**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**Facultad de Ingeniería en Electricidad y Computación**

# Diseño conceptual y simulación de un sistema de alerta vehicular para la prevención de colisiones mediante el análisis en tiempo real de datos de sensores, desarrollado en la plataforma Raspberry Pi.

**PROYECTO INTEGRADOR**

Previo la obtención del Título de:

**Ingeniero en Electrónica y Automatización**

Presentado por:

Josue Enrique Tinoco Salvatierra

GUAYAQUIL - ECUADOR

Año: 2021

# DEDICATORIA

Este proyecto va dedicado a mi familia, por el apoyo incondicional y la motivación de siempre, y por haberme ofrecido el mejor regalo que se le puede dar a un hijo, la educación.

# AGRADECIMIENTOS

Agradezco encarecidamente al Ing. Dennys Cortez, profesor de materia integradora, por su apreciada retroalimentación semana a semana en los avances constantes presentados de la presente investigación.

También se agradece al Ing. Ronald Solís, tutor de este proyecto integrador, por los consejos y guía en la elaboración del presente proyecto.

# DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, me corresponde conforme al reglamento de propiedad intelectual de la institución; *Yo, Josue Enrique Tinoco Salvatierra* doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Autor

# EVALUADORES

Firmado electrónicamente por:
DENNYS DICK CORTEZ ALVAREZ

**Dennys Cortez**

PROFESOR DE LA MATERIA

**Ronald Solís**

PROFESOR TUTOR

# RESUMEN

El presente trabajo surge a partir del muy limitado uso de tecnologías de prevención de colisión dentro de los vehículos comercializados en el mercado local y la carencia de productos comerciales de esta índole. Por esta razón, se propuso el desarrollo de un sistema de alerta de colisión frontal que influya sobre la seguridad vehicular y el bienestar del conductor. Este integra hardware y software mediante la plataforma Raspberry Pi; se analizan datos de sensores, se programan algoritmos, y se crea un entorno de simulación para comprobar el funcionamiento del sistema.

Se diseñó el uso y conexión de sensores tal que puedan comunicarse con el Raspberry Pi, mediante compatibles protocolos de comunicación. Por otra parte, se convirtió una serie de ecuaciones y condicionales preexistentes, en código Python ejecutable con la plataforma utilizada. Se creó escenarios de conducción empleando el software Matlab, donde se pudo simular datos y adaptarlos a los parámetros de los sensores propuestos.

Se logró obtener una óptima configuración del código de adquisición de datos mediante varias iteraciones de pruebas que sirvieron para maximizar el tiempo de respuesta del sistema, y minimizar el error presente en las variables cinemáticas. Se comparó el rendimiento del sistema usando los escenarios de conducción, obteniendo resultados satisfactorios en cada uno.

El sistema genera efectivamente alertas cuando el vehículo se encuentra en curso de colisión, ofreciendo al conductor tiempo suficiente para reaccionar y evitar accidentarse, siempre y cuando se encuentre dentro de los márgenes de velocidad y distancia mínima diseñados para el sistema.

**Palabras Clave:** Sistema de alerta, Prevención de colisión, Integración, Bienestar

# *ABSTRACT*

*This work arises from the limited use of collision avoidance technologies in vehicles sold on the local market, as well as the lack of commercial products of this nature. For this reason, the development of a frontal collision warning system that influences vehicle safety and driver well-being was proposed. This integrates hardware and software through the Raspberry Pi platform; sensor data is analyzed, algorithms are programmed, and a simulation environment is created to test the operation of the system.*

*The use and connection of sensors was designed so that they can communicate with the Raspberry Pi through compatible communication protocols. On the other hand, a series of pre-existing equations and conditionals were converted into executable Python code to be used with the platform. Driving scenarios were created using Matlab software, where data could be simulated, and the parameters of the proposed sensors, adapted.*

*It was possible to obtain an optimal configuration of the data acquisition code through several iterations of tests that served to maximize the response time of the system and minimize the error present in the kinematic variables. The performance of the system was compared using the driving scenarios, obtaining satisfactory results in each one.*

*The system effectively generates alerts when the vehicle is on a collision course, offering the driver enough time to react and avoid an accident, as long as he is within the speed and minimum distance margins designed for the system.*



*Keywords: Alert System, Collision Avoidance, Integration, Well-being*

# ÍNDICE GENERAL

# ABREVIATURAS

| | |
|---|---|
| ESPOL | Escuela Superior Politécnica del Litoral |
| FCW | Front Collision Warning |
| SUV | Sport Utility Vehicle |
| LIDAR | Light Detection and Ranging |
| LED | Light Emitting Diode |
| TFT-LCD | Thin Film Transistor Liquid Crystal Display |
| GPS | Global Positioning System |
| TTC | Time to Collision |
| NHTSA | National Highway Traffic Safety Administration |
| JHU-APL | John Hopkins University Applied Physics Laboratory |
| RPi | Raspberry Pi |
| GPIO | General Purpose Input Output |
| UART | Transmisor Receptor Asíncrono Universal |
| SPI | Interfaz de periféricos serial |
| I2C | Inter Integrated Circuit |
| ADAS | Advanced Driving Assistance Systems |
| DSD | Driving Scenario Designer (MATLAB) |

# SIMBOLOGÍA

| | |
|---|---|
| km/h | Kilómetros por hora |
| m | Metro |
| s | Segundo |
| m/s | Metro por segundo |
| v | Velocidad |
| Vr | Velocidad relativa |
| a | Aceleración |
| Ar | Aceleración relativa |
| V | Voltaje / Voltio |
| A | Amperio |
| W | Vatio |

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# CAPÍTULO 1

## 1. INTRODUCCIÓN

### 1.1 Descripción del problema

Tan solo un breve momento de distracción por parte de los conductores puede llegar a ser el causante de colisiones y todo tipo de accidentes vehiculares. Dichos segundos en el que se pierde la visión de la calle y los alrededores, deberían de ser monitoreados por un sistema inteligente, que analice los parámetros físicos de los alrededores del vehículo, tal que tenga la capacidad de alertar, con prudente tiempo de anticipación, un posible evento de colisión.

