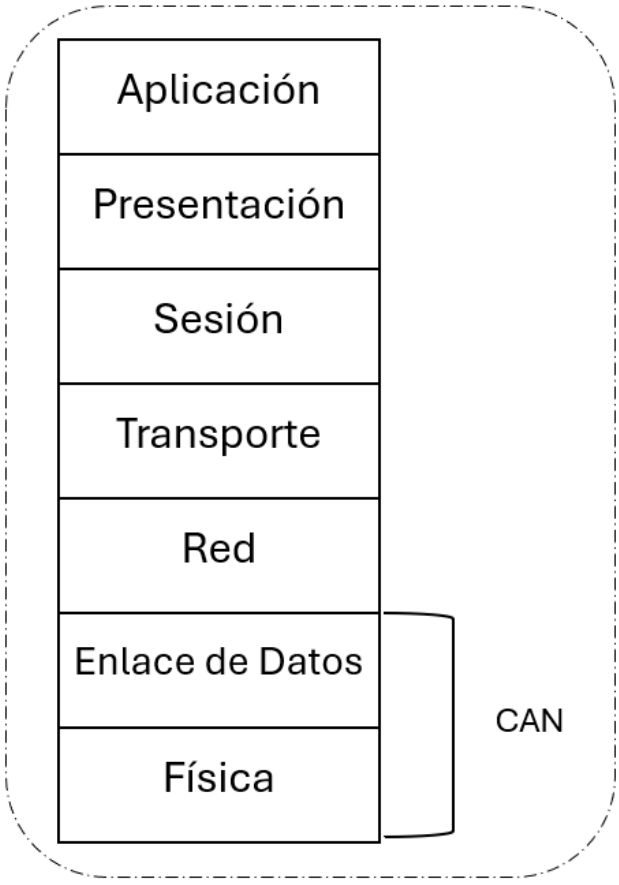


Figura 1.3. Modelo OSI. [3]

1.4.2.1.1. Capa Física

La capa física es la primera capa del modelo OSI y se encarga de la transmisión y recepción de los datos crudos a través de un medio físico. En el protocolo CAN, la capa física se divide en tres subcapas.

- Physical Signaling Layer (PSL): Se encarga de la sincronización y temporización de los bits.
- Physical Medium Attachment (PMA): Convierte los niveles lógicos de transmisión y recepción a los especificados por el protocolo CAN
- Medium Dependent Interface (MDI): Especifica el conector y el medio de transmisión.

Para la transmisión de datos con la tecnología CAN, se emplean impulsos eléctricos cuya señal es cuadrada, comúnmente conocida como modulación por ancho de pulso (PWM). Estas señales son trasmitidas a través de un bus de datos, compuesto por dos cables entrelazados. La unidad de control está equipada un transceptor, que convierte los impulsos eléctricos en mensajes codificados en formato binario. Para que estos mensajes sean interpretados correctamente, se requiere de un controlador y un microprocesador.

La comunicación se realiza de manera diferencial mediante las señales CAN-H y CAN-L. El voltaje para CAN-H puede tomar dos valores: Bit recesivo a 2.5 volts y Bit Dominante a 3.5 Volts. El voltaje para CAN-L también puede tomar dos valores: Bit recesivo a 2.5 volts y Bit dominante a 1.5 volts, ambos referenciados a tierra. La diferencia entre estos voltajes genera una señal entre 0 y 2 Voltios, indicando el nivel lógico de un bit. Este modo diferencial permite el rechazo del ruido [9], véase Figura 1.4.

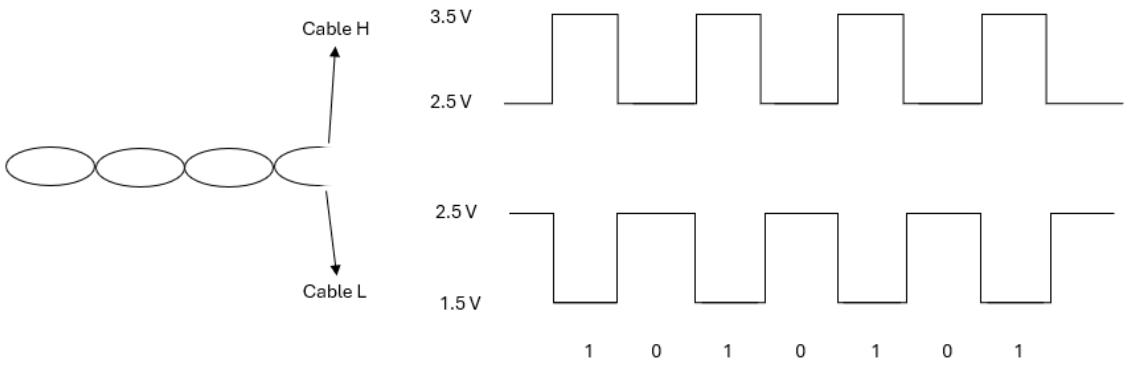


Figura 1.4. Voltajes, bus CAN.

El estándar ISO 11898-2 establece que se deben utilizar dos cables con resistencias de terminación de 120 Ohmios, cables trenzados o apantallados, evitando derivaciones y utilizando diferentes longitudes de línea mínima según la velocidad [7].

1.4.2.1.2. Capa de enlace de Datos

La capa de enlace de datos es la segunda capa del modelo OSI y se encarga de la transferencia de datos entre nodos de la red. El protocolo de acceso al medio empleado en la red es CSMA/CD + AMP (Carrier Sense Multiple Access/Collision Detection + Arbitration on Message Priority). Este protocolo permite a los nodos conectados monitorear el medio constantemente para detectar si un mensaje está siendo transmitido. Así, los nodos evitan enviar datos cuando la red está ocupada. En ocasiones, dos nodos pueden intentar enviar mensajes simultáneamente. Lo cual podría llevar a una colisión. Este conflicto se resuelve mediante un sistema de prioridades, donde el nodo con el identificador más bajo tiene preferencia para transmitir su mensaje en caso de colisión.

El protocolo CAN se caracteriza por un mecanismo de arbitraje no destructivo que gestiona el acceso al bus de datos de manera eficiente y determinista. Este proceso se basa en la asignación de prioridades a los mensajes mediante identificadores binarios, donde un valor menor indica una mayor prioridad.

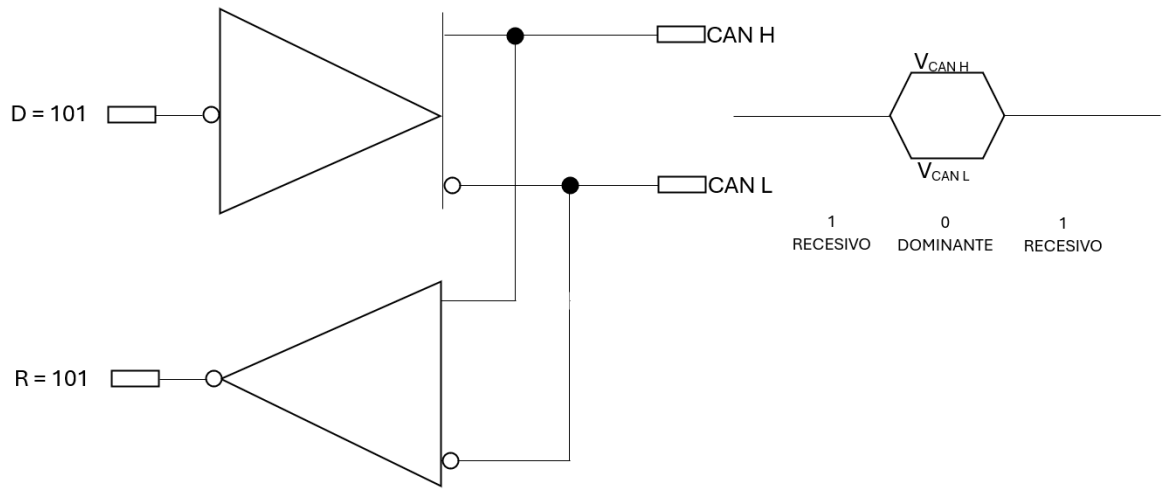


Figura 1.5. Lógica inversa, bus CAN. [10]

La naturaleza cableada del bus CAN, junto con la relación inversa entre los niveles lógicos del bus, y los pines de entradas y salidas de los transceptores (véase Figura 1.5), facilita la implementación de este arbitraje en caso de colisión. Esto es, el nodo con el identificador más bajo prevalece, asegurando que los mensajes críticos no se vean interrumpidos.

Este arbitraje transparente al usuario se realiza a nivel de bit y es gestionado por el controlador CAN. La monitorización continua de las transmisiones propias permite a los nodos detectar colisiones y ceder el bus a los mensajes de mayor prioridad, garantizando así una comunicación fluida y determinista en entornos de tiempo real.

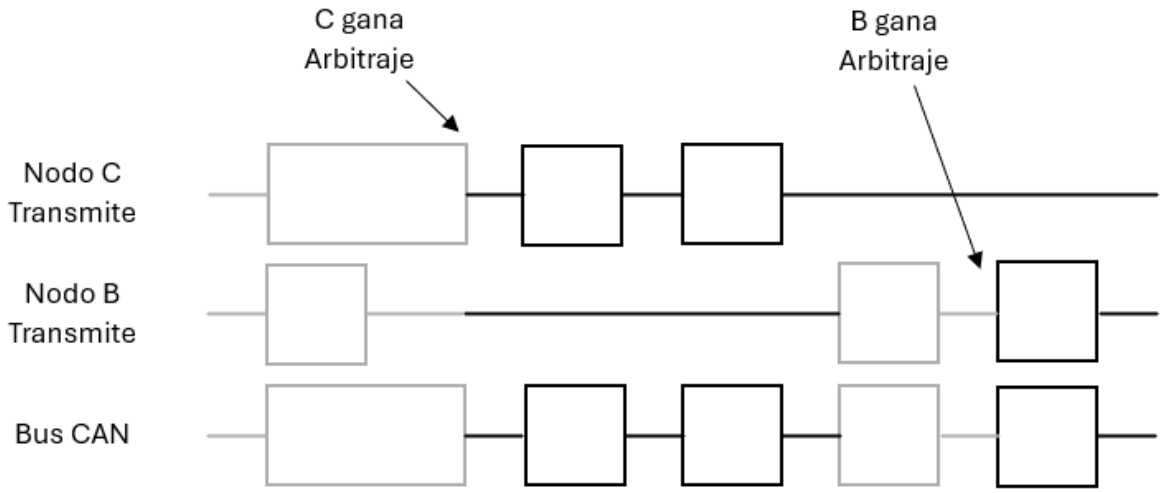


Figura 1.6. Arbitraje Bus CAN. [10]

La Figura 1.6. ilustra el proceso de arbitraje en CAN. Si el bit recesivo del nodo B es sobrescrito por un bit de mayor prioridad del nodo C, B detecta las discrepancias entre el estado del bus y su bit transmitido. En consecuencia, B detiene su transmisión, permitiendo que C continue con su mensaje. El nodo B reintentara transmitir cuando el bus se encuentre libre. Esta funcionalidad, integrada en la capa física de señalización de la norma ISO 11898, reside por completo en el controlador CAN.

1.4.2.2. Estructura del Mensaje

SOF	11-bit Identificador	RTR	IDE	r0	DLC	0...8 Bytes Data	CRC	ACK	EOF	IFS

Figura 1.7. CAN estándar: 11-bit Identificador [10]

La Figura 1.7. presenta visualmente los campos de bits, y su significado en el contexto del mensaje CAN estándar. Estos campos son:

SOF (Start of Frame): Bit dominante que señala el inicio de la transmisión y la sincronización de los nodos del bus.

Identificador: Secuencia de 11 bits que determina la prioridad del mensaje, donde un valor binario de valor menor implica una mayor prioridad.

RTR (Remote Transmission Request): Bit que indica si la trama es una solicitud de datos (dominante) o una transmisión de datos (recesivo).

IDE (Identifier Extension): Bit que distingue entre los identificadores estándar (11 bits) y extendidos (29 bits).

r0: Bit reservado para futuras extensiones del CAN estándar.

DLC (Data Length Code): Campo de 4 bits que especifica la cantidad de bytes de datos en la trama (0-8 bytes).

Datos: Contiene la información a transmitir, con una longitud máxima de 64 bits.

CRC (Cyclic Redundancy Check): Código de 15 bits para la detección de errores en los datos transmitidos.

ACK (Acknowledgment): Bit que confirma la recepción correcta de la trama por parte de los nodos.

EOF (End of Frame): Campo de 7 bits que marca el final de la trama y deshabilita el mecanismo de relleno de bits. El mecanismo de relleno de bits consiste en insertar bits adicionales (bits de relleno) en la trama de datos cuando se detecta una secuencia de cinco bits dominantes o recesivos consecutivos. Esto permite que los nodos de la red sepan cuando esperar la siguiente trama y mantengan la sincronización adecuada.

IFS (Interframe Space): Espacio entre tramas de 7 bits, proporciona un tiempo de separación entre tramas para permitir que los nodos de la red recuperen y se preparen para recibir la siguiente trama.

Figura 1.8. CAN extendido: 29-bits Identificador. [10]

En la mayoría de los casos, el CAN estándar utiliza un identificador de 11 bits para interpretar diferentes tipos de mensajes, lo que permite hasta 2048 mensajes de ID diferentes. Después, el identificador CAN se expandió a 29 bits, dando lugar a lo que se

conoce como CAN extendido (CAN extended) [11]. La Figura 1.8. ilustra la estructura de un mensaje CAN extendido, que difiere del estándar en los siguientes aspectos:

SRR (Substitute Remote Request): Este bit reemplaza al RTR en la misma posición, actuando como marcador de posición en el formato extendido

IDE (Identifier Extension): Un bit recesivo en esta posición indica la presencia de 18 bits adicionales para el identificador, extendiendo su rango.

r1: Bit de reserva adicional ubicado antes del DLC, reservado para futuras modificaciones del estándar.

Estos cambios permiten una mayor flexibilidad en la identificación de mensajes y nodos en redes CAN más complejas, manteniendo la compatibilidad con el CAN estándar.

1.4.2.3. Conector de Enlace de datos (DCL)

El DLC es el conector OBD-II (On-Board Diagnostic) de 16 pines utilizado en todos los vehículos fabricados desde 1996. Las herramientas de escaneo se conectan a la red CAN a través de esta conexión estándar. Algunos pines son obligatorios para todos los fabricantes, mientras que otros quedan a discreción del fabricante individual. Los pines 6 CAN H, y 14 CAN L proporcionan un enlace a la red CAN. Véase figura 1.9.

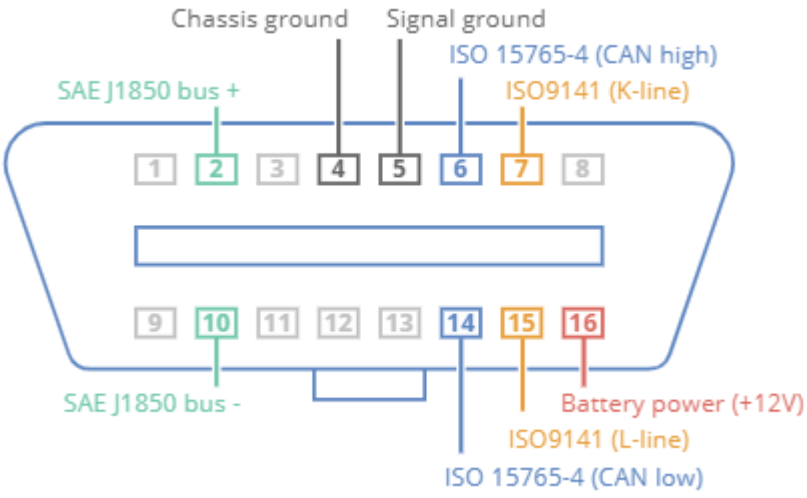


Figura 1.9. Conector OBD-II. [12]

1.4.2.4. Vulnerabilidades del Bus CAN

1.4.2.4.1. Integridad

La integridad en un sistema de comunicación garantiza la precisión y la totalidad de la información transmitida, protegiéndola de alteraciones no autorizadas. Con este fin, el protocolo CAN incorpora un mecanismo como CRC (Cyclic Redundancy Checks) y ECM (Error Confinement Mechanism) para detectar y corregir errores a nivel de bit. Sin embargo, estos métodos no son suficientes ya que el sistema sigue siendo vulnerable a modificaciones maliciosas en ausencia de una autenticación [6].

1.4.2.4.2. Confidencialidad

La confidencialidad en sistema de comunicación, entendida como la protección de datos frente a accesos no autorizados y a su explotación, es un aspecto critico en aplicaciones sensibles. Sin embargo, el protocolo CAN, en su diseño original, no incorpora mecanismos inherentes para garantizar esta confidencialidad, como la autenticación o el cifrado de datos [6].

Esta ausencia de medidas de seguridad intrínsecas expone los datos transmitidos en redes CAN a potenciales riesgos de interceptación y manipulación por parte de actores malintencionados. Para aplicaciones que requieren un alto nivel de confidencialidad, es imprescindible implementar protocolos adicionales que incluyan mecanismos de autenticación robusta y cifrado de datos.

1.4.2.4.3. Disponibilidad

La disponibilidad, definida como el acceso ininterrumpido a un sistema por parte de usuarios autorizados, es un pilar fundamental en sistemas de comunicación confiables. Sin embargo, el protocolo CAN presenta vulnerabilidades inherentes a su mecanismo de arbitraje, que favorece a mensajes de alta prioridad. Esto puede resultar en la negación de acceso al bus para nodos de menor prioridad, especialmente en escenarios de transmisión continua por parte de nodos de alta prioridad. Esta característica, junto con la arquitectura del bus CAN, lo hace susceptible, por ejemplo, a ataques de denegación de servicio (DoS), donde un nodo malicioso puede monopolizar el bus e impedir la comunicación de otros nodos legítimos [6].

1.4.3. Ataques al protocolo del bus CAN.

Esta sección describe diferentes tipos de ataques en la red del bus CAN.

1.4.3.1. Ataques de Acceso Remoto

Los vehículos modernos, están equipados con diversas interfaces inalámbricas para funciones como radio, Bluetooth y sistemas antirrobo, por lo que presentan una vulnerabilidad potencial a través de estas conexiones. Investigaciones, como la de Checkoway [13], ha demostrado la posibilidad de comprometer estas interfaces mediante la implementación de ingeniería inversa, lo que permitiría a los atacantes malintencionados acceder a la red CAN del vehículo y controlar funciones importantes como el desbloqueo de puertas.

Aunque la mayoría de los vehículos utilizan una ECU Gateway para proteger la red CAN de amenazas externas, estudios recientes han revelado que esta Gateway también es susceptible a ataques. La explotación exitosa de esta vulnerabilidad podría otorgar a los atacantes acceso completo al sistema de seguridad del vehículo, lo que conlleva grandes riesgos de robo, manipulación de funciones críticas y, en consecuencia, accidentes potencialmente catastróficos [6].

1.4.3.2. Ataques de Acceso Físico

Los ataques de acceso físico en la seguridad del bus CAN representa una amenaza significativa debido a la posibilidad de acceso directo o indirecto al sistema. El puerto de diagnóstico a bordo (OBD) se destaca como un punto vulnerable clave, ya que permite el acceso a todos los nodos de la red CAN, e incluso en arquitecturas de red en la que se utiliza Gateway para dividir el bus CAN en múltiples segmentos o subredes [6].

La introducción de código malicioso a través del puerto OBD, OBD-II, mediante manipulación física de los cables o la explotación de vulnerabilidades en componentes electrónicos, puede comprometer la integridad de la red CAN. Permitiendo a los atacantes enviar o interceptar mensajes de componente críticos como frenos, dirección, sistema de encendido, etc.

Estas acciones maliciosas pueden tener consecuencias devastadoras, incluyendo:

- Pérdidas financieras: El fabricante puede enfrentar costos significativos por demandas legales por accidentes relacionados con la seguridad, daños a la reputación de la marca y pérdida de confianza del consumidor.
- Riesgos al consumidor: la manipulación de control del vehículo puede provocar accidentes graves o fatales, poniendo en peligro la vida de los ocupantes del vehículo y de otros usuarios de la vía.

La arquitectura abierta del bus CAN, caracterizada por la falta de mecanismos de seguridad robustos en su diseño original, como autenticación y cifrado de mensajes, facilita la comunicación entre los componentes del vehículo, pero también la hace vulnerable a ataques. En una red CAN cualquier nodo puede enviar y recibir mensaje sin restricciones, lo que permite a un atacante con acceso físico al bus inyectar mensajes maliciosos que puedan ser interpretados y ejecutados por los nodos legítimos

1.4.3.3. Ataques existentes

Se ha identificado tres categorías principales de ataques que explotan las vulnerabilidades inherentes del protocolo CAN:

- Eavesdropping (Intercepción de comunicación): Este ataque pasivo permite a un atacante interceptar y monitorear el tráfico de datos en el bus CAN sin interferir en la comunicación legitima. Aunque no modifica directamente el funcionamiento del sistema, el eavesdropping proporciona al atacante información valiosa que puede ser utilizado para planificar ataques más sofisticados. [14]

- Data Insertion (Inyección de datos): La falta de mecanismos de autenticación en CAN permite a un atacante inyectar mensajes Falsos o manipulados en la red. Estos mensajes fraudulentos pueden ser interpretados como comandos legítimos por los nodos receptores, lo que podría resultar en acciones no deseadas o peligrosas. [14]

- Denial of Service (DoS, Denegación de Servicio): Un atacante puede inundar el bus con mensajes de alta prioridad, impidiendo que los nodos legítimos transmitan sus propios mensajes. Esto puede interrumpir el funcionamiento normal del sistema, desactivación de funciones criticas o un mal funcionamiento general. [14]

1.4.4. Vehicle Network Toolbox de MATLAB

Vehicle Network Toolbox, es una herramienta de software que facilita la interacción con redes vehiculares mediante funciones de MATLAB y bloques de Simulink. Permite enviar, recibir, codificar y decodificar mensajes en diversos protocolos como CAN, CAN FD, J1939 y XCP. La identificación y análisis de señales especificas se simplifica gracias al uso de archivos de base de datos CAN estándar, y las señales decodificadas pueden visualizarse en las aplicaciones CAN Explorer y CAN FD Explorer [15]. Vehicle Network Toolbox simplifica la comunicación con redes vehiculares, permitiendo monitorear filtrar y analizar datos del bus CAN en tiempo real, o registrar y reproducir mensajes para análisis posteriores.

1.4.4.1. Canales virtuales de MathWorks

Para facilitar la creación de prototipos de código y la simulación de modelos sin hardware, Vehicle Network Toolbox proporciona un dispositivo CAN virtual MathWorks con dos canales. Estos canales se identifican con el proveedor “MathWorks” y el dispositivo “Virtual 1” [15]. Estos canales virtuales admiten comunicación CAN, CAN FD y J1939, véase Figura 1.10.

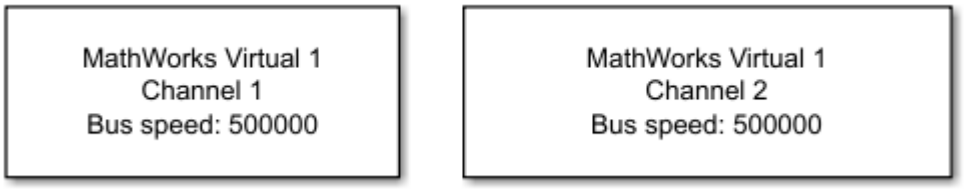


Figura 1.10. Canales Virtuales 1 y 2.

Los dos canales virtuales pertenecen a un dispositivo común, por lo que pueden enviar un mensaje en el canal 1 y recibirlo en el canal 1 y en el canal 2.

1.4.4.2. Limitaciones de los Canales Virtuales

Los canales virtuales, presentan ciertas limitaciones [15]:

- No realizan actividades de protocolo de bajo nivel: esto incluye arbitrajes, frames de error, reconocimiento, etc., que son esenciales en un bus CAN real.
- Comunicación limitada a la sesión de MATLAB: no es posible la comunicación entre diferentes sesiones de Matlab

1.4.4.3. Comunicación en MATLAB

Vehicle Network Toolbox permite la comunicación con redes vehiculares en el entorno de MATLAB [15]. El flujo de trabajo general para transmitir mensajes CAN se puede dividir en los siguientes pasos:

- Creación de un canal CAN: se utiliza la función “canChannel” para crear un canal CAN, especificando el proveedor, el dispositivo y el índice del canal del dispositivo.
- Configuración de propiedades del canal: Las propiedades como la velocidad del bus, se configuran utilizando la función “configBusSpeed”. Si no se especifica la velocidad, se asigna la velocidad predetermina de 500000 bits por segundo.
- Inicio del canal: El canal se inicia utilizando el comando “start”.

- Creación y empaquetado del mensaje: se crea un objeto de mensaje CAN con “canMessage” y se empaqueta con los datos necesarios utilizando “pack”.
- Transmisión del mensaje: El mensaje empaquetado se transite al bus CAN mediante la función “transmit”.
- Recepción del mensaje: Se pude recibir en el mismo canal o en otro Canal, el mensaje se recibe utilizando “receive”.
- Desempaquetado del mensaje: el mensaje recibido se desempaqueta para extraer los datos utilizando “Unpack”.
- Desconexión de los canales: Finalmente, los canales se desconectan utilizando “stop” para finalizar la comunicación

Además de estos pasos, el Vehicle Network Toolbox admite el filtrado de mensajes específicos, mediante las funciones “filterAllowOnly”, “filterAllowAll” y “filterBlockAll”.

1.4.4.4. Comunicación CAN en Simulink

Vehicle Network Toolbox extiende su funcionalidad a Simulink proporcionando bloques para la comunicación CAN [15]. Estos bloques permiten simular el tráfico de mensajes en una red CAN y enviar, recibir mensajes a través del bus CAN.

CAN Configuration, este bloque permite configurar los parámetros de comunicación del bus CAN, como el dispositivo, velocidad de transmisión, véase Figura 1.11. Este bloque asegura que los mensajes se envíen a través de un Canal Virtual del dispositivo “MathWorks Virtual 1”, que emula el bus CAN en el entorno de simulación.

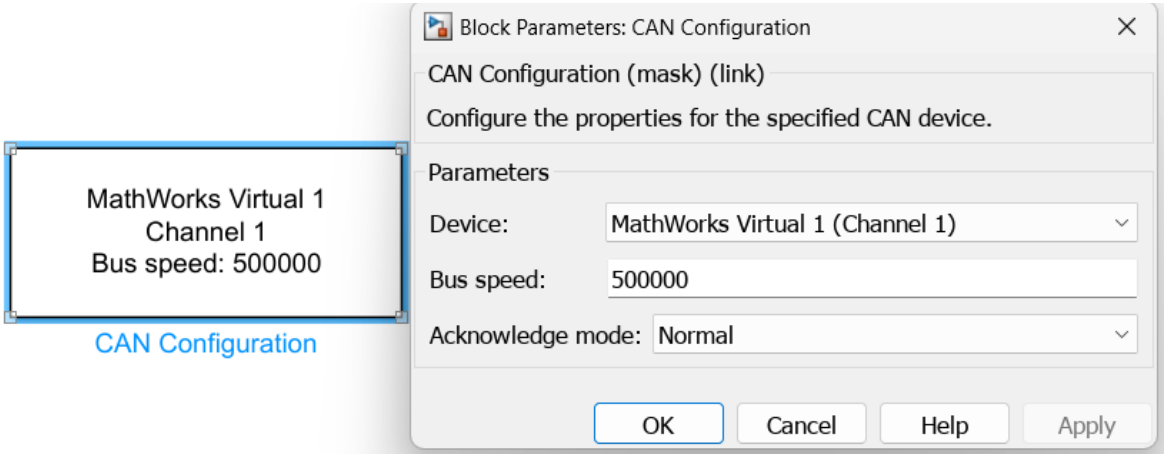


Figura 1.11. Bloque “CAN Configuration”, Simulink.

CAN Pack, este bloque se encarga de codificar señales individuales en un mensaje CAN. Significa que toma los datos de varias señales y los organiza en un formato que pueda ser transmitido a través del bus CAN. El bloque permite diferentes formas de especificar como se realiza este empaquetado, ya sea utilizando datos sin procesar, definiendo manualmente o utilizando las definiciones de un archivo de base de datos CAN (.dbc). Véase Figura 1.12.

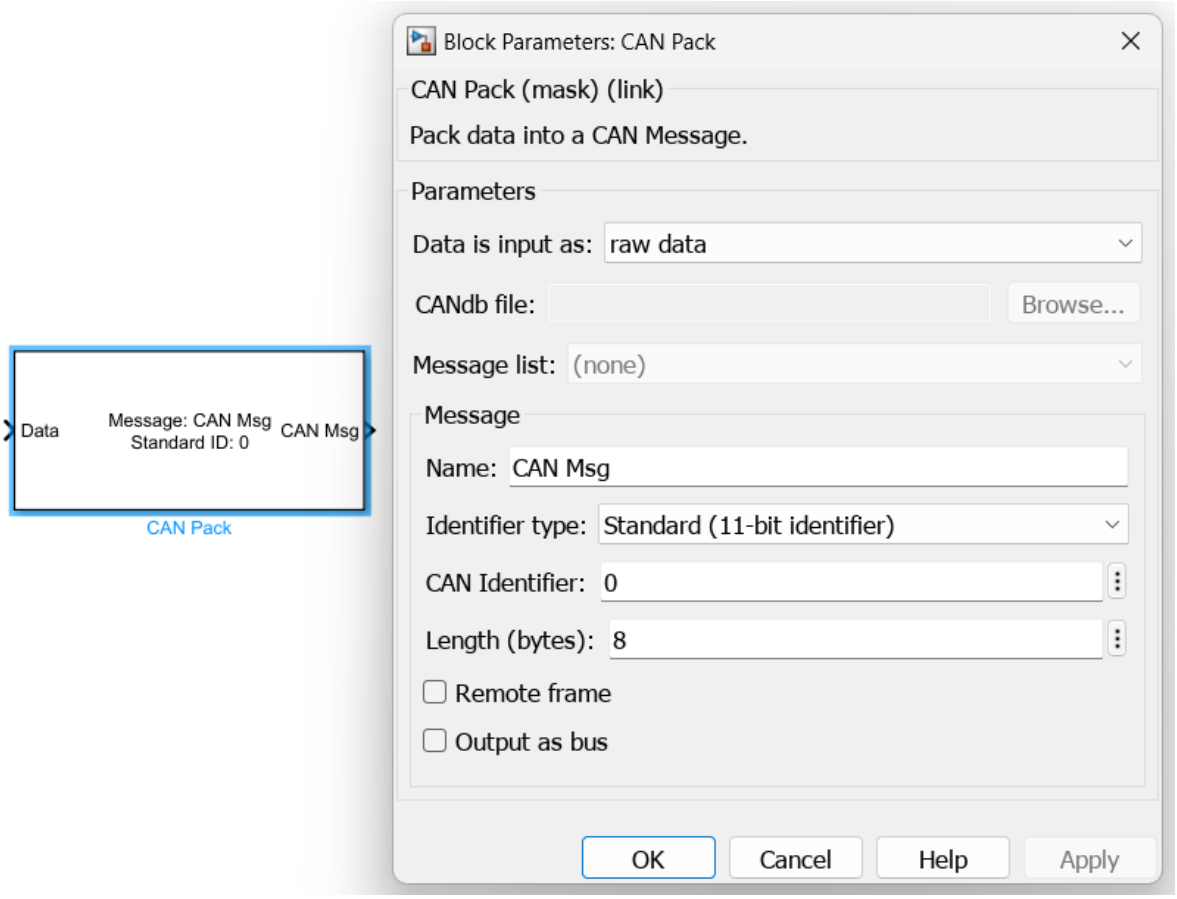


Figura 1.12. Bloque “CAN Pack”, Simulink.

CAN Transmit, envía los mensajes CAN a través el bus, ya sea de forma periódica o en respuesta a eventos específicos. Véase Figura 1.13.

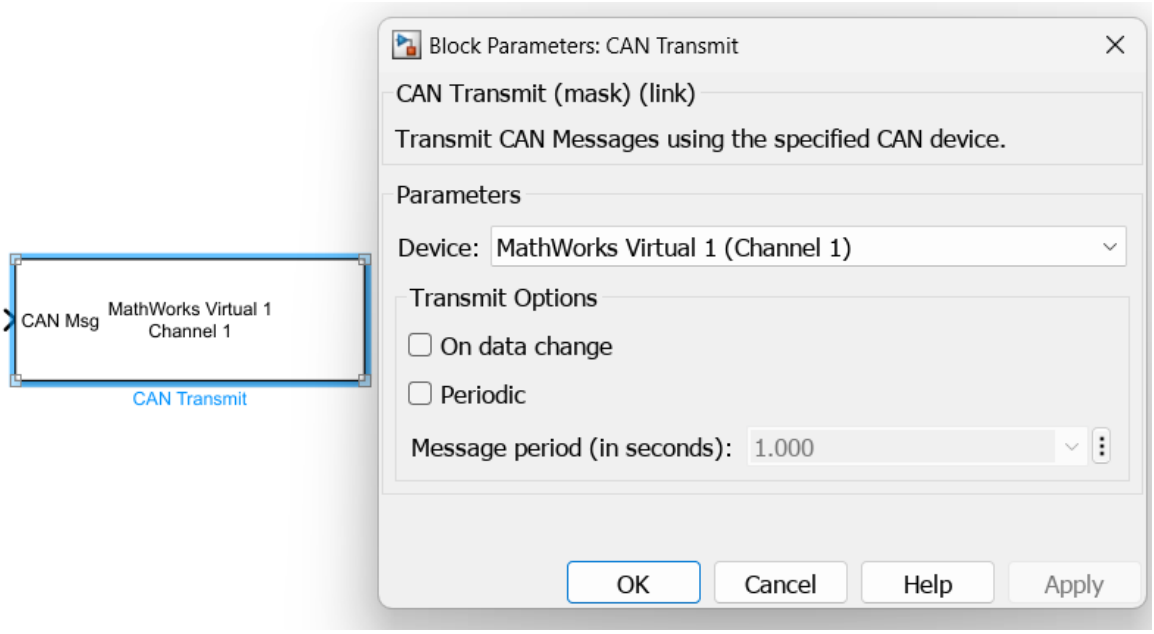


Figura 1.13. Bloque “CAN Transmit”, Simulink.

CAN Receive, este bloque recibe mensajes CAN del bus y los pone a disposición para su procesamiento. Véase Figura 1.14.

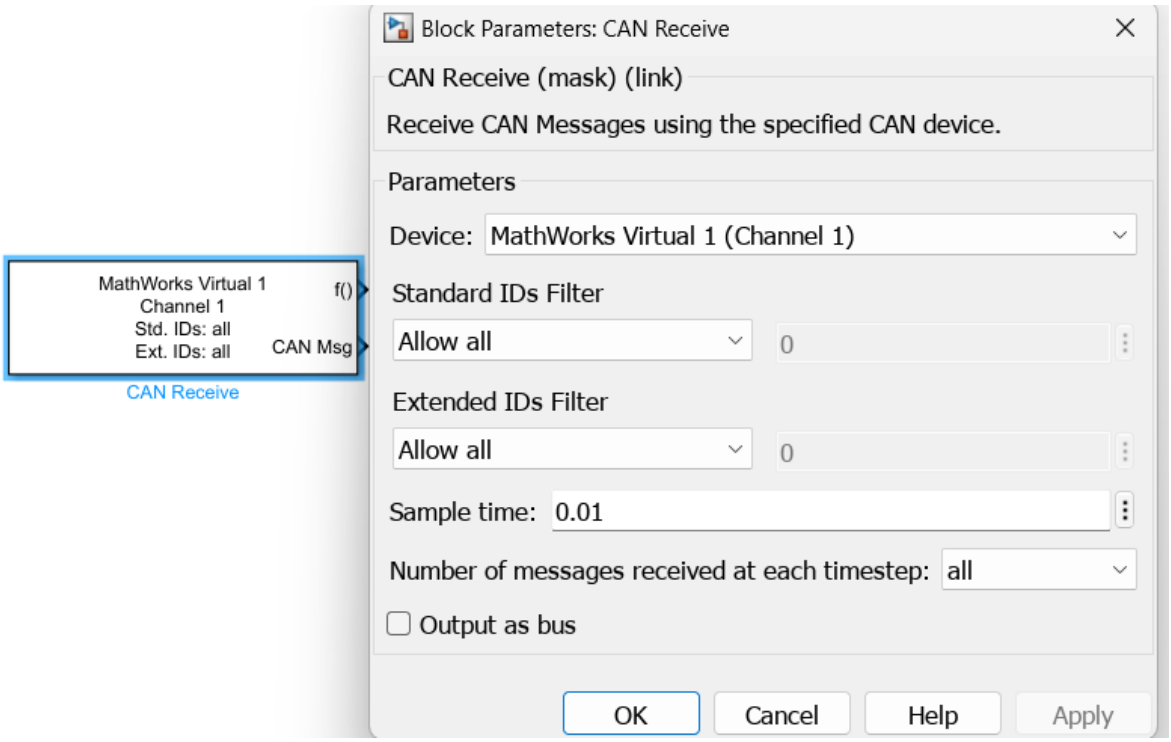


Figura 1.14. Bloque “CAN Recieve”, Simulink.

CAN Unpack, este bloque se encarga de decodificar un mensaje CAN recibido, extrayendo las señales individuales que contiene. Véase Figura 1.15.

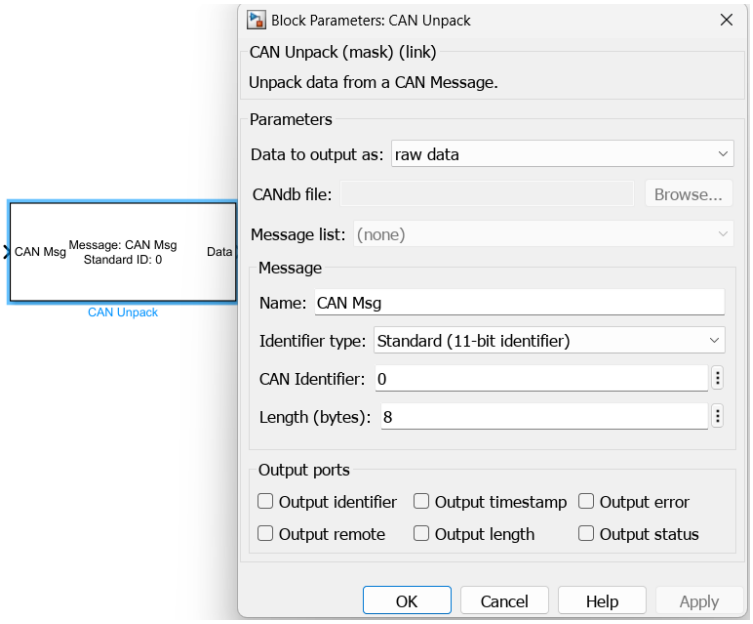


Figura 1.15. Bloque “CAN Unpack”, Simulink.

En el ANEXO A, es una síntesis de la guía de usuario del Vehicle Network Toolbox desarrollada por MathWorks. Este resumen enfatiza en la utilización de la comunicación CAN en MATLAB/Simulink.

1.4.5. Sistema ABS

El sistema de frenos antibloqueo (ABS) es un sistema de seguridad activa esencial en vehículos modernos. Su función principal es prevenir el bloqueo de las ruedas durante el frenado, manteniendo la estabilidad direccional y la capacidad de maniobra del vehículo [16].

1.4.5.1. Componentes

Los componentes principales del sistema ABS incluyen (véase Figura 1.16):

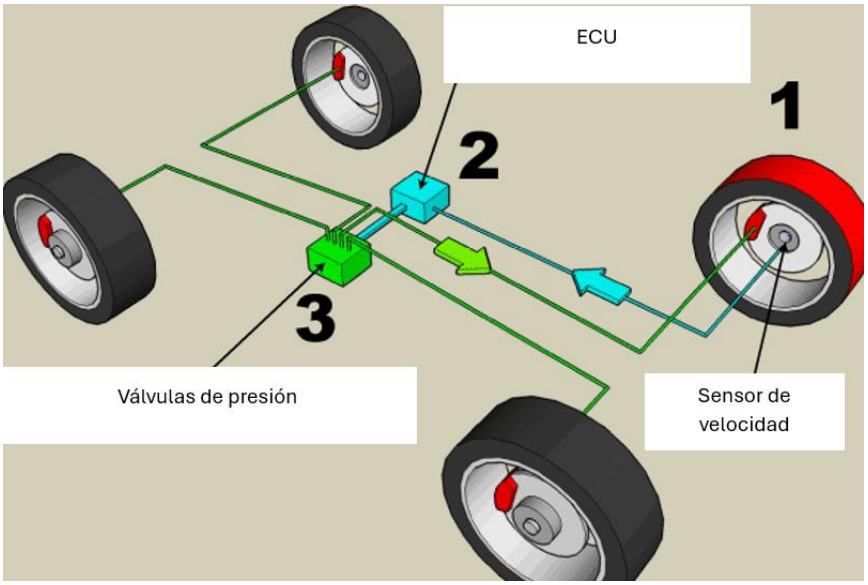


Figura 1.16. Componentes del sistema ABS. [17]

- Sensor de velocidad: Sensores generalmente basados en reluctancia o efecto Hall, miden la velocidad de rotación de cada rueda.
- Unidad de control electrónico: La ECU procesa las señales de los sensores, determina si una rueda esta a punto de bloquearse y activa las válvulas moduladoras de presión [18].
- Válvulas: Estas válvulas, generalmente electrohidráulicas, controlan la presión de frenado en cada rueda de forma independiente.

1.4.6. Sistema de Encendido (Keyless)

Los sistemas de encendido sin uso de llave constan típicamente de un dispositivo portátil (denominado también como llavero inteligente o FOB) que el conductor maneja, sustituyendo las funciones convencionales de una llave metálica. La autenticación del dispositivo se efectúa electrónicamente cuando el conductor inicia el vehículo, que suele realizarse activando un botón [19].

1.4.6.1. Componentes

Los componentes de un sistema simplificado de encendido incluyen (véase Figura 1.17):

- Botón de encendido/apagado: Reemplaza la llave mecánica tradicional.
- Unidad de control electrónico (ECU): Controla el proceso de encendido.

- FOB: Contiene un transmisor que envía señales al vehículo para autenticar al usuario y permitir el encendido.
- Antena: Recibe las señales del llavero inteligente (FOB).
- Motor de Arranque: Su función es convertir la energía eléctrica en energía mecánica para hacer girar el motor del vehículo y ponerlo en marcha.

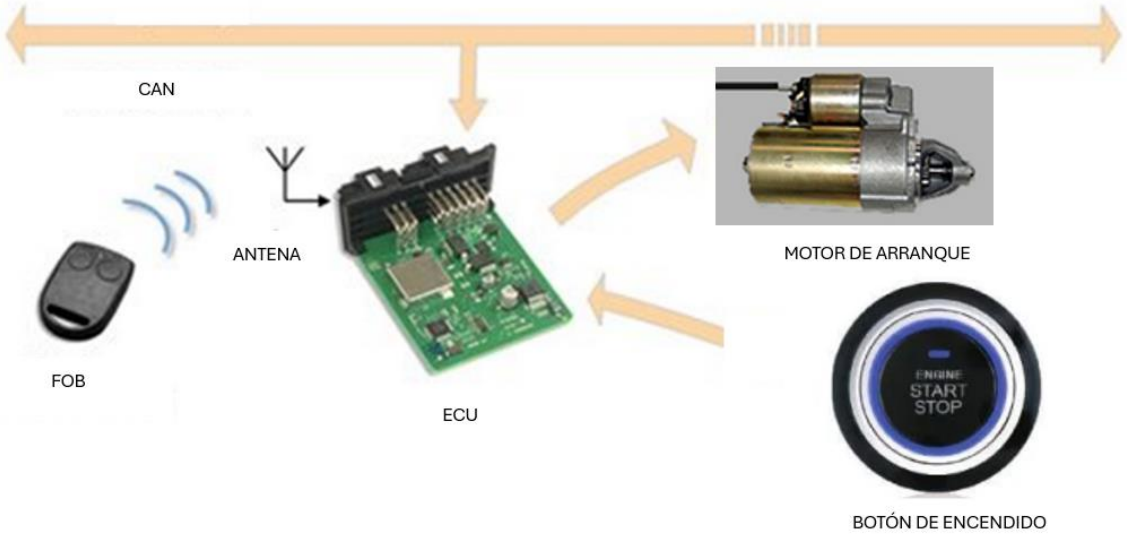


Figura 1.17. Componentes del sistema de Encendido, simplificado.

2. METODOLOGÍA

En esta sección, se detalla el diseño e implementación de ataques de denegación de servicio (DoS) e inyección de datos falsos dirigidos al protocolo CAN, específicamente en dos sistemas vehiculares críticos: el sistema antibloqueo de frenos (ABS) y el sistema de encendido.

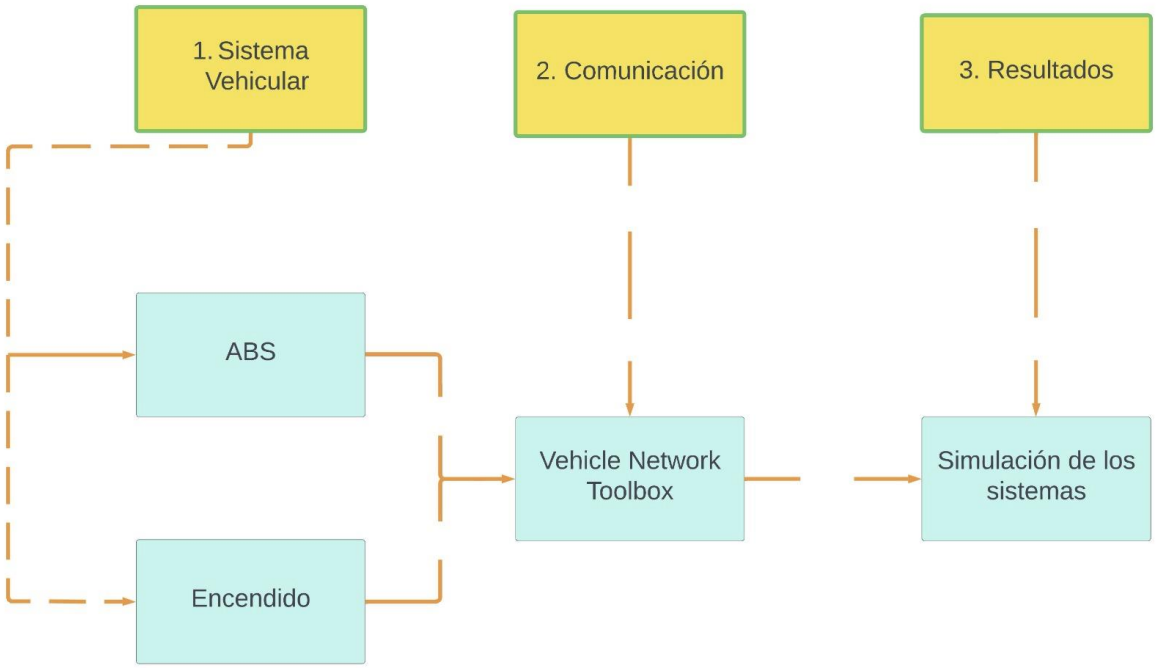


Figura 2.1. Metodología, sistemas en funcionamiento normal.

La metodología propuesta para los sistemas vehiculares en funcionamiento normal, se muestra en la Figura 2.1; la misma que se estructura en tres etapas fundamentales:

1. Diseño e implementación de los sistemas vehiculares en MATLAB: En esta fase inicial, se construye los modelos de los sistemas vehiculares.
2. Implementación de la comunicación CAN: Mediante el Vehicle Network Toolbox de MATLAB, se integra la comunicación CAN a los modelos del sistema vehicular, permitiendo la interacción y el intercambio de datos entre los diferentes componentes.

3. Resultados: Verificación y simulación de los sistemas, se realiza pruebas para validar el correcto comportamiento de los modelos para su posterior análisis.

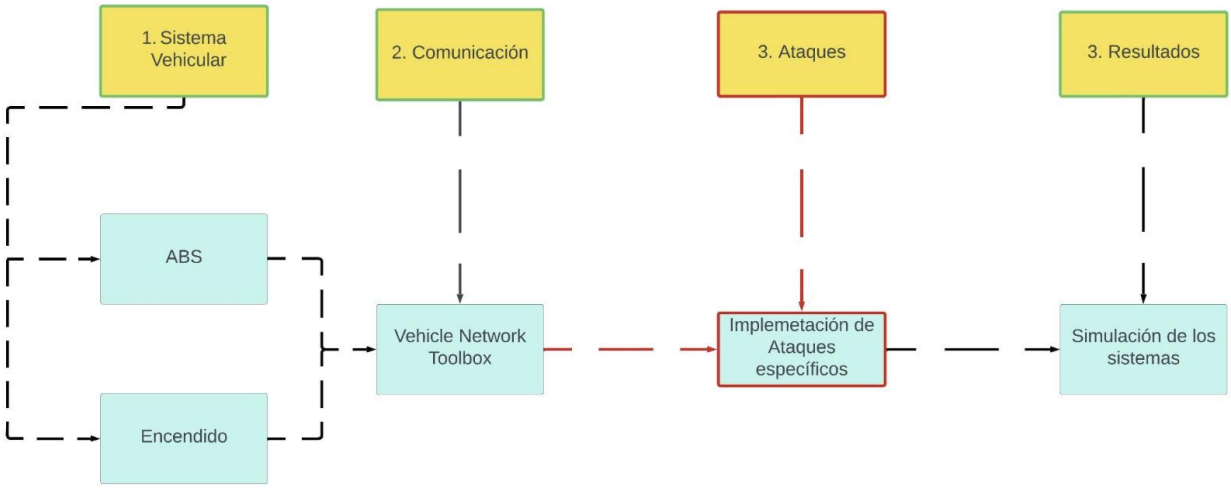


Figura 2.2. Metodología, sistemas en funcionamiento anormal.

En la Figura 2.2, se muestra la metodología propuesta de los sistemas vehiculares en funcionamiento anormal; es decir bajo un ataque, el cual consta de cuatro etapas, de las cuales las dos primeras etapas se derivan del funcionamiento normal.

3. Implementación de ataques: Se implementa ataques específicos dirigidos a los sistemas vehiculares simulados, con el objetivo de observar las posibles consecuencias de estas amenazas.

4. Simulación con ataques y análisis comparativo: Se ejecutan simulaciones del sistema vehicular bajo los ataques implementados. Se compara con los resultados de las simulaciones sin ataques para identificar su impacto.

2.1. Sistema ABS

El sistema de frenos antibloqueo (ABS) que se simula en este trabajo se desarrolla dentro del entorno de MATLAB, utilizando el Vehicle Network Toolbox para emular la comunicación CAN del sistema vehicular. Esta simulación incorpora tres componentes críticos: el sensor, unidad de control, y actuador. La siguiente Figura 2.3, ilustra el flujo general de la simulación.

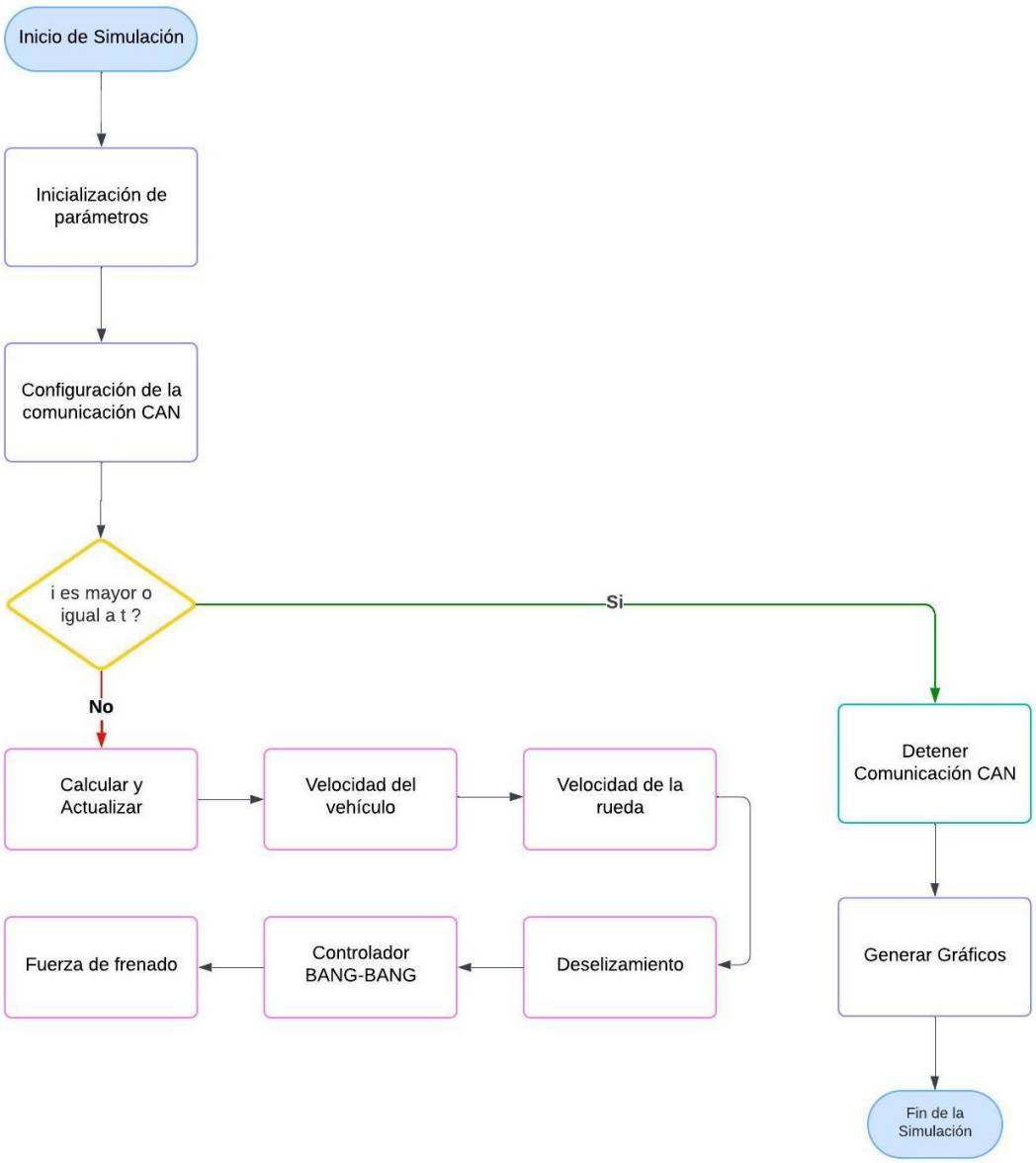


Figura 2.3. Diagrama de flujo, script MATLAB simulación.

Esta simulación consta de un bucle que se ejecuta durante un tiempo definido, calculando y actualizando dinámicamente las variables de la velocidad del vehículo, velocidad de la rueda, deslizamiento y la fuerza de frenado. Además, el bucle incorpora la simulación del sensor, que mide la velocidad de la rueda y añade ruido gaussiano para representar las imperfecciones del mundo real. El controlador, basado en el algoritmo Bang-Bang, utiliza la lectura del sensor para calcular el deslizamiento y ajustar la fuerza de frenado en consecuencia, manteniendo el deslizamiento dentro del rango optimo. El actuador, aplica la fuerza de frenado calculada por el controlador. Finalmente, se calcula el torque de la rueda, la aceleración angular y la nueva velocidad de la rueda, teniendo en cuenta factores como la fricción.

El script de MATLAB en el que se implementa el sistema ABS, y se utiliza el Vehicle Network Toolbox para simular la comunicación en el sistema, se encuentra detallado en el ANEXO B.

2.1.1. Sensor

La ecuación 2, modela la medición de la velocidad de una rueda en un sistema ABS. Esta medición incorpora ruido gaussiano.

$$v_{sensor}(i) = v_{rueda}(i) + (\sigma_{ruido} * randn()) \tag{2.1}$$

Donde:

$V_{rueda}(i)$: Velocidad lineal de la rueda en el instante (i).

σ_{ruido}: Desviación estándar del ruido del sensor, cuantifica la incertidumbre inherente a la medición.

randn(): Números aleatorios con una distribución normal estándar.

2.1.2. Velocidad del vehículo

La ecuación 2.2 describe la desaceleración de un vehículo durante el frenado. Esta ecuación se basa en la ecuación logística que incorpora la velocidad del vehículo, un coeficiente de frenado y un factor de desaceleración.

$$a_{vehiculo}(i) = -\frac{C_f * v_{vehiculo}(i)^2}{1 + k * v_{vehiculo}(i)} \tag{2.2}$$

Donde:

C_f: Coeficiente de frenado

k: Factor de desaceleración

La ecuación 2.3 actualiza la velocidad de vehículo en le instante (i+1) a partir de su velocidad en el instante anterior (i). Esta ecuación modela la evolución de la velocidad del vehículo durante el proceso de frenado. Al considerar la desaceleración calculada en la ecuación 2.2, se obtiene una nueva velocidad que refleja la reducción gradual de la velocidad debido a la acción de los frenos.

$$a_{vehiculo}(i+1) = \max(0, v_{vehiculo}(i) + a_{vehiculo(i)} * \Delta t) \tag{2.3}$$

2.1.3. Deslizamiento

La ecuación 2.4 calcula el deslizamiento de las ruedas en el instante 'i', el deslizamiento es una medida de cuanto patina la rueda con respecto a la superficie de la carretera. El propósito principal de esta ecuación es proporcionar al controlador ABS una medida del deslizamiento de las ruedas.

$$deslizamiento(i) = 1 - \left(\frac{v_{sensor}(i) * R}{v_{vehiculo}(i)}\right) \max\left(0, v_{vehiculo}(i) + a_{vehiculo(i)} * \Delta t\right) \quad (2.4)$$

El controlador ABS utiliza la información del deslizamiento para ajustar la presión de frenado en cada rueda. Si el deslizamiento supera le umbral optimo, el controlador reduce la presión de frenado para evitar el bloqueo. Si el deslizamiento es inferior al umbral optimo, el controlador aumenta la presión del frenado para maximizar la fuerza de frenado.

2.1.4. Controlador BANG-BANG

El controlador Bang-Bang es un algoritmo de control discreto que ajusta la fuerza de frenado en función del deslizamiento de las ruedas. Utiliza una lógica de control basada en umbrales para determinar si la fuerza de frenado debe aumentar o disminuir:

Si el deslizamiento(i) > umbral superior:

$$F_{frenado}(i+1) = \max\left(0, F_{actuador}(i) * 0.95\right) \quad (2.5)$$

Si el deslizamiento(i) < umbral inferior:

$$F_{frenado}(i+1) = \min\left(F_{actuador}(i) * 1.05, \frac{T_{max}}{R}\right) \quad (2.6)$$

Si umbral inferior <= deslizamiento (i) <= umbral superior:

$$F_{frenado}(i+1) = \mathrm{F}_{\mathrm{actuador}}(i) \quad (2.7)$$

El propósito principal del controlador es mantener el deslizamiento de las ruedas dentro de un rango optimo para maximizar la fuerza de frenado y evitar el bloqueo de las ruedas.

2.1.5. Actuador

La ecuación 2.8, modela la fuerza de frenado aplicada por el actuador en le instante (i+1). Esta fuerza depende si el retardo del actuador ha transcurrido o no, ecuación 2.9.

$$F_{actuador}(i+1) = F_{controlador} \quad (2.8)$$

$$F_{actuador}(i+1) = F_{frenado}(1) \quad (2.9)$$

Las ecuación modela el retraso utilizando un condición lógica. Si el retardo del actuador ha pasado, la fuerza aplicada es igual a la fuerza deseada calculada por el controlador. Caso contrario, la fuerza aplicada se mantiene en el valor inicial hasta que el retardo haya transcurrido.

2.1.6. Torque de la rueda

La ecuación 2.10, calcula el torque de frenado aplicado a cada rueda del vehículo. Este torque es esencial para entender como la fuerza de frenado generada por el actuador se traduce en una rotación de las ruedas, y en última instancia, en una desaceleración del vehículo.

$$T_{rueda} = \frac{F_{actuador}(i+1)}{n} * R \tag{2.10}$$

Donde:

n: Número total de ruedas del vehículo

R: Radio del neumático.

El propósito principal de esta ecuación es establecer la fuerza real aplicada a las ruedas, considerando la distribución de la fuerza de frenado entre todas las ruedas. Esta información es crucial para calcular la desaceleración del vehículo y actualizar su velocidad en cada instante de tiempo.

2.1.7. Aceleración angular de la rueda

La ecuación 2.11, describe la dinámica de la rueda bajo la influencia del frenado.

$$\alpha_{rueda} = \frac{-T_{rueda}}{I_{rueda}} * C_{amortiguamiento} * \left(\frac{v_{vehiculo}(i)}{R} - v_{rueda}(i) \right) \tag{2.11}$$

Donde:

$C_{amortiguamiento}$: Coeficiente, controla la fuerza del efecto amortiguamiento

I_{rueda}: Inercia de la rueda

Esta ecuación ayuda a modelar el torque de frenado aplicado por el sistema ABS, afectando la rotación de las ruedas. Al calcular la aceleración angular, podemos determinar cómo cambia la velocidad de rotación de la rueda en respuesta a la fuerza de frenado.

2.1.8. Velocidad de la rueda

La ecuación 2.12, actualiza la velocidad angular de la rueda en el instante (i+1) basándose en su aceleración angular.

$$v_{rueda}(i+1) = \max(0, v_{rueda}(i) + \alpha_{rueda} * \Delta t) \quad (2.12)$$

El propósito principal de esta ecuación es simular el comportamiento dinámico de las ruedas bajo las condiciones de frenado, garantizando que la velocidad de la rueda no exceda la velocidad del vehículo.

2.1.9. Configuración de parámetros

La configuración de los parámetros iniciales de la dinámica del vehículo es fundamental para garantizar que el modelo del sistema ABS refleje un comportamiento real. Esta etapa de la simulación involucra el establecimiento de valores específicos para un conjunto de parámetros claves que influyen directamente en la dinámica vehicular y en el rendimiento del sistema ABS.

Parámetros de simulación:

Se establece un tiempo total de simulación de 5 segundos, y se genera un vector de tiempo "t" con 1000 puntos, lo que da como resultado un tiempo 'dt'. Esto permite realizar los cálculos a lo largo de la simulación, capturando el comportamiento dinámico del sistema.

```
% Parámetros de la simulación
tiempo_simulacion = 15;     % Tiempo de simulación en segundos
t = linspace(0, tiempo_simulacion, 1000); % Vector de tiempo
dt = t(2) - t(1);           % Paso de tiempo
```

Figura 2.4. Parámetros de simulación.

Parámetros del vehículo:

- Masa del vehículo: Se estableció una masa de 1500 Kg, que es un valor promedio representativo de un sedán [20]. La masa del vehículo es un factor determinante en la dinámica de frenado y en la inercia global, influenciando directamente el rendimiento del sistema ABS.
- Aceleración Gravitacional: Es de 9.81 m/s^2, la cual es el valor estándar de aceleración en la Tierra.

- Velocidad inicial: Se seleccionó una velocidad de 35 m/s (126 km/h) para simular condiciones de autopista [21].
- Coeficiente de arrastre aerodinámico: Este coeficiente representa la resistencia al avance que experimenta un vehículo debido a la interacción con el aire. El valor de 0.4 es típico para automóviles modernos.
- Área frontal: Se estableció una superficie frontal de 2.2 metros cuadrados, basados en valores típicos para vehículos convencionales, lo que permite una evaluación de la resistencia aerodinámica que enfrenta el vehículo durante la conducción.
- Coeficiente de Resistencia al Rodamiento: 0.015 es un valor común para neumáticos sobre asfalto, la superficie más común para vehículos.

```
% Parámetros del vehículo
masa = 1500;                % Masa del vehículo (kg)
g = 9.81;                   % Aceleración gravitacional (m/s^2)
velocidad_inicial = 30;     % Velocidad inicial del vehículo (m/s)
coeficiente_arrastre = 0.4; % Coeficiente de arrastre del aire
area_frontal = 2.2;         % Área frontal del vehículo (m^2)
coeficiente_resistencia_rodadura = 0.015; % Coeficiente de resistencia a la rodadura
```

Figura 2.5. Parámetros del vehículo.

Parámetros de las ruedas:

- Radio: Se fijo en 0.4m es un valor típico para neumáticos de vehículos de pasajeros, como sedanes. Utilizar un tamaño de neumático estándar garantiza que la simulación represente condiciones de conducción normales.
- Inercia de la rueda: Se fijo en 0.5 kg.m^2, configuración que afecta la rapidez de respuesta del sistema de frenos y la precisión de la medición del deslizamiento de las ruedas. Este valor es comúnmente utilizado en simulaciones de vehículos para replicar el comportamiento rotacional de ruedas de tamaño mediano.

```
% Parámetros de la rueda
radio_neumatico = 0.3;      % Radio del neumático (m)
inercia_rueda = 0.5;        % Inercia de la rueda (kg*m^2)
num_ruedas = 4;             % Número de ruedas (suponiendo 4)
```

Figura 2.6. Parámetros de la rueda.

Parámetros del sistema ABS:

- Deslizamiento óptimo: un valor de deslizamiento del 20% (0.2) es comúnmente aceptado como el punto óptimo donde el neumático ofrece mejor tracción durante el frenado [22]. Este valor se basa en estudios empíricos que muestran que, a este nivel de deslizamiento, la fuerza de frenado es máxima sin perder el control.
- Umbral Alto de deslizamiento: Al establecer el umbral en 0.25 (25%), el controlador se activa para reducir la fuerza de frenado si el deslizamiento excede este valor, evitando así el bloqueo de las ruedas.
- Umbral Bajo de deslizamiento: Con un umbral de 0.15 (15%), el sistema asegura que el deslizamiento se mantenga dentro del rango optimo ajustando la fuerza de frenado para alcanzar el deslizamiento ideal.

```
% Parámetros del controlador ABS (Bang-Bang)
deslizamiento_optimo = 0.2;         % Relación de deslizamiento óptima
umbral_deslizamiento_alto = 0.25;   % Umbral superior de deslizamiento para liberación de freno
umbral_deslizamiento_bajo = 0.15;   % Umbral inferior de deslizamiento para aplicación de freno
```

Figura 2.7. Parámetros del Controlador ABS.

2.1.10. Configuración Comunicación CAN

Se establece un canal virtual utilizando la interfaz ‘canChannel’ de MathWorks, que permite simular la comunicación CAN en el entorno de MATLAB sin la necesidad de un hardware físico, se detalla en el ANEXO B, junto con el script completo del sistema ABS.

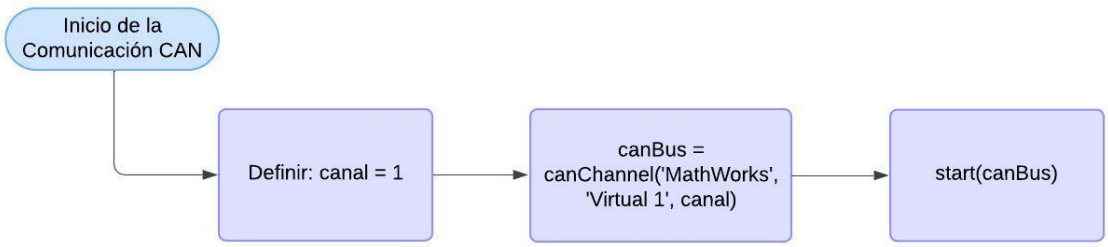


Figura 2.8. Configuración CAN.

Se define identificadores de mensajes para la lectura del sensor y la salida de controlador. Facilitando la transmisión de datos estructurados a través del bus CAN, asegurando que cada mensaje sea correctamente identificado y procesado.

```
% Definir identificadores de mensajes
idMensajeSensor = 1; % Identificador de mensaje de lectura del sensor
idMensajeControlador = 2; % Identificador de mensaje de salida del controlador
longitudMensaje = 8; % Longitud de datos en bytes (máx. 8 para CAN estándar)
```

Figura 2.9. Identificadores.

2.1.11. Gráficos

Al final de la simulación, se generan gráficos que permiten evaluar el comportamiento del sistema ABS. Cada gráfico se centra en una variable clave del sistema, proporcionando información visual sobre como estas cambian durante el frenado. Por ejemplo, el gráfico de velocidad del vehículo, muestra como la velocidad del vehículo cambia a lo largo del tiempo durante la simulación del frenado. Una disminución progresiva en la velocidad indica que el sistema de frenado está actuando sobre el vehículo. Mientras que el gráfico de velocidad de la rueda, representa la velocidad de las ruedas del vehículo durante el frenado. Se compara con la velocidad del vehículo para identificar cualquier deslizamiento. Por otra parte, el gráfico de deslizamiento, indica el deslizamiento de las ruedas en relación con la relación a la velocidad del vehículo. El deslizamiento óptimo se mantiene dentro de un rango que maximiza la tracción y el control del vehículo. Finalmente, el gráfico de fuerza de frenado del actuador, presenta la fuerza (N) de frenado que el actuador aplica a las ruedas.

2.2. Sistema de Encendido

La simulación del sistema de encendido por botón se lleva a cabo en el entorno de MATLAB/Simulink, implementando el Vehicle Network Toolbox para emular la comunicación CAN; esto permite modelar la interacción entre los componentes clave del sistema: Sensor, unidad de control electrónico, y el actuador (motor de arranque). Para lo cual se utilizan bloques específicos del Vehicle Network Toolbox para simular la transmisión y recepción de mensajes CAN. La Figura 2.10, presenta la implementación del sistema de encendido en Simulink.

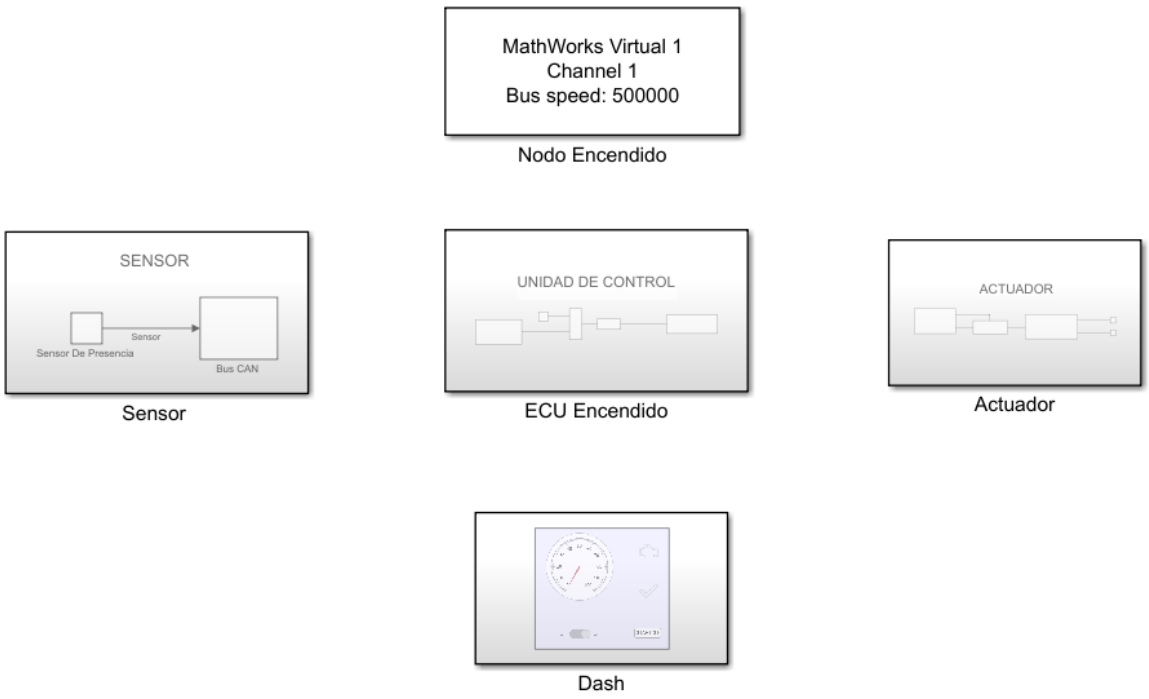


Figura 2.10. Sistema de Encendido, Simulink.

2.2.1. Sensor

El subsistema del sensor de presencia del llavero inteligente (FOB) se implementa en Simulink para simular la detección de la presencia del FOB en el interior del vehículo. Utilizaremos un botón como entrada para representar la detección del FOB, y el sistema genera un mensaje CAN para indicar la presencia del llavero. La Figura 2.11, muestra el subsistema diseñado en Simulink

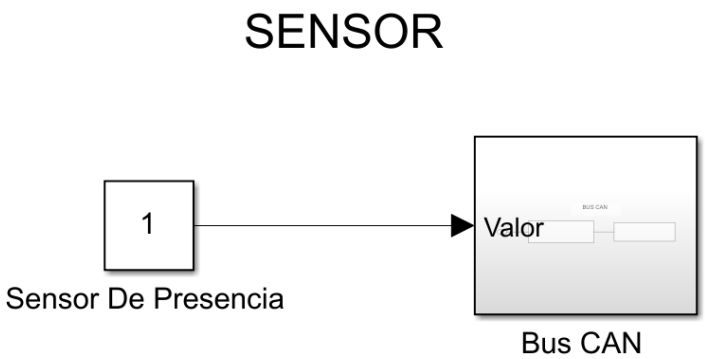


Figura 2.11. Subsistema Sensor proximidad, Simulink.

El bloque “Sensor de Presencia”, es un valor constante que puede tomar el valor de ‘1’ cuando el FOB esta presente, o de ‘0’ cuando el FOB no se encuentra dentro del vehículo.

Este bloque está conectado al bloque ‘Slider Switch’ (Interruptor Deslizante), permite simular de manera interactiva la presencia o ausencia del FOB, durante la simulación. La Figura 2.12, muestra el bloque ‘Bus CAN’, el cual está compuesto por el ‘CAN Pack’, y el bloque ‘CAN Transmit’.

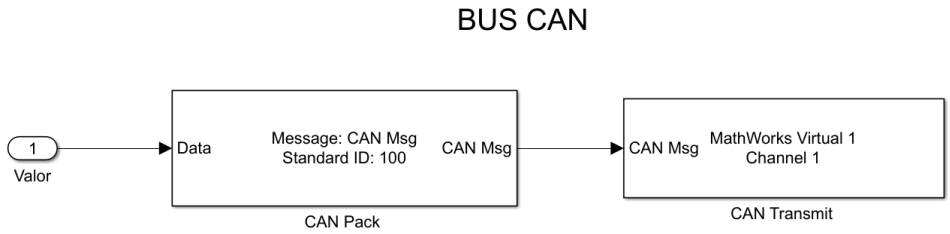


Figura 2.12. Bus CAN, Simulink.

Para configurar el bloque ‘CAN Pack’, se hace Doble-clic en el bloque, se abre el cuadro de diálogo de los parámetros, Figura 2.13. Se establece los siguientes parámetros:

- **‘Data is input as**’: “raw data”.
- **‘Name’:** “CAN Msg” (valor predeterminado)
- **‘Identifier Type’:** “Standard (11-bit identifier)” (valor predeterminado, CAN normal)
- **‘Identifier’:** “100” (Identificador (ID) del valor del sensor)
- **‘Length (bytes)’: “**1”

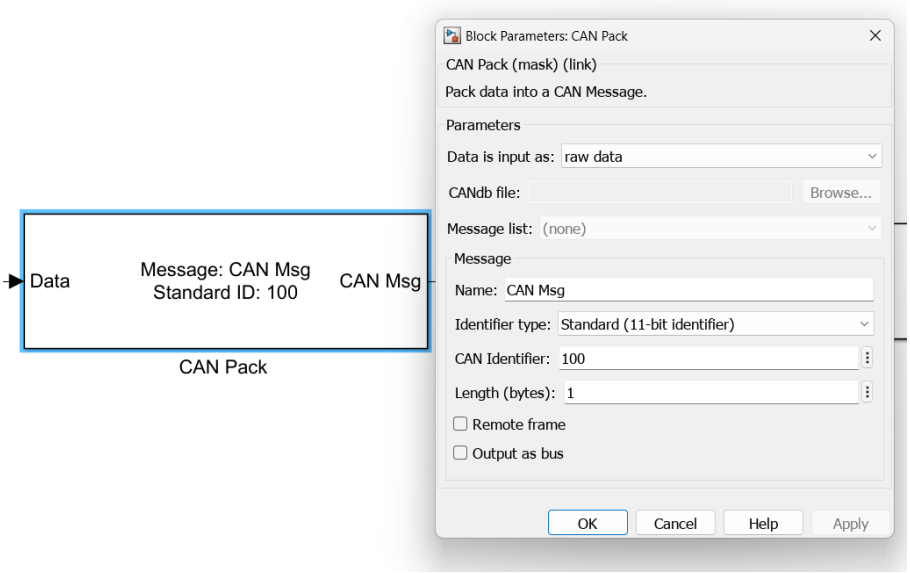


Figura 2.13. Bloque ‘CAN Pack’, Simulink.

Para configurar el bloque “CAN Transmit”, se da doble-clic sobre el bloque para abrir el cuadro de diálogo de los parámetros, Figura 2.14. Verificar que en la sección “**Device**”, este seleccionado “MathWorks Virtual 1 (Channel 1)”.

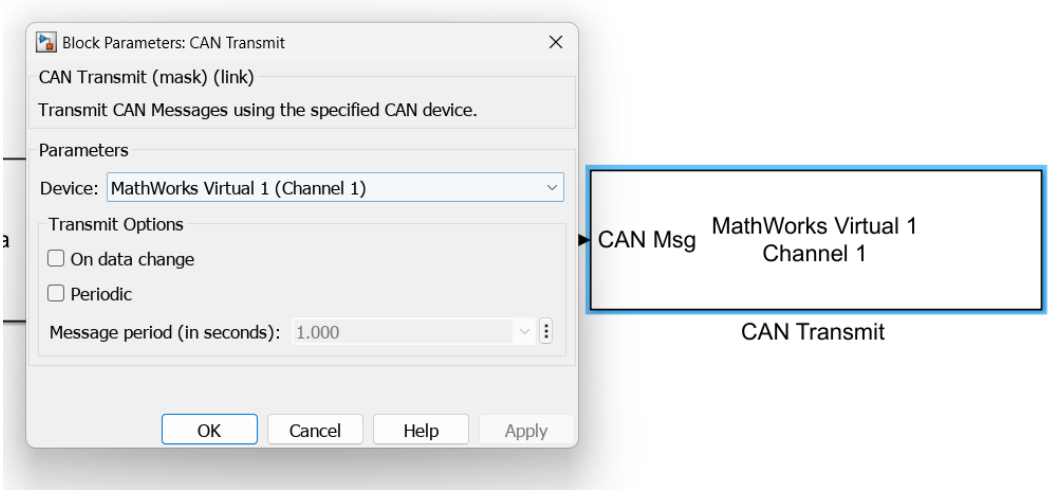


Figura 2.14. Bloque ‘CAN Transmit’, Simulink.

2.2.2. Unidad de Control

La unidad de control (ECU) del sistema de encendido se implementa en Simulink, para gestionar la lógica de encendido y pagado del vehículo en función de la interacción de usuario con el botón de encendido y el mensaje del sensor recibido a través del bus CAN (véase Figura 2.15).

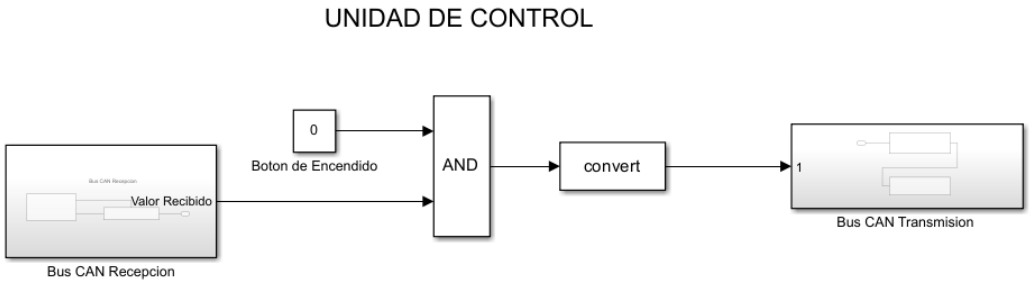


Figura 2.15. Unidad de Control ‘ECU’, Simulink.

El bloque “Botón de Encendido” toman los valores de ‘0’ o ‘1’, y esta enlazado al bloque “Push Button”. Este bloque permite al usuario simular la presión del botón de encendido. El bloque está configurado para mantener el cambio de valor hasta que se vuelva a hacer clic.

El subsistema “Bus CAN Recepción”, está compuesto por lo bloques “CAN Recieve” y “CAN Unpack”, véase Figura 2.16.

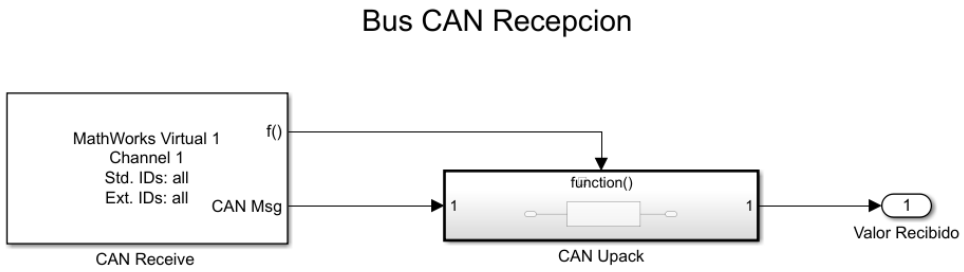


Figura 2.16. Bus CAN Recepción, Simulink.

La configuración del bloque “CAN Recieve” para abrir su cuadro de dialogo de parámetros, se da doble-clic sobre el bloque, véase Figura 2.17. Se verifica y establece los siguientes parámetros:

- **Device:** “MathWorks Virtual 1 (Channel 1)”.
- **Standard IDs Filter:** “Allow all” (valor predefinido)
- **Extended IDs Filter:** “Allow all” (valor predefinido)
- **Sample time:** “0.01”.
- **Number of messages received at each timestep:** “all” (valor predefinido)

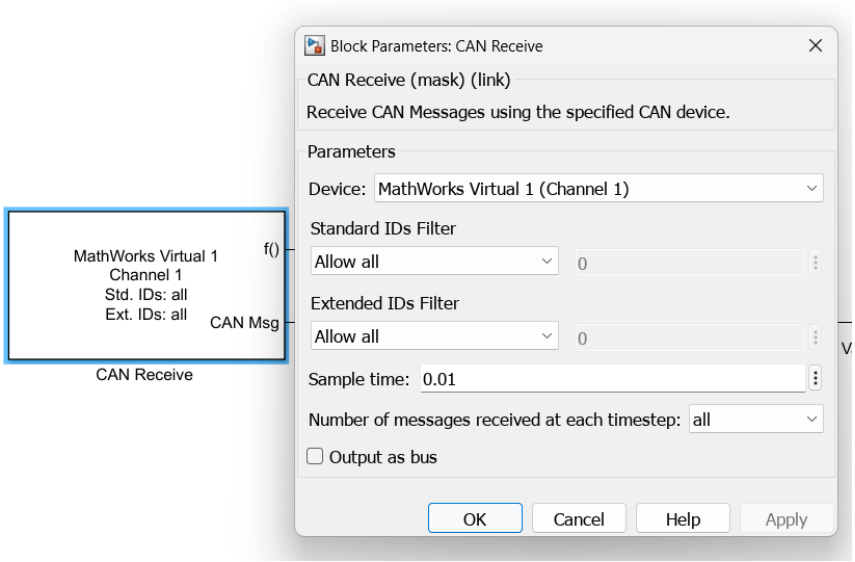


Figura 2.17. Bloque “CAN Recieve”, Simulink.

Configuración del bloque “CAN Unpack”, para abrir el cuadro de dialogo de parámetros, se da doble-clic en el bloque, véase Figura 2.18. Se establece los siguientes parámetros:

- **Data to be output as:** “raw data”
- **Name:** “CAN Msg” (valor predeterminado)
- **Identifier Type:** “Standard (11-bit identifier)” (valor predeterminado)
- **Identifier:** “100” (ID del mensaje desde el sensor)
- **Length(bytes):** “1”

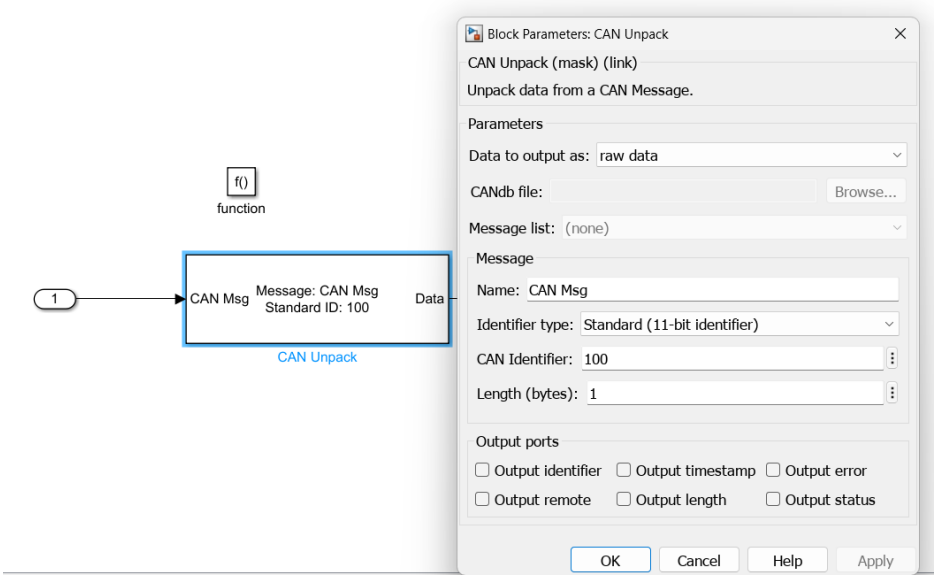


Figura 2.18. Bloque “CAN Unpack”, Simulink.

La lógica de control del sistema de encendido utiliza las señales del botón de encendido y la lectura del sensor de presencia del FOB para determinar si se cumplen las condiciones necesarias para encender el motor. Véase Figura 2.19.

Si (Botón Encendido == 1) && (Sensor Lectura == 1):

Control Encendido = 1

Si (Botón Encendido == 1) && (Sensor Lectura == 0):

Control Encendido = 0

Si (Botón Encendido == 0) && (Sensor Lectura == 1):

Control Encendido = 0

Si (Botón Encendido == 0) && (Sensor Lectura == 0):

Control Encendido = 0

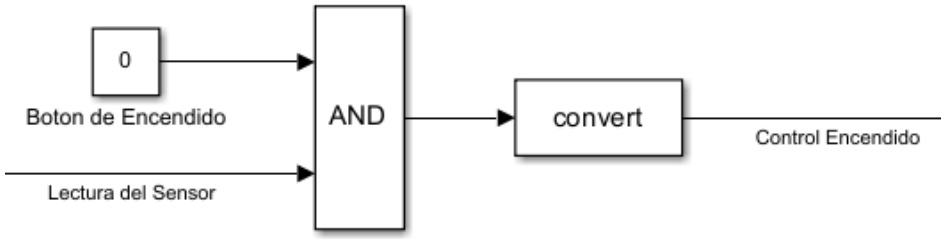


Figura 2.19. Lógica del Controlador, Simulink.

El valor “Control Encendido” de la lógica de control se utiliza como entrada al subsistema “Bus CAN Transmisión”, véase Figura 2.20. De esta manera, la ECU del encendido se comunica con el actuador.

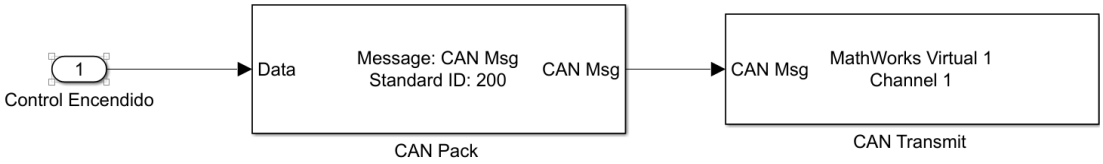


Figura 2.20. Subsistema “Bus CAN Transmisión”, Simulink

En el cuadro de dialogo del bloque “CAN Pack”, se verifica y establece el parámetro del ID:

- **‘Data is input as**’: “raw data”.
- **‘Name’:** “CAN Msg” (valor predeterminado)
- **‘Identifier Type’:** “Standard (11-bit identifier)” (valor predeterminado, CAN normal)
- **‘Identifier’:** “200” (Identificador (ID) del valor del controlador)
- **‘Length (bytes)’: “**1”

Para garantizar la correcta transmisión del mensaje CAN generado por la unidad de control, se abre el cuadro de dialogo del bloque “CAN Transmit”. Se verifica y se establece que la opción “Device” este seleccionado como “MathWorks Virtual 1 (Channel 1)”.

2.2.3. Actuador

El actuador en el sistema de encendido, representado en Simulink por el subsistema "Actuador", véase Figura 2.21, es el encargado de realizar la acción de simular "las revoluciones de motor", una vez recibida la señal de control mediante el bus CAN.

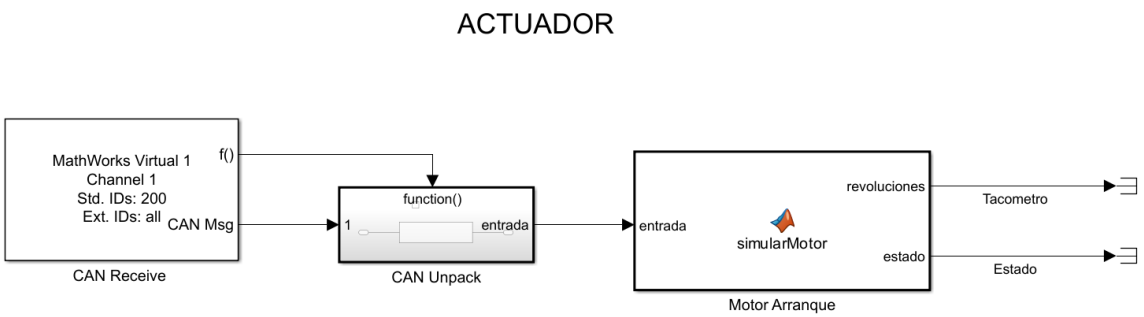


Figura 2.21. Subsistema "Actuador", Simulink.

El subsistema, está compuesto por los bloques "CAN Recieve", "CAN Unpack", y por una función personalizada.

En el cuadro de diálogo del bloque "CAN Recieve", se verifica y se establece los siguientes parámetros para recibir correctamente el mensaje del actuador:

- **Device:** "MathWorks Virtual 1 (Channel 1)".
- **Standard IDs Filter:** "Allow only", "200" (ID del mensaje del controlador)
- **Extended IDs Filter:** "Allow all" (valor predefinido)
- **Sample time:** "0.01".
- **Number of messages received at each timestep:** "all" (valor predefinido)

En el bloque "CAN Unpack", se procede a verificar y establecer los siguientes parámetros:

- **Data to be output as:** "raw data"
- **Name:** "CAN Msg" (valor predeterminado)
- **Identifier Type:** "Standard (11-bit identifier)" (valor predeterminado)
- **Identifier:** "200" (ID del mensaje desde el controlador)
- **Length(bytes):** "1"

El bloque de “MATLAB Function” llamado “Motor Arranque”, tiene como objetivo simular el comportamiento de un motor en términos de sus revoluciones y estado. Generando las señales de salida ‘tacómetro’ y ‘estado’. El script de esta función, que se detalla en el ANEXO C, responde al valor enviado por la unidad de control, reflejando el estado de encendido, apagado y aceleración del motor, véase Figura 2.22.

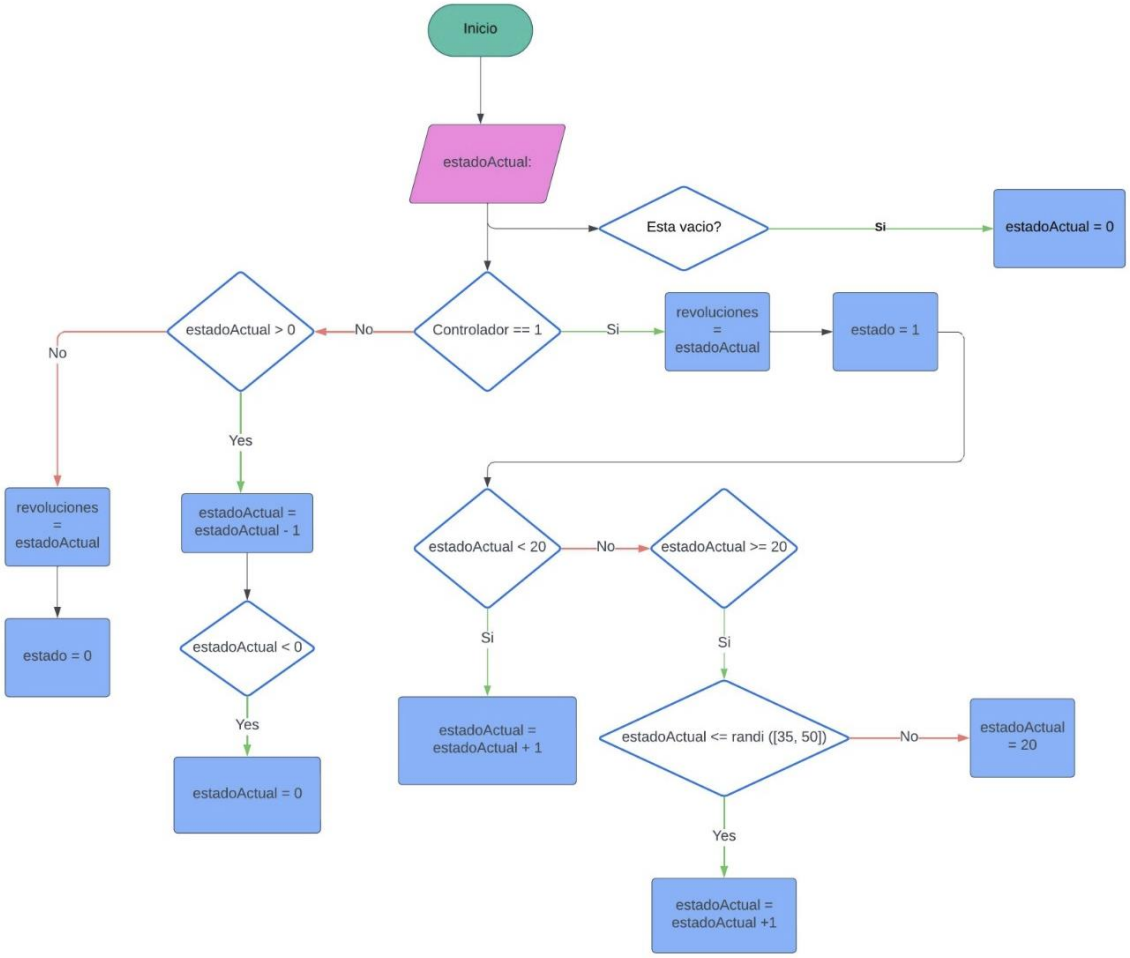


Figura 2.22. Diagrama de flujo “MATLAB Function”, Simulink.

2.2.4. Tablero

El tablero de instrumentos implementado en Simulink proporciona una interfaz gráfica intuitiva para monitorear y controlar el sistema de encendido simulado. El tacómetro muestra las revoluciones una vez encendido, mientras que los indicadores luminosos señalan el estado de encendido del motor y la presencia del FOB. El interruptor deslizante y el botón de encendido permiten al usuario interactuar con la simulación. Véase Figura 2.23.

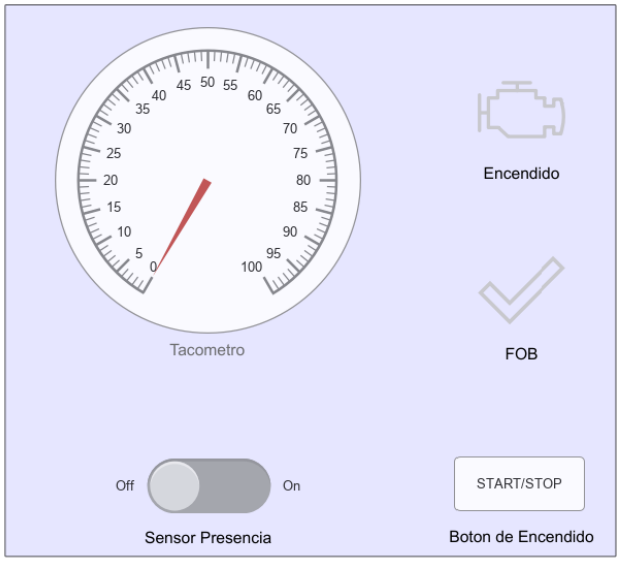


Figura 2.23. Tablero, Simulink.

Se implemento el bloque “Gauge”, el cual actúa de tacómetro. Este bloque esta enlazado a la señal “Tacómetro”, la cual es generada en el subsistema “Actuador”. Véase Figura 2.24, definiendo los siguientes parámetros:

- **Minimum:** “0”
- **Maximum:** “100”
- **Tick Interval:** “Auto” (valor predeterminado)

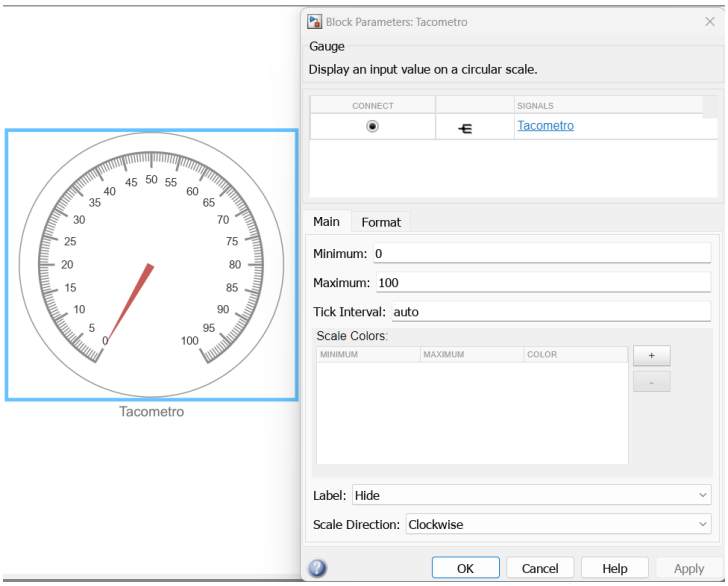


Figura 2.24. Tablero, Tacómetro, Simulink.

El indicador de encendido, se implemento utilizando el bloque “Lamp”, el cual esta conectado a la señal "Estado" generada por el subsistema "Actuador". Esta señal controla directamente el comportamiento del indicador, el cual se ilumina en verde cuando el valor de la señal es 1 (encendido) y en rojo cuando es 0 (apagado). Esta implementación proporciona una representación visual clara y efectiva del estado operativo del sistema, véase la Figura 2.25.

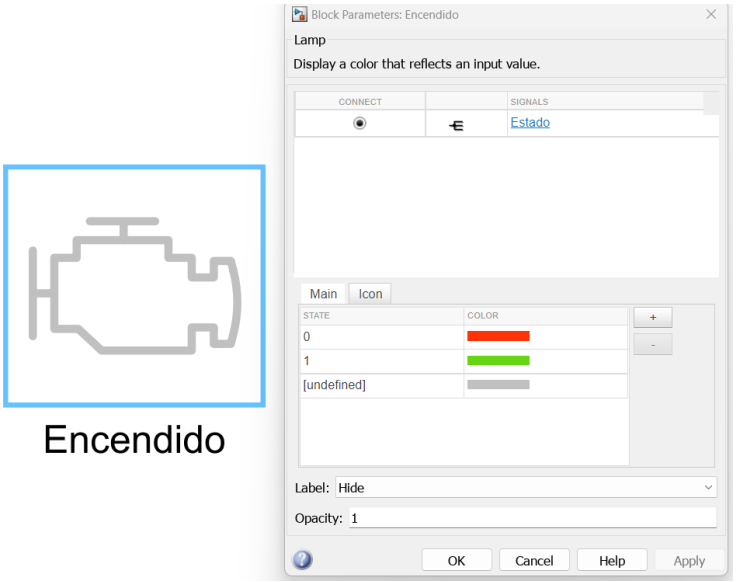


Figura 2.25. Tablero, Indicador de Encendido, Simulink.

El indicador de presencia del FOB, implementado con el bloque “Lamp”, cambia de color según la señal recibida del subsistema “ECU Encendido”. El color rojo indica la ausencia del FOB (valor de 0), mientras que el verde señala su presencia (valor de 1). La Figura 2.26 ilustra la conexión de la señal al bloque.

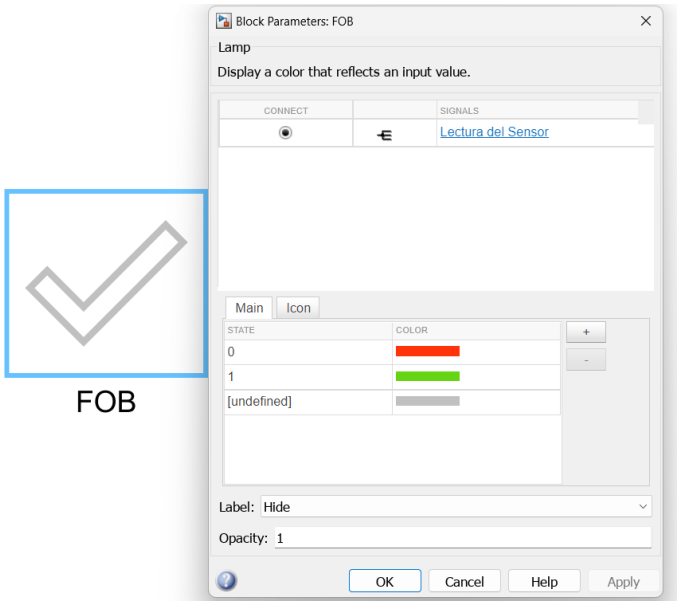


Figura 2.26. Tablero, Indicador del Sensor, Simulink.

El bloque “Slider Switch”, simula un sensor de proximidad. Este bloque genera dos valores ‘0’ (ausencia del FOB) y ‘1’ (presencia del FOB) según su posición: izquierda para ‘0’ y derecha para ‘1’. Estos valores son escritos en el bloque “Sensor De Presencia”, véase Figura 2.27

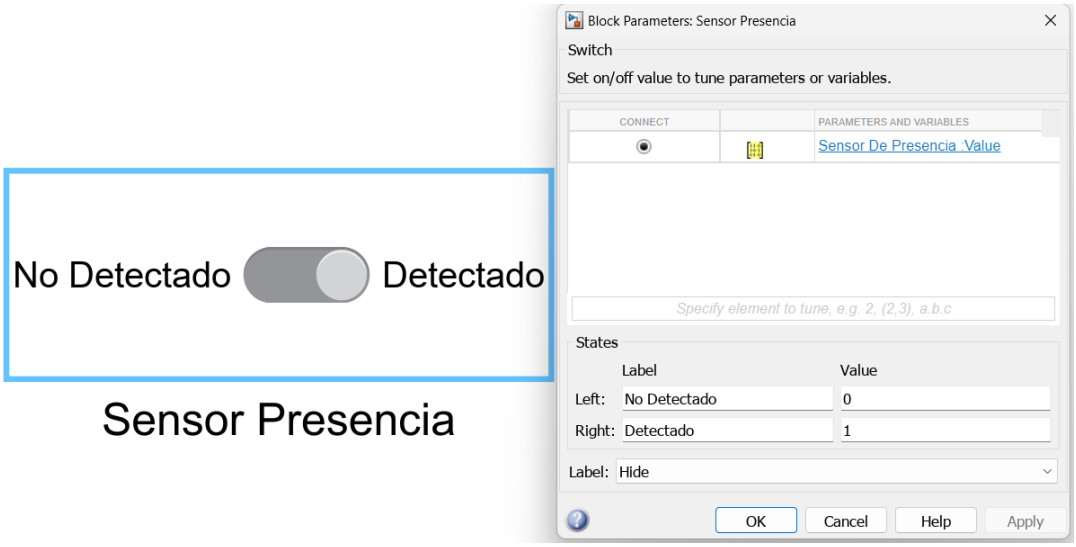


Figura 2.27. Tablero, Sensor, Simulink.

2.2.5. Comunicación CAN

Finalmente, se agrega el bloque “CAN Configuration”, para configura la comunicación, véase Figura 2.28. En el cuadro de diálogo se establece y verifica los siguientes parámetros:

- **Device:** “MathWorks Virtual 1 (Channel 1)
- **Bus speed:** ‘500000’
- **Acknowledge mode:** ‘Normal’

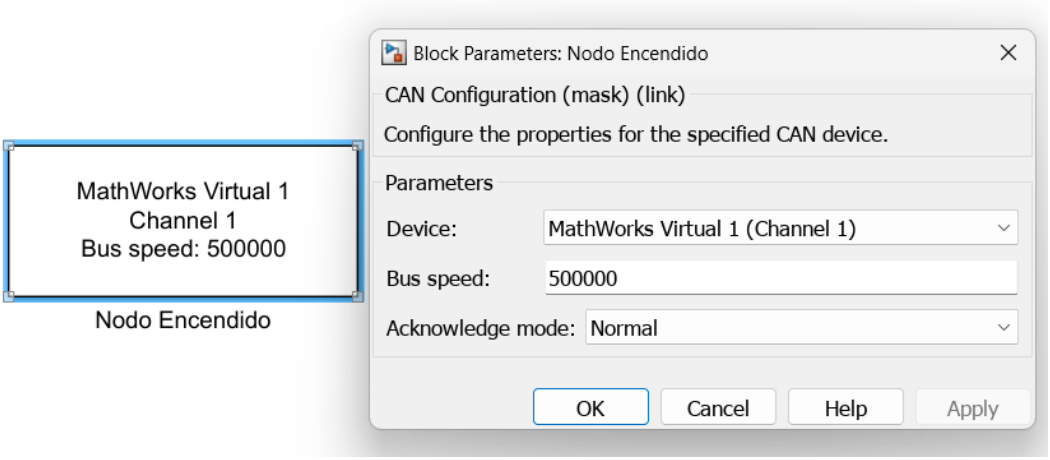


Figura 2.28. Parámetros, CAN, Simulink.

2.3. Ataques

Mediante la implementación de dos tipos de ataques dirigidos a la red de comunicación CAN en el entorno de simulación permite analizar sus efectos sobre el funcionamiento de los sistemas.

2.3.1. DoS

Se lleva a cabo un ataque de denegación de servicio (DoS) dirigido al sistema ABS. En el script desarrollado para la simulación del sistema ABS, se implementa la lógica que activara el ataque al alcanzar un determinado tiempo de simulación. Este ataque se realiza mediante la inundación de la red con mensajes falsificados (véase Figura 2.29). Esta sobrecarga en el bus CAN interfiere con la transmisión de mensajes legítimos, comprometiendo el funcionamiento del sistema ABS. Para detalles sobre el script utilizado, consulte el ANEXO D.

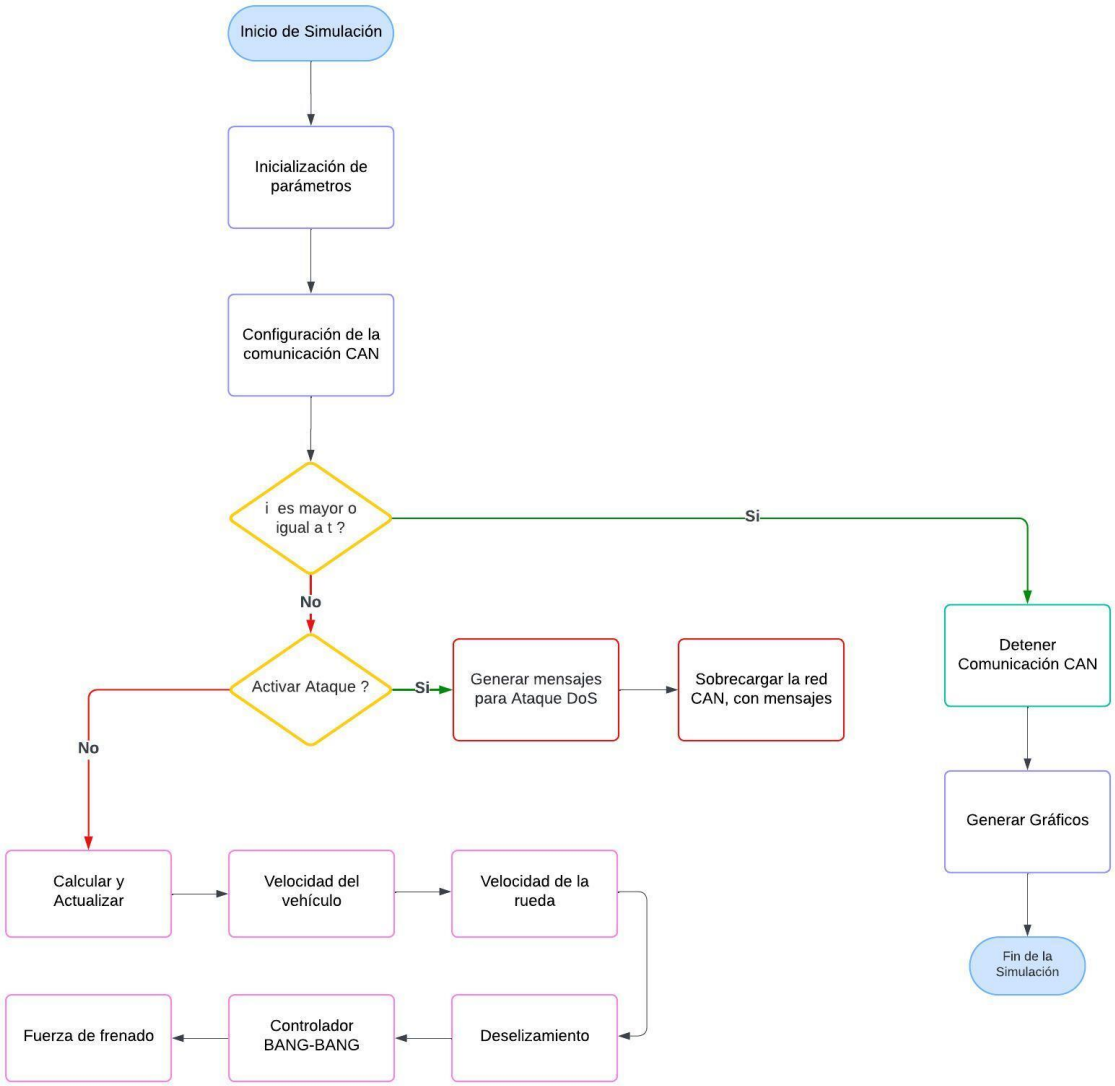


Figura 2.29. Diagrama de flujo, Ataque DoS, Sistema ABS.

Una variable ‘activarAtaque’ determina cuando se inicia y finaliza el ataque, configurando las variables ‘tiempoInicioAtaque’ y ‘tiempoFinAtaque’ para delimitar temporalmente el ataque dentro de la simulación. La variable ‘flooding’ esta implementada como condición del bucle ‘for’ para generar mensajes e incrementar los mensajes durante el ataque DoS, véase Figura 2.30.

```
% Parámetros del ataque DoS
activarAtaque = true; % Activa o desactiva el ataque DoS
tiempoInicioAtaque = 2; % Tiempo en segundos para iniciar el ataque
tiempoFinAtaque = 3; % Tiempo en segundos para terminar el ataque
flooding = 400; % Frecuencia de mensajes durante el ataque
```

Figura 2.30. Parámetros, Ataque DoS, Sistema ABS.

Para simular el ataque de denegación de servicio (DoS) en el sistema ABS, se implementa un bucle ‘for’ que inyecta datos falsos en las lecturas del sensor de velocidad de la rueda. Dentro de este bucle, el valor legítimo de la variable ‘lecturaSensorRecibida’ se multiplica por un factor aleatorio generado mediante la función ‘randi([1,10])’, lo cual altera de forma intencionada la información transmitida. Los valores de la función pueden ser ajustados en cualquier rango, ya que el objetivo principal es simular datos falsificados a partir de datos legítimos, como se muestra en la Figura 2.31.

```
%% Ataque DoS agregar un for loop
if activarAtaque && t(i) > tiempoInicioAtaque && t(i) < tiempoFinAtaque
    % Simular un ataque DoS enviando datos de sensor falsos
    for k = 1:flooding
        lecturaSensorRecibida = lecturaSensorRecibida * randi([1, 10]); % Mensajes falsos

        mensajes_transmitidos(i+1) = mensajes_transmitidos(i) + 1;
    end
end
```

Figura 2.31. Ataque DoS, Sistema ABS.

Los mensajes alterados son empaquetados y transmitidos a través del bus CAN usando el identificador original del sensor, el cual fue asignado en la variable ‘idMensajeSensor = 1’

2.3.2. Inyección de Datos

El ataque de inyección de datos tiene como objetivo de engañar al sistema de encendido para que crea que el FOB esta presente y permita el encendido del motor, incluso cuando el FOB no se Encuentra realmente dentro del vehículo, este ataque será implementado utilizando Simulink en el modelo desarrollo del Sistema de Encendido. Agregado subsistema llamado “Ataque de inyección”, véase Figura 2.32, y un bloque de visualización y un interruptor en el subsistema “DASH”, véase Figura 2.33.

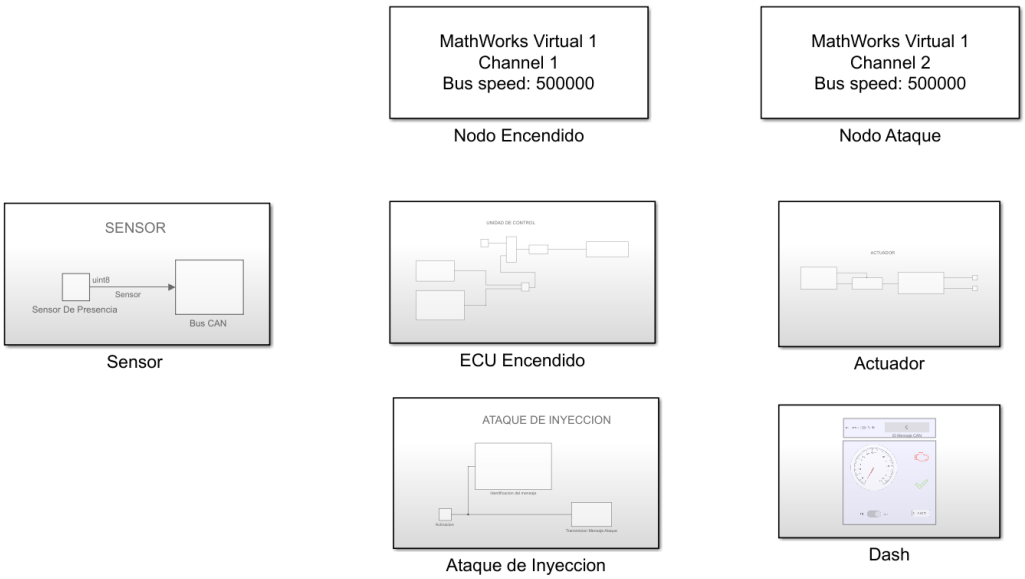


Figura 2.32. Sistema de Encendido con Ataque.

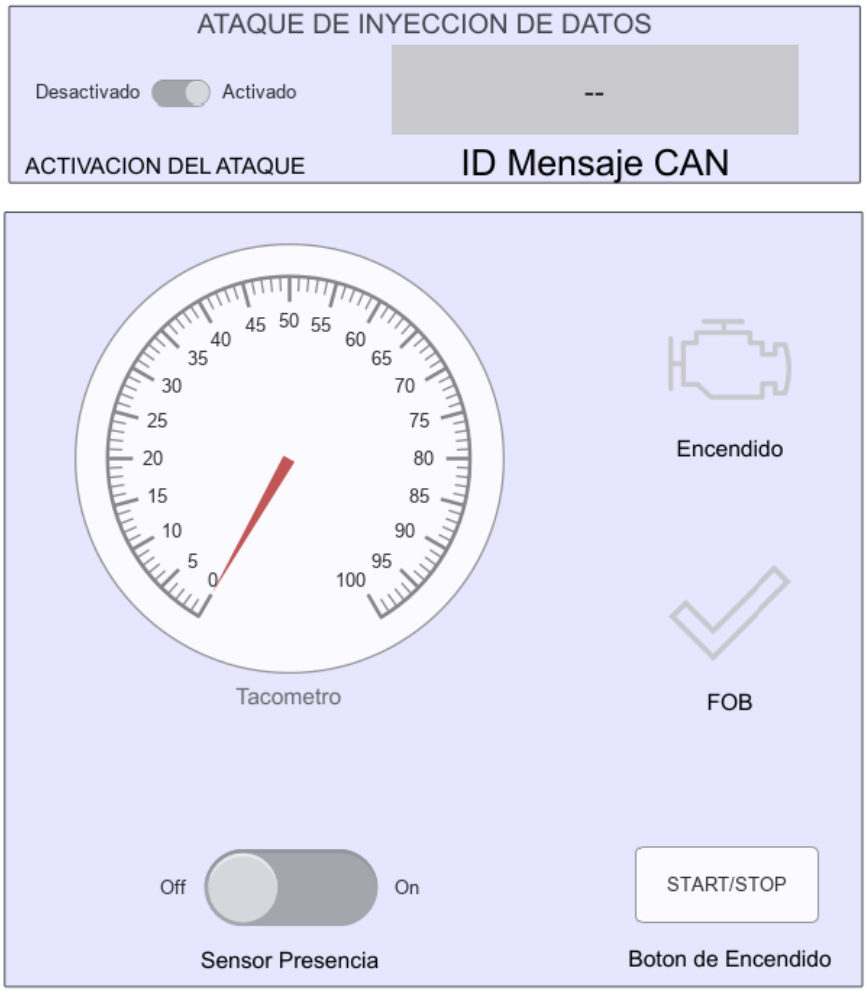


Figura 2.33 Tablero, Sistema de Encendido con Ataque.

Para implementar el ataque se determina el identificador usado en el mensaje CAN para transmitir el estado del sensor. Este mensaje será el objetivo del ataque.

Mediante el uso del bloque “Slider Switch”, el cual activa o desactiva el ataque. Envía el valor de ‘0’ cuando está en la posición izquierda, indicado que el ataque esta desactivado, y cuando se encuentra en la posición de la derecha transmite un valor de ‘1’, asociado cuando el ataque está en funcionamiento. La señal producida por este interruptor está conectado al bloque “Inyección Data”, véase Figura 2.34.

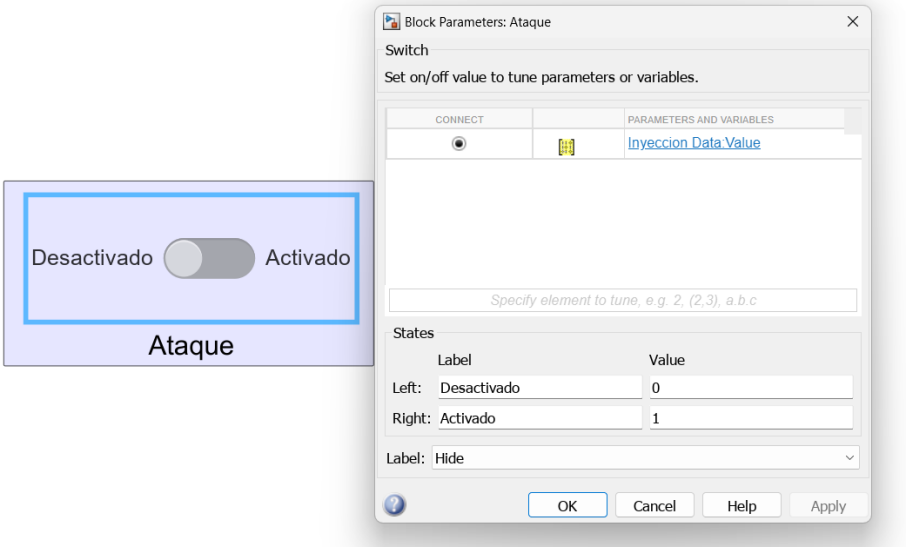


Figura 2.34. Interruptor, Ataque Inyección de datos, Sistema Encendido.

Con la señal de activación del ataque en el subsistema “Inyección de Ataque”, se procede a leer el identificador del mensaje CAN generado por el subsistema “Sensor”. Este valor está asociado al bloque “Display”, el cual muestra el ID del mensaje CAN, véase Figura 2.35.

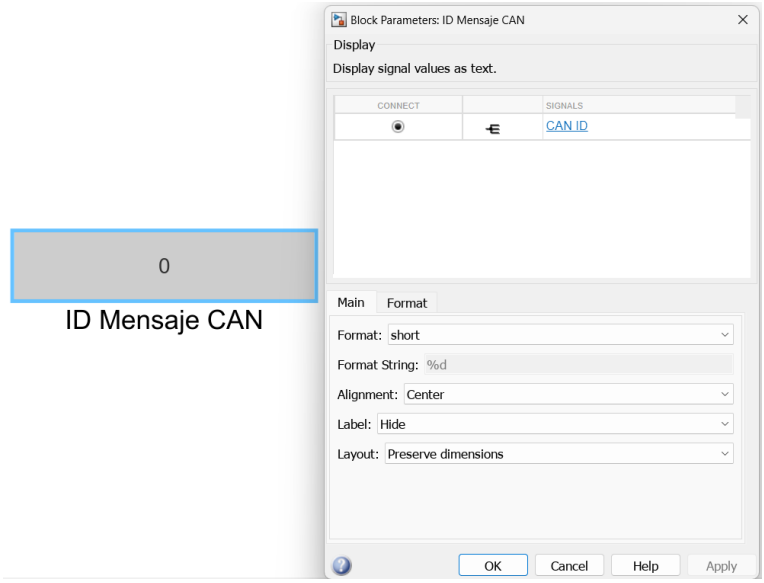


Figura 2.35. Visualización, CAN ID, Sistema Encendido.

Para leer el ID del mensaje se implementó los bloques “CAN Recieve” y “CAN Unpack”. En el cuadro de dialogo del bloque “CAN Recieve”, se establece en la opción “Device” se selecciona “MathWorks Virtual 1 (Channel 1)”.

En el cuadro de dialogo de parámetros del bloque “CAN Unpack”, se selecciona la opción “Output Identifier” para extraer el ID del mensaje CAN, véase Figura 2.36.

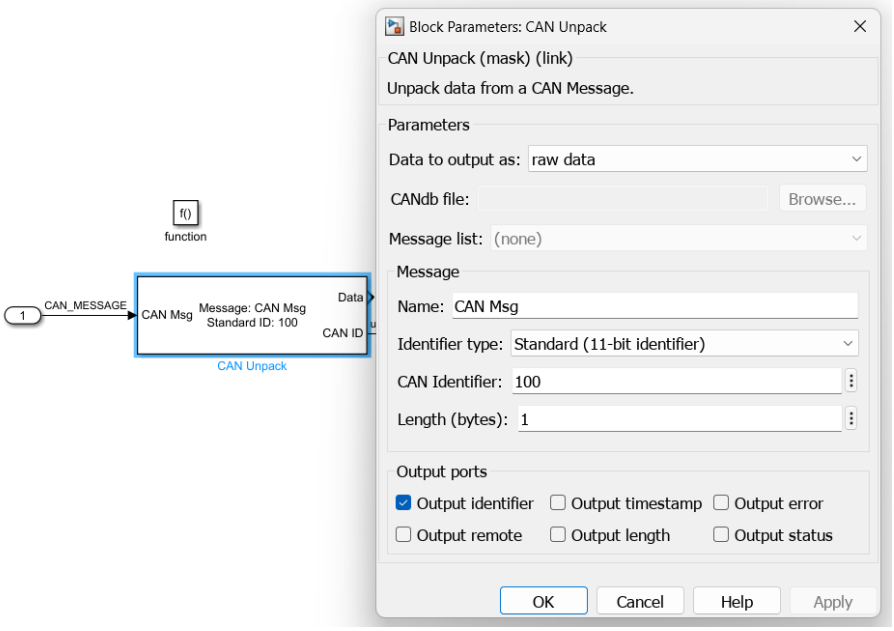


Figura 2.36. Bloque “CAN Unpack”, CAN ID, Sistema Encendido.

El bloque “MATLAB Function” implementa la lógica que procesa la señal de “Activación” del y el valor “ID”, como se muestra en la Figura 2.37. El código detallado de esta función se encuentra en el ANEXO E:

Si Activación == 1:

Valor = ID

Si Activación == 0:

Valor = 0:

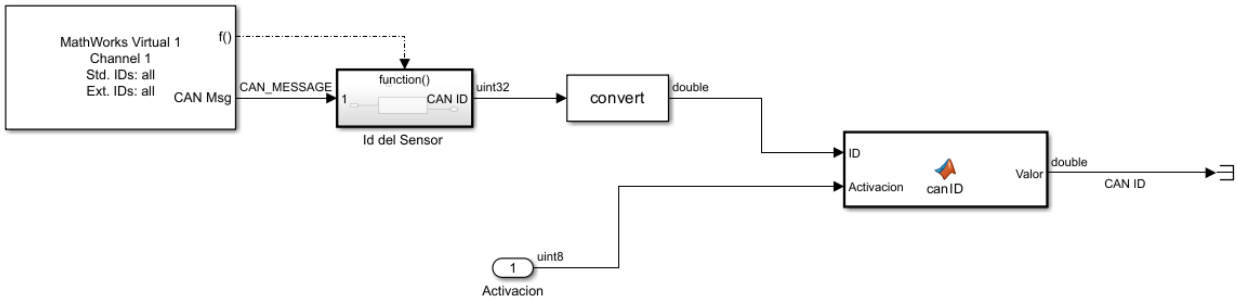


Figura 2.37. CAN ID, Sistema Encendido.

Para la transmisión del valor al subsistema “ECU Encendido”, se realiza mediante la implementación de los bloques “CAN Pack” y “CAN Transmit”, véase Figura 2.38.

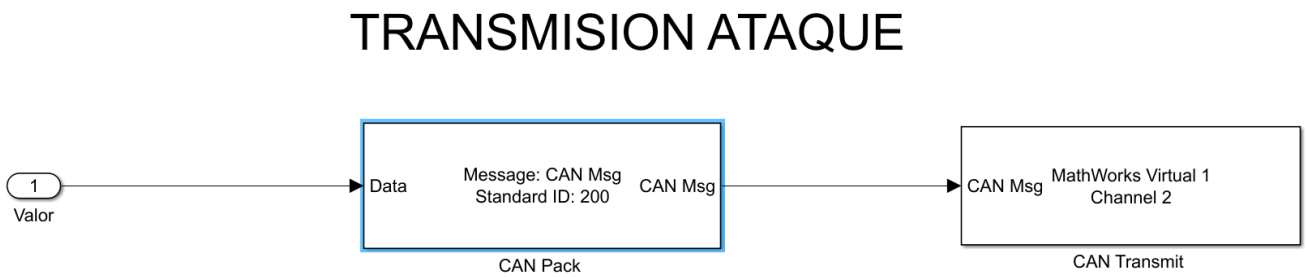


Figura 2.38. Ataque, Comunicación, Sistema Encendido.

En el cuadro de dialogo del bloque “CAN Pack”, se verifica y establece los siguientes parámetros:

- **‘Data is input as**’: “raw data”.
- **‘Name’:** “CAN Msg” (valor predeterminado)
- **‘Identifier Type’:** “Standard (11-bit identifier)” (valor predeterminado, CAN normal)
- **‘Identifier’:** “200” (Identificador (ID) del mensaje a suplantar)

- **‘Length (bytes)’: “**1”

En el bloque “CAN Transmit”, se verifica y establece en el cuadro de dialogo, que la sección “**Device**”, este seleccionado “MathWorks Virtual 1 (Channel 2)”, véase Figura 2.39.

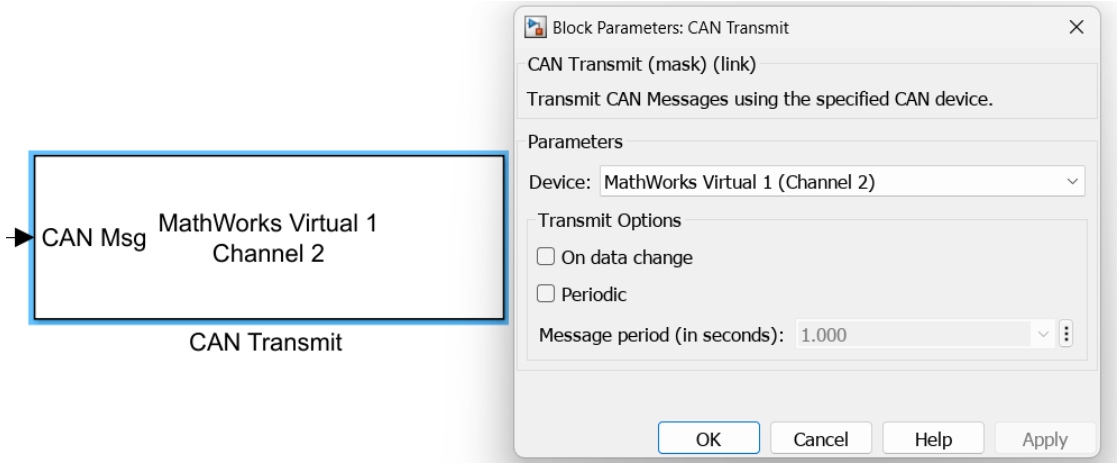


Figura 2.39. Bloque “CAN Transmit”, Parámetros, Sistema Encendido.

En el subsistema se agrega los bloques “CAN Recieve” y “CAN Unpack”. El bloque “CAN Recieve” se verifica que el parámetro “Device” este seleccionado “MathWorks Virtual 1 (Channel 2)”.

El bloque “CAN Unpack” con los siguientes parámetros:

- **Data to be output as:** “raw data”
- **Name:** “CAN Msg” (valor predeterminado)
- **Identifier Type:** “Standard (11-bit identifier)” (valor predeterminado)
- **Identifier:** “200” (ID del mensaje desde el controlador)
- **Length(bytes):** “1”

Finalmente se agrega el bloque “CAN Configuration”, véase Figura 2.40. Seleccionar el “Channel 2” el cual actuara como el nodo de ataque.

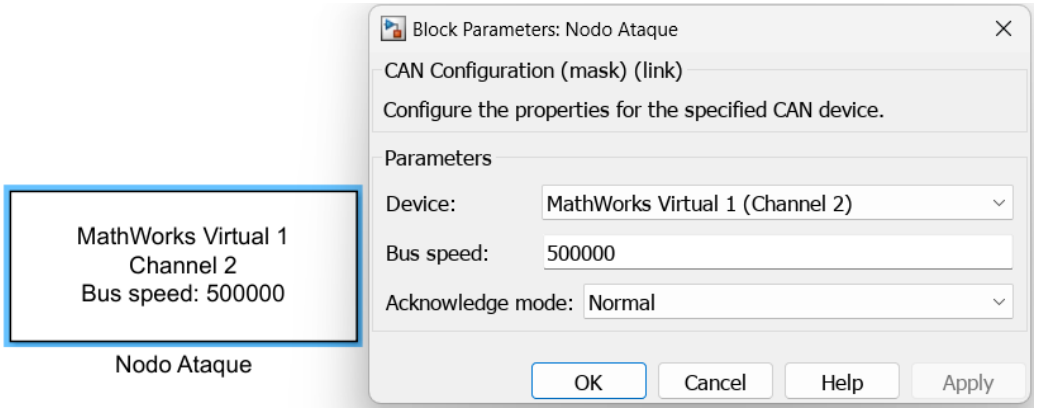


Figura 2.40. Bloque “CAN Configuration”, Parámetros.

3. RESULTADOS Y DISCUSIÓN

En esta sección se presenta los resultados obtenidos de las simulaciones realizadas en MATLAB/Simulink, que evalúan el impacto de dos tipos de ataques – Dos (Denegación de Servicio) e inyección de datos – en la red de comunicación CAN de dos sistemas vehiculares críticos: sistema antibloqueo de frenos (ABS) y el sistema de encendido sin llave (Keyless).

Se llevaron a cabo dos conjuntos de pruebas para cada sistema:

- Pruebas sin ataques: Estas pruebas establecen la línea base del comportamiento esperado del sistema ABS y de encendido en condiciones normales de funcionamiento, sin la presencia de amenazas externas.
- Pruebas con ataques: En este conjunto de pruebas, se simularon dos tipos de ataques específicos a la red CAN:
    - Ataque DoS al sistema ABS: Se inyectaron mensajes CAN falsos para alterar la lectura del sensor de velocidad de las ruedas, interrumpiendo el funcionamiento normal del sistema ABS.
    - Ataque de inyección de datos al sistema de encendido: Se simulo la presencia del llavero inteligente (FOB) mediante la inyección de mensajes CAN, intentando engañar al sistema para que permita el encendido del motor sin la autorización del usuario.

3.1. Resultados

3.1.1. Pruebas sin ataques

3.1.1.1. Sistema ABS

En la Figura 3.1, se observa el comportamiento del sistema ABS en condiciones normales de funcionamiento, la velocidad del vehículo (línea verde) disminuye de manera controlada desde su velocidad inicial hasta alcanzar un velocidad próxima a cero, . La velocidad de la rueda (línea roja) también experimenta una disminución, aunque presenta algunas oscilaciones menores que son corregidas por el sistema ABS, evitando el bloqueo de las ruedas.

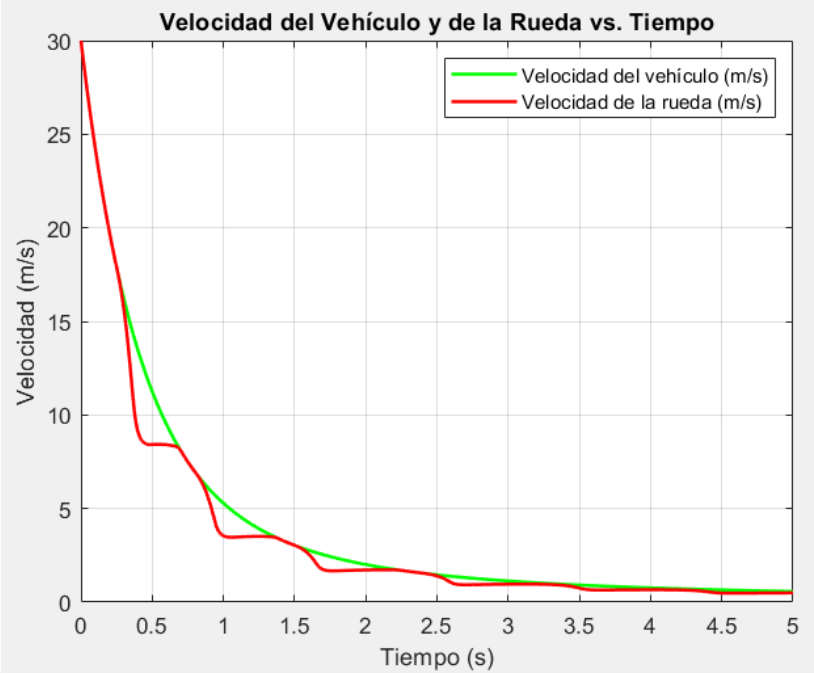


Figura 3.1. Curva de cambio de velocidad del vehículo y la rueda.

El gráfico de deslizamiento, véase Figura 3.2, muestra la evolución del deslizamiento de la rueda (línea rosada) durante un evento de frenado a lo largo del tiempo, en condiciones normales sin la presencia de ataques.

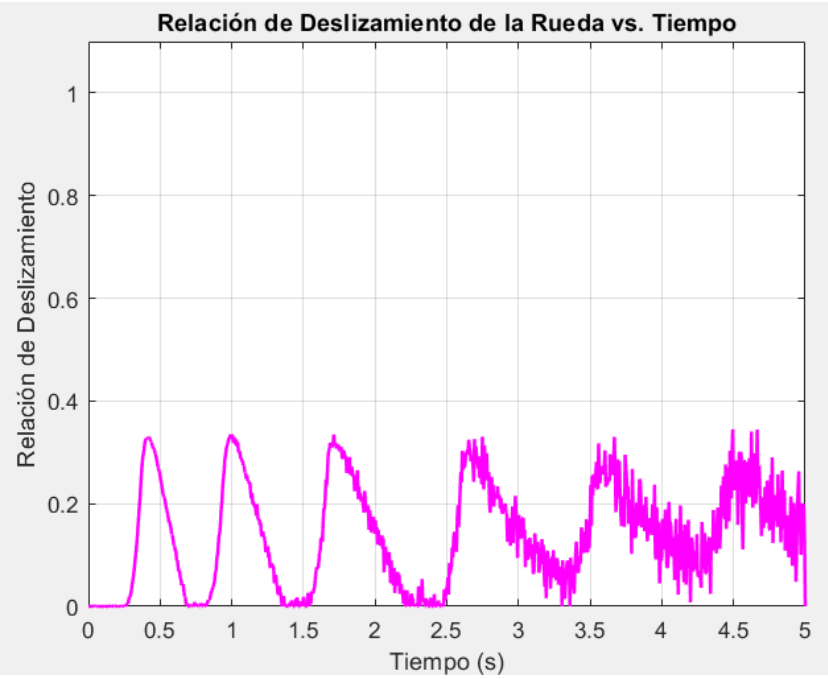


Figura 3.2. Curva variación del deslizamiento.

La Figura 3.3, ilustra la variación de la fuerza de frenado ejercida por el actuador (línea negra) del sistema ABS a lo largo del tiempo durante el evento de frenado.

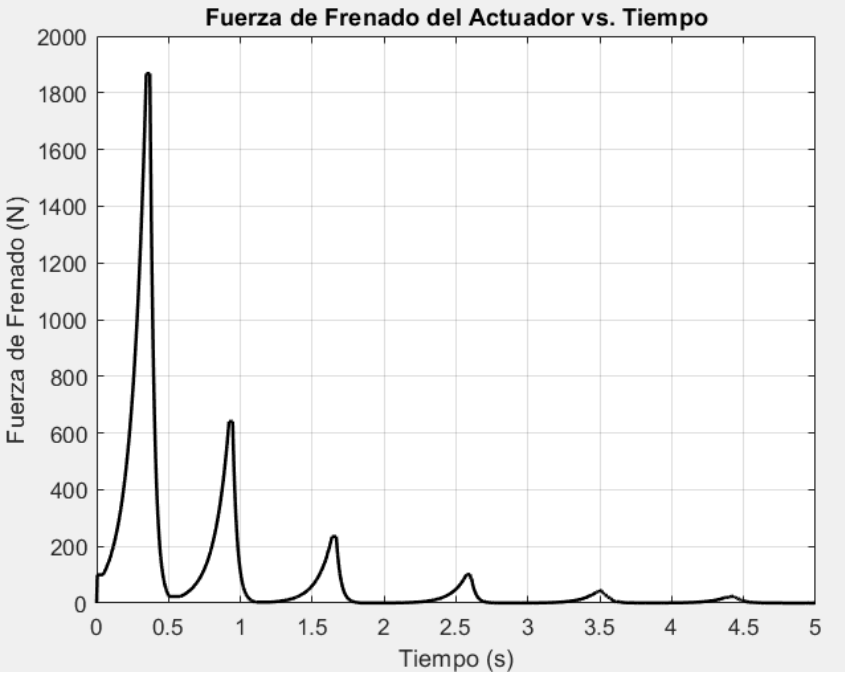


Figura 3.3. Fuerza de Frenado del Actuador.

3.1.1.2. Sistema de encendido

La Figura 3.4, ilustra el funcionamiento normal del sistema de encendido, el cual consta de tres estados principales:

- Estado 1: El sistema se encuentra inactivo. El indicador de presencia del FOB y el indicador de encendido están en rojo, y el tacómetro marca cero revoluciones, lo que indica que el motor esta apagado y el FOB ausente.
- Estado 2: El FOB es detectado dentro del vehículo. El indicador de presencia del FOB cambia a verde, pero el indicador de encendido permanece en rojo y el tacómetro sigue en cero. Esto significa que el sistema esta listo para arrancar el motor, pero aun no se ha presionado el botón.
- Estado 3: El motor está encendido y en funcionamiento. El indicador de encendido y el indicador de presencia del FOB están en verde. El tacómetro muestra un valor distinto de cero, lo que refleja las revoluciones del motor en marcha.

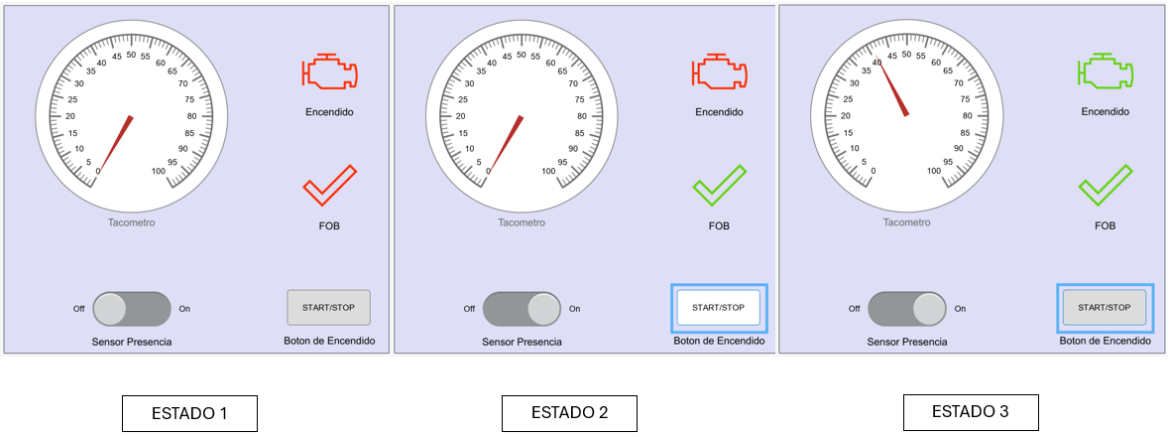


Figura 3.4. Secuencia de encendido.

3.1.2. Pruebas con ataques

3.1.2.1. Sistema de ABS

La figura 3.5, que presenta la velocidad del vehículo y de la rueda, el tiempo de activación del ataque DoS durante un evento de frenado. La franja rosada, que abarca el intervalo de tiempo entre 2 y 3 segundos, señal el periodo en el que ataque DoS esta activo. Durante este periodo, la velocidad de la rueda (línea azul) experimenta un descenso a cero mientras que la velocidad del vehículo (línea verde) es diferente a cero, durante esta ventana de tiempo.

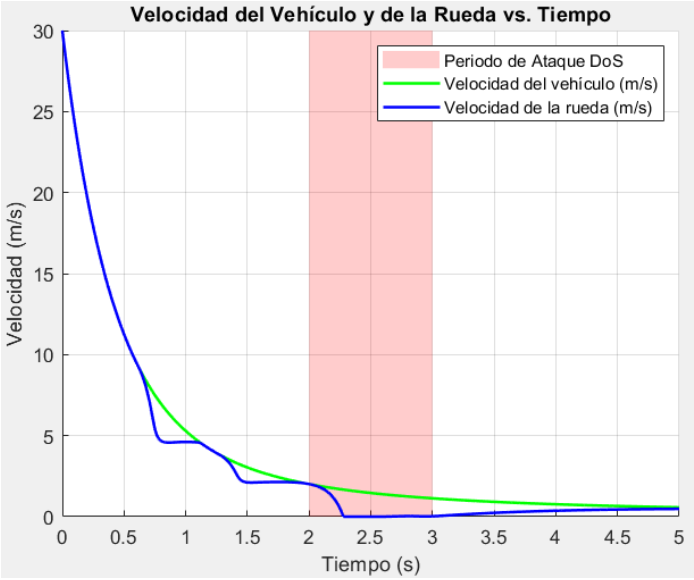


Figura 3.5. Curva de cambio de velocidad del vehículo y la rueda, Ataque DoS.

La Figura 3.6, muestra la relación de deslizamiento de una rueda durante un evento de frenado, destacando el periodo en el que se simula un ataque de Denegación de Servicios (DoS) al sistema ABS.

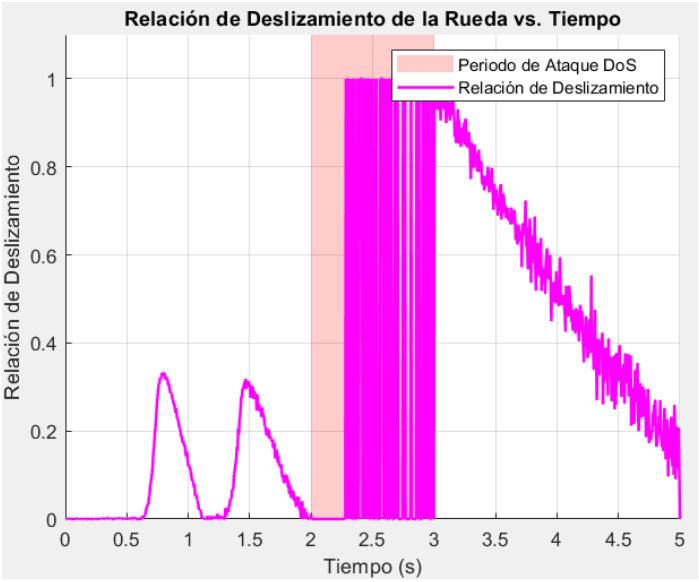


Figura 3.6. Curva variación del deslizamiento, Ataque DoS.

La Figura 3.7, ilustra la variación de la fuerza de frenado ejercida por el actuador del sistema ABS a lo largo del tiempo durante el evento de frenado, y resalta el área rosada el cual representa el periodo en el que está activo el ataque DoS. Durante el ataque, la fuerza de frenado (línea negra) se vuelve errática y alcanza valores mucho más altos que en condiciones normales.

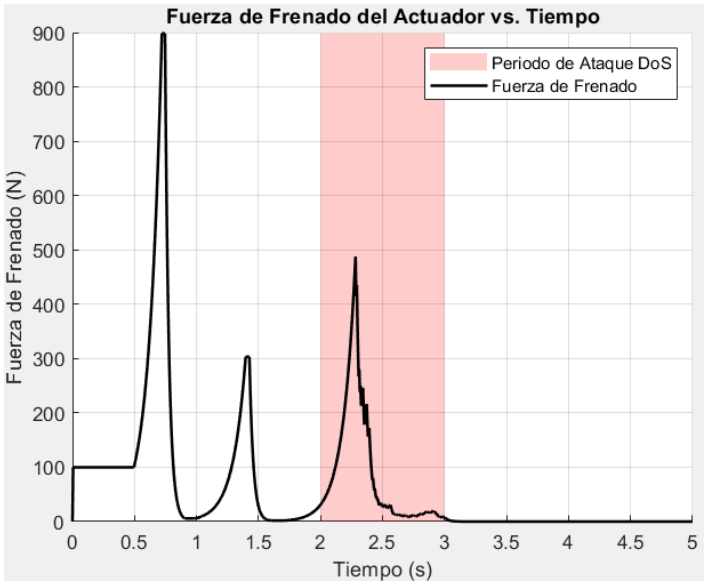


Figura 3.7. Fuerza de Frenado del Actuador, Ataque DoS.

La Figura 3.8, muestra la transmisión de mensajes durante el tiempo de duración de la simulación, y resalta el tiempo durante el ataque DoS esta activó, franja rosada.

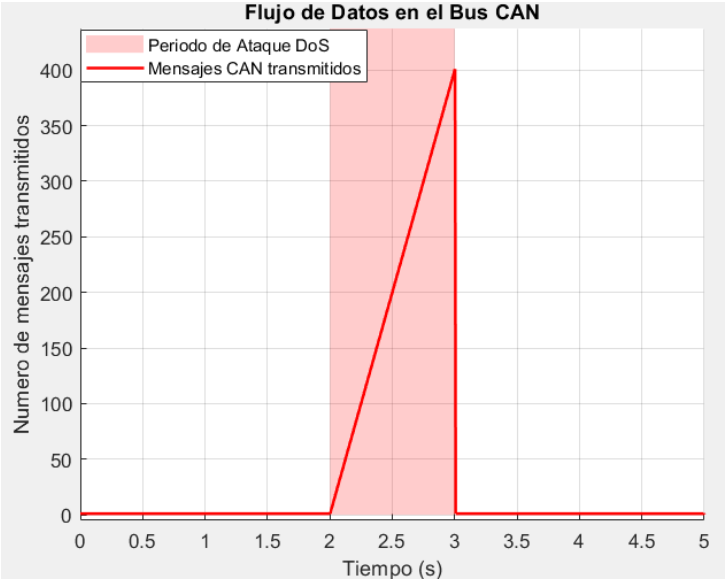


Figura 3.8. Flujo de datos, Ataque DoS.

3.1.2.2. Sistema de Encendido

La Figura 3.9, ilustra los diferentes estados del tablero implementado en Simulink para visualizar el comportamiento del sistema de encendido, bajo un ataque de inyección de datos.

- Estado 1: Sistema se encuentra en reposo, con los indicadores en rojo y el tacómetro en cero, indicando que el motor esta apagado y el FOB no está presente.
- Estado 2: Activación del ataque de inyección de datos, donde se visualiza el ID del mensaje CAN del sensor de presencia evidenciado la manipulación de la presencia del FOB
- Estado 3: El indicador del sensor permanece en rojo (señalando la ausencia del FOB), el indicador de encendido cambia a verde y el tacómetro muestra un valor distinto de cero.

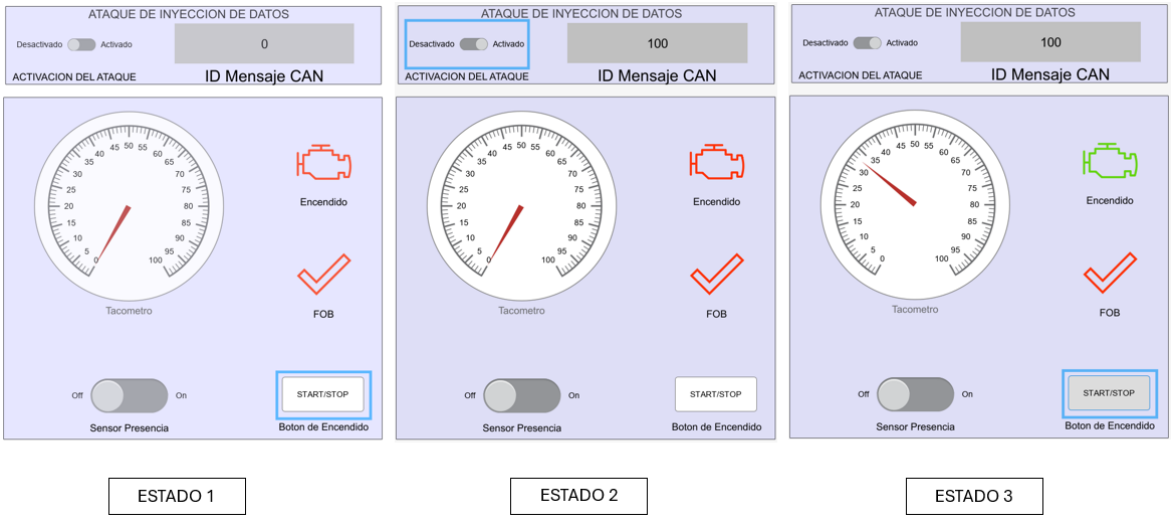


Figura 3.9. Secuencia de encendido, Ataque DoS.

3.2. Discusión

3.2.1. Resultados sin ataques

Los resultados de la simulación en condiciones normales, ilustrados en las Figuras 3.1 a 3.3, demuestran el correcto funcionamiento del sistema ABS. La disminución controlada de la velocidad del vehículo y las pequeñas oscilaciones corregidas en la velocidad de la rueda evidencian la efectividad del sistema en prevenir el bloqueo de las ruedas durante el frenado. El deslizamiento se mantiene dentro del rango optimo, lo que confirma la maximización de la tracción y el control del vehículo. Los ajustes precisos en la fuerza de frenado ejercida por el actuador reflejan la capacidad del controlador ABS para adaptarse a las condiciones cambiantes durante el frenado.

En la Figura 3.1, las curvas de las velocidades se aproximan a cero, esto es debido al tipo de control implementado (BANG-BANG) que estas curvas no llegue completamente a cero durante el tiempo de la simulación. En la Figura 3.2, se muestra como el controlador (BANG-BANG) no logra corregir eficientemente el deslizamiento y mantenerlo dentro del valor optimo de 0.2 en el transcurso de la simulación, mostrando un comportamiento oscilatorio durante los últimos segundos.

La secuencia de encendido mostrada en la Figura 3.4 valida la lógica de control implementada en la simulación. El sistema responde adecuadamente a la presencia del FOB y a la interacción del usuario con el botón de encendido, permitiendo el arranque del motor solo cuando se cumplen las condiciones de seguridad.

3.2.2. Resultados con ataques

Sistema ABS (Ataque DoS), las Figuras 3.5 a 3.7 revelan el impacto significativo del ataque DoS en el sistema ABS. La manipulación de los mensajes del sensor de velocidad de la rueda induce fluctuaciones abruptas en la velocidad de la rueda y en el deslizamiento, lo que con lleva a una pérdida de control sobre el frenado. El comportamiento errático de la fuerza de frenado ejercida por el actuador, con picos que superan ampliamente los valores normales, evidencia la desestabilización del sistema y el riesgo potencial de bloqueo de la ruedas y pérdida del control del vehículo. Estos resultados resaltan la vulnerabilidad del sistema ABS a ataque DoS y la importancia de implementar medidas de seguridad para proteger la integridad de la comunicación CAN.

En la Figura 3.8, se observa durante los primeros segundos de la simulación la transmisión de los mensajes se mantiene constante en uno. Sin embargo, al iniciar el ataque DoS al sistema ABS, específicamente entre los segundos dos y tres, se produce un incremento en el número de mensajes transmitidos. Este flujo de mensajes alcanza un pico de 400 mensajes en un segundo, lo que causa una sobrecarga significativa en el BUS CAN. Este comportamiento es característico del ataque DoS.

La Figura 3.9 demuestra el ataque de inyección de datos en comprometer la seguridad del sistema de encendido. La manipulación exitosa de los mensajes CAN permite simular la presencia del FOB, engañando al sistema y permitiendo el encendido del motor sin autorización. Este resultado crítico subraya la necesidad de fortalecer la autentificación y la integridad de los mensajes.

En general, los resultados de las simulaciones con ataques resaltan las vulnerabilidades inherentes al protocolo CAN y la importancia de implementar medidas de seguridad adicionales para proteger los sistemas vehiculares críticos. La ausencia de autentificación y cifrado en el diseño original de CAN facilita la explotación de estas vulnerabilidades, lo que puede tener consecuencias graves para la seguridad y la integridad de los vehículos. Es fundamental considerar la implementación de mecanismos de seguridad mas robustos, como la autentificación de mensajes, el cifrado de datos y la detección de anomalías.

4. CONCLUSIONES

- Mediante una revisión bibliográfica exhaustiva se adquirió conocimientos sobre el funcionamiento, características y vulnerabilidades inherentes al bus CAN en el

contexto de las redes vehiculares. Identificando y seleccionando dos ataques críticos para su simulación y análisis en un entorno controlado.

- La implementación de una red CAN simulada utilizando el Vehicle Network Toolbox de MATLAB, replico la estructura básica y la interacción entre sensores, actuadores y unidades de control. Sin embargo, la falta de actividades de bajo nivel, como arbitraje, limito la capacidad de evaluar el comportamiento real de la red CAN en condiciones de ataque.

- La selección de sensores y actuadores específicos, como el sensor de velocidad en el sistema ABS y el sensor de presencia del llavero (FOB) en el sistema de encendido, permitió modelar la interacción de dichos componentes con la red CAN, enriqueciendo el análisis de las vulnerabilidades inherentes del protocolo.

- En el desarrollo de los ataques simulados a la red CAN permitió inyectar mensajes maliciosos de manera controlada y precisa, emulando las acciones de un ataque real. Brindando la flexibilidad necesaria para evaluar diferentes escenarios de ataques, como la manipulación en los mensajes del sensor de velocidad de la rueda en el sistema ABS y la inyección de mensajes falsos en el sistema de encendido, enriqueciendo el análisis de las vulnerabilidades y sus consecuencias.

- La simulación de ataques, inyección de datos y la denegación de servicio (DoS), demostró de manera clara el impacto negativo que estos pueden tener sobre la integridad y funcionalidad de los sistemas vehiculares. Estos resultados subrayan la importancia de implementar medidas de seguridad adicionales en las redes CAN.

- La simulación del ataque de denegación de servicio (DoS) demostró de manera clara el impacto negativo que puede tener sobre la integridad y funcionalidad de los sistemas vehiculares. Durante el ataque, el flujo de mensajes en el BUS CAN aumentó de manera significativa, pasando de un mensaje en cada interacción del bucle de simulación en condiciones normales hasta 400 mensajes en un segundo, saturando completamente la red de comunicación. Esta sobrecarga afectó directamente la capacidad del sistema ABS, para funcionar correctamente, poniendo en riesgo la seguridad del vehículo.

- Las pruebas realizadas en las simulaciones, en condiciones normales como bajo ataques, garantizaron la validez y la confiabilidad de los resultados obtenidos. El correcto funcionamiento de la red CAN sin ataques permite establecer una línea

base comparando el comportamiento en los sistemas ABS y de encendido en presencia de amenazas.

A continuación, se proponen las siguientes recomendaciones:

- Implementar mecanismos de autenticación y cifrado en el protocolo CAN, la falta de estos mecanismos en el diseño original de la comunicación CAN, lo hace vulnerable a ataques de inyección de datos y suplantación de identidad. Por esto es recomendable explorar e implementar soluciones que incorporen autenticación y cifrado para proteger la integridad y confidencialidad de los mensajes transmitidos en la red CAN.
- Es importante implementar sistemas de detección de anomalías y mecanismos de respuestas a incidentes para identificar y mitigar ataques en tiempo real. Incluyendo el monitoreo continuo del tráfico de la red CAN, detección de patrones de mensajes inusuales y la implementación de contramedidas para bloquear o aislar nodos maliciosos.
- A pesar de las limitaciones identificadas en la simulación, como la ausencia de procesos de bajo nivel, se recomienda la exploración de herramientas y técnicas que permitan una evaluación más realista y completa de las redes CAN en escenarios vehiculares.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] H. F. Garcia, “IMPLEMENTACIÓN DE UN MODELO DE PROTOCOLO CAN EN UN AUTOMOVIL VOLSWAGEN 2009 MEDIANTE SIMULINK”, 2020. [En línea]. Disponible en: https://example.com

[2] M. L. Defaz Andrango, “Escuela Politecnica Nacional”, 2007. [En línea]. Disponible en: https://bibdigital.epn.edu.ec/bitstream/15000/403/1/CD-0788.pdf

[3] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, y L. Kilmartin, “Intra-Vehicle Networks: A Review”, *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, núm. 2, pp. 534–545, abr. 2015, doi: 10.1109/TITS.2014.2320605.

[4] L. G. Sánchez Vela *et al.*, “Revisión documental del protocolo CAN como herramienta de comunicación y aplicación en vehículos”, 2016, *Ingeniería Vehicular e Integridad Estructural del Instituto Mexicano del Transporte, Sanfandila, Qro*.

[5] F. Hartwich, B. Müller, T. Führer, y R. Hügel, “CAN with Flexible Data-Rate”, en *Robert Bosch GmbH*, [En línea]. Disponible en: http://www.can-cia.org/fileadmin/cia/files/icc/13/hartwich.pdf

[6] A. Alfardus y D. B. Rawat, “Evaluation of CAN Bus Security Vulnerabilities and Potential Solutions”, en *Proceedings - 2023 6th International Conference of Women in Data Science at Prince Sultan University, WiDS-PSU 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 90–97. doi: 10.1109/WiDS-PSU57071.2023.00030.

[7] “ISO 11898-2:2016 Road vehicles -- Controller area network (CAN) -- Part 2: High-speed medium access unit”.

[8] W. Lawrenz, *CAN System Engineering: From Theory to Practical Applications*, 2a ed. Springer-Verlag, 2013.

[9] G. Stoeger, M. Karner, y R. Weiss, “Automotive Networks and Protocols”, en *Automotive Software Engineering*, Vienna: Springer, pp. 21–54.

[10] S. Corrigan, “Introduction to the Controller Area Network (CAN)”, *Texas Instruments Incorporated*, vol. SLOA101B, pp. 1–17, 2016.

[11] J. Pöyhönen, “Kyberturvallisuuden johtaminen ja kehittäminen osana kriittisen infrastruktuurin organisaation toimintaa–Systeemiajattelu”, 2020.

[12] “OBD2 Explained - A Simple Intro [2023] – CSS Electronics”. Consultado: el 23 de julio de 2024. [En línea]. Disponible en: https://www.csselectronics.com/pages/obd2-explained-simple-intro

[13] S. Checkoway y et al., “Comprehensive experimental analyses of automotive attack surfaces”, en *USENIX Security Symposium*, San Francisco, pp. 447–462.

[14] M. Bozdal, M. Samie, y I. Jennions, “A Survey on CAN Bus Protocol: Attacks, Challenges, and Potential Solutions”, en *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, IEEE, ago. 2018, pp. 201–205. doi: 10.1109/iCCECOME.2018.8658720.

[15] “Vehicle Network Toolbox™ User’s Guide”, vol. R2023a, 2023.

[16] R. Rajamani, *Vehicle Dynamics and Control*. New York: Springer, 2012.

[17] “Anti-Lock Braking Systems (ABS) Working”. Consultado: el 25 de julio de 2024. [En línea]. Disponible en: https://www.linkedin.com/pulse/anti-lock-braking-systems-abs-working-karthick-m

[18] H. Winner, S. Hakuli, y G. Wolf, *Handbook of Driver Assistance Systems: Basic Information, Components and Systems for Active Safety and Comfort*. Wiesbaden: Springer Vieweg, 2015.

[19] “Keyless Ignition Systems | NHTSA”. Consultado: el 25 de julio de 2024. [En línea]. Disponible en: https://www.nhtsa.gov/driver-assistance-technologies/keyless-ignition-systems

[20] U. Epa, O. of Transportation, y A. Quality, “The 2022 EPA Automotive Trends Report: Greenhouse Gas Emissions, Fuel Economy, and Technology since 1975 (EPA-420-R-22-029, December 2022)”.

[21] “Los límites de velocidad en Ecuador | Panavial”. Consultado: el 1 de agosto de 2024. [En línea]. Disponible en: https://www.panavial.com/limites-velocidad-ecuador/

[22] W. Cao, “Modeling and simulation of the anti-lock braking system based on MATLAB/Simulink”, *J Phys Conf Ser*, vol. 1941, núm. 1, p. 012075, jun. 2021, doi: 10.1088/1742-6596/1941/1/012075.

6. ANEXOS

ANEXO A. Documentación del Vehicle Network Toolbox

ANEXO B. Script de MATLAB, Sistema ABS. Funcionamiento Normal

ANEXO C. Función de MATLAB, Motor de Arranque, Simulink

ANEXO D. Script de MATLAB, Sistema ABS. Funcionamiento Anormal.

ANEXO E. Función de MATLAB, Simulink CAN ID

ANEXO A

Documentación del Vehicle Network Toolbox

La documentación del Vehicle Network Toolbox se encuentra disponible en el siguiente enlace:

VNTANEXOA.pdf

ANEXO B

Script de MATLAB, Sistema ABS. Funcionamiento Normal

```
%% Simulación ABS con Dinámica del Vehículo, Controlador Bang-Bang, Sensor y Actuador con Comunicación CAN

% Parámetros de la simulación
tiempo_simulacion = 5;     % Tiempo de simulación en segundos
t = linspace(0, tiempo_simulacion, 1000); % Vector de tiempo
dt = t(2) - t(1);          % Paso de tiempo

% Parámetros del vehículo
masa = 1500;               % Masa del vehículo (kg)
g = 9.81;                  % Aceleración gravitacional (m/s^2)
velocidad_inicial = 30;    % Velocidad inicial del vehículo (m/s)
coeficiente_arrastre = 0.4; % Coeficiente de arrastre del aire
area_frontal = 2.2;        % Área frontal del vehículo (m^2)
coeficiente_resistencia_rodadura = 0.015; % Coeficiente de resistencia a la rodadura

% Parámetros de la rueda
radio_neumatico = 0.3;     % Radio del neumático (m)
inercia_rueda = 0.5;       % Inercia de la rueda (kg*m^2)
num_ruedas = 4;            % Número de ruedas (suponiendo 4)

% Parámetros del controlador ABS (Bang-Bang)
deslizamiento_optimo = 0.2;         % Relación de deslizamiento óptima
umbral_deslizamiento_alto = 0.25;    % Umbral superior de deslizamiento para liberación de freno
umbral_deslizamiento_bajo = 0.15;    % Umbral inferior de deslizamiento para aplicación de freno

% Torque de frenado máximo
Torque_maximo_freno = 1000;    % Torque de frenado máximo (N*m), reemplazar con valor real

% Parámetros del sensor (simulado)
```

```
desviacion_estandar_ruido_sensor = 0.1; % Desviación estándar del ruido del sensor

% Parámetros del actuador (simulado)
retardo_actuador = 0.05;     % Retardo de respuesta del actuador en segundos

% Definir canal y parámetros del mensaje CAN
canal = 1; % Número de canal CAN
canBus = canChannel('MathWorks', 'Virtual 1', canal);

% Abrir canal CAN
start(canBus);   %iniciar comunicacion CAN

% Definir identificadores de mensajes
idMensajeSensor = 1; % Identificador de mensaje de lectura del sensor
idMensajeControlador = 2; % Identificador de mensaje de salida del controlador
longitudMensaje = 8; % Longitud de datos en bytes (máx. 8 para CAN estándar)

% Preasignar matrices
velocidad_vehiculo = zeros(size(t));
velocidad_rueda = zeros(size(t));
deslizamiento = zeros(size(t));
fuerza_frenado = zeros(size(t));
fuerza_longitudinal = zeros(size(t));

% Inicializar velocidad del vehículo y de la rueda
velocidad_vehiculo(1) = velocidad_inicial;
velocidad_rueda(1) = velocidad_inicial / radio_neumatico;   % Inicialización
corregida en rad/s

% Inicializar fuerza de frenado con un valor inicial
fuerza_frenado(1) = 100;  % Ejemplo de fuerza de frenado inicial (N)

% Inicializar matrices de sensor y actuador
velocidad_rueda_sensor = zeros(size(t));
fuerza_frenado_actuador = zeros(size(t));

% Constante de amortiguamiento (damping)
```

```
coeficiente_amortiguamiento = 0.5; % Ajusta este valor para controlar el amortiguamiento

% Coeficiente de frenado no lineal
coeficiente_frenado = 0.50; % Ajusta este valor para controlar la frenada no lineal
factor_desaceleracion = 0.2; % Ajusta este factor para controlar el comportamiento de la curva

% Bucle de simulación
for i = 1:length(t)-1
    %% Sensor
    % Simular ruido del sensor (ruido gaussiano simple)
    velocidad_rueda_sensor(i) = velocidad_rueda(i) + desviacion_estandar_ruido_sensor * randn; % Medición del sensor simulada con ruido

    % Transmitir lectura del sensor (idMensajeSensor)
    datosMensajeSensor = typecast(velocidad_rueda_sensor(i), 'uint8');
    mensajeCAN_Sensor = canMessage(idMensajeSensor, false, longitudMensaje);
    mensajeCAN_Sensor.Data = [datosMensajeSensor, zeros(1, longitudMensaje - length(datosMensajeSensor))]; % Rellenar los bytes restantes con ceros si es necesario
    transmit(canBus, mensajeCAN_Sensor);

    % Pausa para asegurar que el mensaje sea transmitido
    pause(0.01);

    % Leer mensajes CAN para simular la recepción de datos del sensor
    mensajesRecibidos = receive(canBus, Inf, 'OutputFormat', 'timetable');

    % Inicializar variable para almacenar la lectura del sensor recibida
    lecturaSensorRecibida = [];

    % Procesar cada mensaje recibido
    for j = 1:height(mensajesRecibidos)
        datosRecibidos = mensajesRecibidos.Data{j};
        % Convertir el array de bytes de nuevo a un entero
        valorRecibido = typecast(uint8(datosRecibidos), 'double');
```

```
        % Almacenar valor recibido basado en el ID del mensaje
        if mensajesRecibidos.ID(j) == idMensajeSensor
            lecturaSensorRecibida = valorRecibido;
        end
    end

    %% Controlador
    % Modelo de frenado no lineal del vehículo
    desaceleracion = -coeficiente_frenado * velocidad_vehiculo(i)^2 / (1 +
factor_desaceleracion * velocidad_vehiculo(i)); % Frenado logístico

    % Simular la velocidad del vehículo con el frenado no lineal
    velocidad_vehiculo(i+1) = velocidad_vehiculo(i) + desaceleracion * dt;

    % Asegurarse de que la velocidad no sea negativa
    velocidad_vehiculo(i+1) = max(0, velocidad_vehiculo(i+1));

    % Calcular deslizamiento usando la lectura del sensor recibida
    deslizamiento(i) = 1 - (lecturaSensorRecibida * radio_neumatico) /
velocidad_vehiculo(i);
    deslizamiento(i) = max(0, min(1, deslizamiento(i))); % Limitar deslizamiento
entre 0 y 1

    % Lógica del controlador Bang-Bang ajustada
    if deslizamiento(i) > umbral_deslizamiento_alto
        fuerza_frenado(i+1) = max(0, fuerza_frenado_actuador(i) * 0.85); %
Reducir la fuerza de frenado más gradualmente
    elseif deslizamiento(i) < umbral_deslizamiento_bajo
        fuerza_frenado(i+1) = min(fuerza_frenado_actuador(i) * 1.05,
Torque_maximo_freno / radio_neumatico); % Incrementar la fuerza de frenado más
gradualmente
    else
        fuerza_frenado(i+1) = fuerza_frenado_actuador(i);
    end

    % Transmitir salida del controlador (idMensajeControlador)
    datosMensajeControlador = typecast(fuerza_frenado(i+1), 'uint8');
```

```
    mensajeCAN_Controlador = canMessage(idMensajeControlador, false, longitudMensaje);
    mensajeCAN_Controlador.Data = [datosMensajeControlador, zeros(1, longitudMensaje - length(datosMensajeControlador))]; % Rellenar los bytes restantes con ceros si es necesario
    transmit(canBus, mensajeCAN_Controlador);

    % Pausa para asegurar que el mensaje sea transmitido
    pause(0.01);

    % Leer mensajes CAN para simular la recepción de datos de salida del controlador
    mensajesRecibidos = receive(canBus, Inf, 'OutputFormat', 'timetable');

    % Inicializar variable para almacenar la salida del controlador recibida
    salidaControladorRecibida = [];

    % Procesar cada mensaje recibido
    for j = 1:height(mensajesRecibidos)
        datosRecibidos = mensajesRecibidos.Data{j};
        % Convertir el array de bytes de nuevo a un entero
        valorRecibido = typecast(uint8(datosRecibidos), 'double');

        % Almacenar valor recibido basado en el ID del mensaje
        if mensajesRecibidos.ID(j) == idMensajeControlador
            salidaControladorRecibida = valorRecibido;
        end
    end

    %% Actuador
    % Simular el retardo de respuesta del actuador (modelo de retardo simple)
    if i > retardo_actuador / dt
        fuerza_frenado_actuador(i+1) = salidaControladorRecibida;
    else
        fuerza_frenado_actuador(i+1) = fuerza_frenado(1); % Usar la fuerza de frenado inicial si el retardo no ha pasado
    end
```

```
    % Calcular el Torque de la rueda considerando la fricción variable
    torque_rueda = fuerza_frenado_actuador(i+1) / num_ruedas * radio_neumatico;

    % Introducir variabilidad en la respuesta del actuador
    variabilidad_respuesta_actuador = 1 + 0.05 * randn; % Pequeña variación aleatoria

    % Calcular la aceleración angular considerando el amortiguamiento
    aceleracion_angular = -torque_rueda / inercia_rueda + coeficiente_amortiguamiento * (velocidad_vehiculo(i) / radio_neumatico - velocidad_rueda(i));

    % Actualizar la velocidad de la rueda
    velocidad_rueda(i+1) = max(0, velocidad_rueda(i) + aceleracion_angular * dt);

    % Limitar la velocidad de la rueda para evitar exceder la velocidad del vehículo
    velocidad_rueda(i+1) = min(velocidad_rueda(i+1), velocidad_vehiculo(i+1)/radio_neumatico);

    % Calcular la fuerza longitudinal total
    fuerza_arrastre = 0.5 * coeficiente_arrastre * area_frontal * velocidad_vehiculo(i+1)^2;
    resistencia_rodadura = coeficiente_resistencia_rodadura * masa * g;
    fuerza_longitudinal(i+1) = -fuerza_frenado(i+1) - fuerza_arrastre - resistencia_rodadura;
end

% Detener y liberar canal CAN
stop(canBus);

% Gráficos

% Gráfico 1: Velocidad del vehículo y velocidad de la rueda
figure;
plot(t, velocidad_vehiculo, 'g', 'LineWidth', 1.5, 'DisplayName', 'Velocidad del vehículo (m/s)');
hold on; % Mantener el gráfico para superponer
```

```
plot(t, velocidad_rueda * radio_neumatico, 'r', 'LineWidth', 1.5, 'DisplayName',
'Velocidad de la rueda (m/s)');
xlabel('Tiempo (s)');
ylabel('Velocidad (m/s)');
title('Velocidad del Vehículo y de la Rueda vs. Tiempo');
legend('show');
grid on;
hold off; % Liberar el gráfico para otras operaciones

% Gráfico 2: Relación de Deslizamiento
figure;
plot(t, deslizamiento, 'm', 'LineWidth', 1.5);
xlabel('Tiempo (s)');
ylabel('Relación de Deslizamiento');
title('Relación de Deslizamiento de la Rueda vs. Tiempo');
ylim([0, 1.1]);
grid on;

% Gráfico 3: Fuerza de Frenado del Actuador
figure;
plot(t, fuerza_frenado_actuador, 'k', 'LineWidth', 1.5);
xlabel('Tiempo (s)');
ylabel('Fuerza de Frenado (N)');
title('Fuerza de Frenado del Actuador vs. Tiempo');
grid on;
```

ANEXO C

Función de MATLAB, Motor de Arranque, Simulink

```
function [revoluciones, estado]  = simularMotor(Controlador)
    % Variable persistente para mantener el estado actual del motor
    persistent estadoActual

    % Inicializa estadoActual si es la primera vez que se llama a la función
    if isempty(estadoActual)
        estadoActual = 0;
    end

    if Controlador == 1
        if estadoActual < 20
            % Incrementa el estado de 0 a 20
            estadoActual = estadoActual + 1;
            pause(0.15);
        elseif estadoActual >= 20
            % Una vez alcanzado 20, comienza a oscilar entre 20 y 35
            if estadoActual <= randi([35,50])
                estadoActual = estadoActual + 1; % Incrementa rápidamente para
oscilación
                pause(0.15);
            else
                estadoActual = 20; % Retorna a 20 una vez alcanzado 35
                pause(0.15);
            end
        end
        revoluciones = estadoActual; % Devuelve el estado actual
        estado = 1;
    else
        if estadoActual > 0
            % Si se recibe señal de apagar, comienza a decrementar hasta 0
            estadoActual = estadoActual - 1;
            if estadoActual < 0
                estadoActual = 0; % Asegura que no se pase de 0
            end
        end
        revoluciones = estadoActual; % Devuelve el estado actual
        estado = 0;
    end
end
```

ANEXO D

Script de MATLAB, Sistema ABS. Funcionamiento Anormal.

```
%% Simulación ABS con Dinámica del Vehículo, Controlador Bang-Bang, Sensor y Actuador con Comunicación CAN, Ataque DoS

% Parámetros de la simulación
tiempo_simulacion = 5;     % Tiempo de simulación en segundos
t = linspace(0, tiempo_simulacion, 1000); % Vector de tiempo
dt = t(2) - t(1);          % Paso de tiempo

% Parámetros del vehículo
masa = 1500;               % Masa del vehículo (kg)
g = 9.81;                  % Aceleración gravitacional (m/s^2)
velocidad_inicial = 30;    % Velocidad inicial del vehículo (m/s)
coeficiente_arrastre = 0.4; % Coeficiente de arrastre del aire
area_frontal = 2.2;        % Área frontal del vehículo (m^2)
coeficiente_resistencia_rodadura = 0.015; % Coeficiente de resistencia a la rodadura

% Parámetros de la rueda
radio_neumatico = 0.3;     % Radio del neumático (m)
inercia_rueda = 0.5;       % Inercia de la rueda (kg*m^2)
num_ruedas = 4;            % Número de ruedas (suponiendo 4)

% Parámetros del controlador ABS (Bang-Bang)
deslizamiento_optimo = 0.2;         % Relación de deslizamiento óptima
umbral_deslizamiento_alto = 0.25;    % Umbral superior de deslizamiento para liberación de freno
umbral_deslizamiento_bajo = 0.15;    % Umbral inferior de deslizamiento para aplicación de freno

% Torque de frenado máximo
torque_maximo_freno = 1000;    % Torque de frenado máximo (N*m), reemplazar con valor real

% Parámetros del sensor (simulado)
desviacion_estandar_ruido_sensor = 0.1; % Desviación estándar del ruido del sensor

% Parámetros del actuador (simulado)
retardo_actuador = 0.5;    % Retardo de respuesta del actuador en segundos

% Definir canal y parámetros del mensaje CAN
canal = 1; % Número de canal CAN
canBus = canChannel('MathWorks', 'Virtual 1', canal);

% Abrir canal CAN
start(canBus);   % Iniciar comunicación CAN

% Definir identificadores de mensajes
idMensajeSensor = 1; % Identificador de mensaje de lectura del sensor
idMensajeControlador = 2; % Identificador de mensaje de salida del controlador
longitudMensaje = 8; % Longitud de datos en bytes (máx. 8 para CAN estándar)

% Parámetros del ataque DoS
activarAtaque = true; % Activa o desactiva el ataque DoS
```

```
tiempoInicioAtaque = 2; % Tiempo en segundos para iniciar el ataque
tiempoFinAtaque = 3; % Tiempo en segundos para terminar el ataque
flooding = 400; % Frecuencia de mensajes durante el ataque

% Preasignar matrices
velocidad_vehiculo = zeros(size(t));
velocidad_rueda = zeros(size(t));
deslizamiento = zeros(size(t));
fuerza_frenado = zeros(size(t));
fuerza_longitudinal = zeros(size(t));
mensajes_transmitidos = zeros(size(t)); % matriz de contador de mensajes CAN
transmitidos

% Inicializar velocidad del vehículo y de la rueda
velocidad_vehiculo(1) = velocidad_inicial;
velocidad_rueda(1) = velocidad_inicial / radio_neumatico;   % Inicialización
corregida en rad/s

% Inicializar fuerza de frenado con un valor inicial
fuerza_frenado(1) = 100;  % Ejemplo de fuerza de frenado inicial (N)

% Inicializar matrices de sensor y actuador
velocidad_rueda_sensor = zeros(size(t));
fuerza_frenado_actuador = zeros(size(t));

% Constante de amortiguamiento (damping)
coeficiente_amortiguamiento = 0.5; % Ajusta este valor para controlar el
amortiguamiento

% Coeficiente de frenado no lineal
coeficiente_frenado = 0.5; % Ajusta este valor para controlar la frenada no lineal
factor_desaceleracion = 0.2; % Ajusta este factor para controlar el comportamiento
de la curva

% Bucle de simulación
for i = 1:length(t)-1
    %% Sensor
    % Simular ruido del sensor (ruido gaussiano simple)
    velocidad_rueda_sensor(i) = velocidad_rueda(i) +
desviacion_estandar_ruido_sensor * randn; % Medición del sensor simulada con ruido

    % Transmitir lectura del sensor (idMensajeSensor)
    datosMensajeSensor = typecast(velocidad_rueda_sensor(i), 'uint8');
    mensajeCAN_Sensor = canMessage(idMensajeSensor, false, longitudMensaje);
    mensajeCAN_Sensor.Data = [datosMensajeSensor, zeros(1, longitudMensaje -
length(datosMensajeSensor))]; % Rellenar los bytes restantes con ceros si es
necesario
    transmit(canBus, mensajeCAN_Sensor);

    % Pausa para asegurar que el mensaje sea transmitido
    pause(0.01);

    % Leer mensajes CAN para simular la recepción de datos del sensor
    mensajesRecibidos = receive(canBus, Inf, 'OutputFormat', 'timetable');

    % Inicializar variable para almacenar la lectura del sensor recibida
    lecturaSensorRecibida = [];
```

```
    % Procesar cada mensaje recibido
    for j = 1:height(mensajesRecibidos)
        datosRecibidos = mensajesRecibidos.Data{j};
        % Convertir el array de bytes de nuevo a un entero
        valorRecibido = typecast(uint8(datosRecibidos), 'double');

        % Almacenar valor recibido basado en el ID del mensaje
        if mensajesRecibidos.ID(j) == idMensajeSensor
            lecturaSensorRecibida = valorRecibido;
        end
    end

    % Contador de mensajes
    mensajes_transmitidos(i) = mensajes_transmitidos(i) + 1;

    %% Ataque DoS agregar un for loop
    if activarAtaque && t(i) > tiempoInicioAtaque && t(i) < tiempoFinAtaque
        % Simular un ataque DoS enviando datos de sensor falsos
        for k = 1:flooding
            lecturaSensorRecibida = lecturaSensorRecibida * randi([1, 10]); %
Mensajes falsos

            mensajes_transmitidos(i+1) = mensajes_transmitidos(i) + 1;
        end
    end

    %% Controlador
    % Modelo de frenado no lineal del vehículo
    desaceleracion = -coeficiente_frenado * velocidad_vehiculo(i)^2 / (1 +
factor_desaceleracion * velocidad_vehiculo(i)); % Frenado logístico

    % Simular la velocidad del vehículo con el frenado no lineal
    velocidad_vehiculo(i+1) = velocidad_vehiculo(i) + desaceleracion * dt;

    % Asegurarse de que la velocidad no sea negativa
    velocidad_vehiculo(i+1) = max(0, velocidad_vehiculo(i+1));

    % Calcular deslizamiento usando la lectura del sensor recibida
    deslizamiento(i) = 1 - (lecturaSensorRecibida * radio_neumatico) /
velocidad_vehiculo(i);
    deslizamiento(i) = max(0, min(1, deslizamiento(i))); % Limitar deslizamiento
entre 0 y 1

    % Lógica del controlador Bang-Bang ajustada
    if deslizamiento(i) > umbral_deslizamiento_alto
        fuerza_frenado(i+1) = max(0, fuerza_frenado_actuador(i) * 0.85); %
Reducir la fuerza de frenado más gradualmente
    elseif deslizamiento(i) < umbral_deslizamiento_bajo
        fuerza_frenado(i+1) = min(fuerza_frenado_actuador(i) * 1.05,
torque_maximo_freno / radio_neumatico); % Incrementar la fuerza de frenado más
gradualmente
    else
        fuerza_frenado(i+1) = fuerza_frenado_actuador(i);
    end

    % Transmitir salida del controlador (idMensajeControlador)
```

```
    datosMensajeControlador = typecast(fuerza_frenado(i+1), 'uint8');
    mensajeCAN_Controlador     =     canMessage(idMensajeControlador,     false,
longitudMensaje);
    mensajeCAN_Controlador.Data     =     [datosMensajeControlador,     zeros(1,
longitudMensaje  -  length(datosMensajeControlador))];  %  Rellenar  los  bytes
restantes con ceros si es necesario
    transmit(canBus, mensajeCAN_Controlador);

    % Pausa para asegurar que el mensaje sea transmitido
    pause(0.01);

    % Leer mensajes CAN para simular la recepción de datos de salida del
controlador
    mensajesRecibidos = receive(canBus, Inf, 'OutputFormat', 'timetable');

    % Inicializar variable para almacenar la salida del controlador recibida
    salidaControladorRecibida = [];

    % Procesar cada mensaje recibido
    for j = 1:height(mensajesRecibidos)
        datosRecibidos = mensajesRecibidos.Data{j};
        % Convertir el array de bytes de nuevo a un entero
        valorRecibido = typecast(uint8(datosRecibidos), 'double');

        % Almacenar valor recibido basado en el ID del mensaje
        if mensajesRecibidos.ID(j) == idMensajeControlador
            salidaControladorRecibida = valorRecibido;
        end
    end

    %% Actuador
    % Simular el retardo de respuesta del actuador (modelo de retardo simple)
    if i > retardo_actuador / dt
        fuerza_frenado_actuador(i+1) = salidaControladorRecibida;
    else
        fuerza_frenado_actuador(i+1) = fuerza_frenado(1); % Usar la fuerza de
frenado inicial si el retardo no ha pasado
    end

    % Calcular el torque de la rueda considerando la fricción variable
    torque_rueda = fuerza_frenado_actuador(i+1) / num_ruedas * radio_neumatico;

    % Introducir variabilidad en la respuesta del actuador
    variabilidad_respuesta_actuador = 1 + 0.05 * randn; % Pequeña variación
aleatoria

    % Calcular la aceleración angular considerando el amortiguamiento
    aceleracion_angular     =     -torque_rueda     /     inercia_rueda     +
coeficiente_amortiguamiento  *  (velocidad_vehiculo(i)  /  radio_neumatico  -
velocidad_rueda(i));

    % Actualizar la velocidad de la rueda
    velocidad_rueda(i+1) = max(0, velocidad_rueda(i) + aceleracion_angular * dt);

    % Limitar la velocidad de la rueda para evitar exceder la velocidad del
vehículo
```

```
    velocidad_rueda(i+1)                =                min(velocidad_rueda(i+1),
velocidad_vehiculo(i+1)/radio_neumatico);

    % Calcular la fuerza longitudinal total
    fuerza_arrastre   =   0.5   *   coeficiente_arrastre   *   area_frontal   *
velocidad_vehiculo(i+1)^2;
    resistencia_rodadura = coeficiente_resistencia_rodadura * masa * g;
    fuerza_longitudinal(i+1)   =   -fuerza_frenado(i+1)   -   fuerza_arrastre   -
resistencia_rodadura;
end

% Detener y liberar canal CAN
stop(canBus);

% Gráficos

% Definir el inicio y fin del ataque para el sombreado
inicio_ataque = 2; % Tiempo de inicio del ataque DoS
fin_ataque = 3;   % Tiempo de fin del ataque DoS

% Gráficos combinados: Velocidad del vehículo y velocidad de la rueda
figure;
hold on; % Mantener el gráfico para superponer
% Rellenar el área del ataque
fill([inicio_ataque fin_ataque fin_ataque inicio_ataque], ...
     [0 0 max(velocidad_vehiculo) max(velocidad_vehiculo)], ...
     'r',  'FaceAlpha',  0.2,  'EdgeColor',  'none',  'DisplayName',  'Periodo  de
Ataque DoS');
plot(t, velocidad_vehiculo, 'g', 'LineWidth', 1.5, 'DisplayName', 'Velocidad del
vehículo (m/s)');
plot(t, velocidad_rueda * radio_neumatico, 'b', 'LineWidth', 1.5, 'DisplayName',
'Velocidad de la rueda (m/s)');
xlabel('Tiempo (s)');
ylabel('Velocidad (m/s)');
title('Velocidad del Vehículo y de la Rueda vs. Tiempo');
legend('show');
grid on;
hold off; % Liberar el gráfico para otras operaciones

% Gráfico 2: Relación de Deslizamiento
figure;
hold on;
% Rellenar el área del ataque
fill([inicio_ataque fin_ataque fin_ataque inicio_ataque], ...
     [0 0 1.1 1.1], ...
     'r',  'FaceAlpha',  0.2,  'EdgeColor',  'none',  'DisplayName',  'Periodo  de
Ataque DoS');
plot(t,  deslizamiento,  'm',  'LineWidth',  1.5,  'DisplayName',  'Relación  de
Deslizamiento');
xlabel('Tiempo (s)');
ylabel('Relación de Deslizamiento');
title('Relación de Deslizamiento de la Rueda vs. Tiempo');
ylim([0, 1.1]);
legend('show');
grid on;
hold off;
```

```
% Gráfico 3: Fuerza de Frenado del Actuador
figure;
hold on;
% Rellenar el área del ataque
fill([inicio_ataque fin_ataque fin_ataque inicio_ataque], ...
     [min(fuerza_frenado_actuador)                  min(fuerza_frenado_actuador)
max(fuerza_frenado_actuador) max(fuerza_frenado_actuador)], ...
     'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none', 'DisplayName', 'Periodo de
Ataque DoS');
plot(t, fuerza_frenado_actuador, 'k', 'LineWidth', 1.5, 'DisplayName', 'Fuerza de
Frenado');
xlabel('Tiempo (s)');
ylabel('Fuerza de Frenado (N)');
title('Fuerza de Frenado del Actuador vs. Tiempo');
legend('show');
grid on;
hold off;

% Grafico 4: Contandor de mensajes
figure;
hold on;
% Rellenar el área del ataque
fill([inicio_ataque fin_ataque fin_ataque inicio_ataque], ...
     [min(fuerza_frenado_actuador)                  min(fuerza_frenado_actuador)
max(fuerza_frenado_actuador) max(fuerza_frenado_actuador)], ...
     'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none', 'DisplayName', 'Periodo de
Ataque DoS');
plot(t, mensajes_transmitidos, 'r', 'LineWidth', 1.5, 'DisplayName', 'Mensajes
CAN transmitidos');
xlabel('Tiempo (s)');
ylabel('Numero de mensajes transmitidos');
title('Flujo de Datos en el Bus CAN');
legend('show');
grid on;
hold off;
```

ANEXO E

Función de MATLAB, Simulink CAN ID

```
function Valor = canID(ID, Activacion)
    % controlID: Devuelve el ID si Activacion es 1, de lo contrario devuelve 0.
    % Entradas:
    %   - ID: Valor del ID a procesar.
    %   - Activacion: Indicador de activación (0 o 1).
    % Salida:
    %   - output: ID si Activacion es 1, NaN si Activacion es 0.

    if Activacion == 1
        Valor = ID;
    else
        Valor = 0;
    end
end
```

ORDEN DE EMPASTADO