En este proyecto se pretende resolver la carencia de productos comerciales en la región, para tecnologías de prevención de colisión vehiculares. Por tal motivo se requiere del diseño de un sistema que conste de un procesador lógico computacional, una serie de sensores para obtener datos de parámetros cinemáticos, y una plataforma de software con algoritmos y programas, tal que se logre la integración de todas las partes.

Se plantea incorporar múltiples enfoques de investigaciones relacionadas, ya sea en el uso de sensores, o métodos y algoritmos, para crear un sistema robusto en el cual se demuestre el proceso de adquisición de datos y el rendimiento del Raspberry Pi. También se demostrará la funcionalidad de la lógica de toma de decisiones implementada en el código, haciendo uso de un entorno de simulación de cinemática vehicular.

## 1.2 Justificación del problema

Actualmente el uso de sistemas de prevención de colisión vehicular es muy limitado en el mercado ecuatoriano. Debido a los altos impuestos y costos de importación, los fabricantes de automóviles deciden remover estas tecnologías para abaratar precios y que el producto final sea más asequible a los clientes. Por esta razón, existe una notoria escasez de estas tecnologías en los vehículos ofertados en Ecuador.

Según las estadísticas de siniestros de tránsito proporcionadas por la Agencia Nacional de Tránsito del Ecuador, entre el año 2008 hasta el 2020 se ha producido un promedio de 26,471 accidentes automovilísticos anuales, de los cuales se ha categorizado que los choques por alcance, sobre los cuales funciona el sistema de alerta propuesto, han alcanzado hasta un 13% del total de dichos accidentes. Tomando en cuenta sólo los accidentes en los que dos vehículos están involucrados, esta cifra aumenta al 25%. En septiembre del 2021 del total de choques por alcance, el 22% presentó lesionados o fallecidos y el 78% restante solo presentó daños. Sobre estos datos se basa la importancia de la propuesta actual, que busca la prevención no sólo de pérdidas económicas debido a los accidentes mencionados, sino también de lesiones o pérdidas de vidas en estos siniestros.

Por esta razón se propone una solución viable económicamente, aplicable a la mayoría de los vehículos que circulan las calles en el país, tal que exista un beneficio en el bienestar y la seguridad automovilística. Se ha dispuesto diseñar un sistema, de fácil desarrollo e implementación, que sirva para advertir al conductor de un vehículo, por medio de alertas, las situaciones en las que se aproxime un posible evento colisión, y de esta manera influir en la reacción del conductor a la prevención de un accidente, en el mejor de los casos, o por lo menos disminuir la gravedad del impacto.

## 1.3 Objetivos

### 1.3.1 Objetivo General

- Diseñar y simular un sistema de alerta vehicular mediante el análisis de datos de sensores en tiempo real para la prevención de colisiones.

### 1.3.2 Objetivos Específicos

- Configurar la placa de ordenador Raspberry Pi tal que procese y analice los datos de sensores en sus entradas GPIO.
- Programar mediante el lenguaje Python códigos y algoritmos que permitan una adecuada adquisición y procesamiento de datos, y generación de alertas.
- Crear un entorno de simulación para los datos de sensores y algoritmos, sobre el cual se puedan extraer gráficas de datos y variables cinemáticas, y sea posible probar el funcionamiento general del sistema.
- Analizar la efectividad del sistema para diferentes iteraciones de configuración y/o programación.

## 1.4 Enfoque del Proyecto

El enfoque de este proyecto se basa en varios aspectos técnicos y de diseño, los cuales pueden asociarse de acuerdo a: simplicidad, factibilidad y rentabilidad. Naturalmente, las decisiones tomadas en la realización de la presente investigación se dan a cabo siempre y cuando presenten concordancia con las tres características de enfoque mencionadas.

Con respecto a la simplicidad, surge la elección de que el sistema sea totalmente independiente a los sistemas internos del vehículo en el cual estará instalado, lo que también incluye evitar cualquier tipo de instalación intrusiva en la carrocería. Por esta razón, la ubicación de unidad encargada de detectar la distancia al vehículo delantero está pensada para colocarse en el centro del panel frontal del vehículo, paralelo al plano de la calle. Consecuentemente, este sistema de prevención de colisión estará dirigido únicamente para vehículos de pequeña o mediana altura, de tipo compacto, sedán, familiar, o SUV con un límite máximo de altura de 1.4m, tomada desde la calle hasta el

panel frontal. Esto es debido a que si el sistema se implementase en vehículos que sobrepasan este valor, no se detectarían a los automóviles delanteros que su techo se encuentre por debajo del rayo de detección del sensor empleado.

Con respecto a la factibilidad, el actual proyecto se dispone a estandarizar los datos asociados a la capacidad de frenado usado en el algoritmo de prevención de colisión, obviando factores como peso del vehículo, su aerodinámica, el estado de las llantas, entre otros factores, a excepción de la fricción del suelo, que se determina en caso de la presencia o no de lluvia. Estos valores son estandarizados teniendo en cuenta la increíblemente gran cantidad de modelos de vehículos en el mercado, que conllevaría al inevitable requerimiento de crear una base de datos y una interfaz humano máquina para que se escojan los parámetros asociados a los factores mencionados, antes de cada instalación, procedimientos que no han sido planificados en la realización del presente proyecto. De esta manera, este se vuelve mucho más factible en el sentido a que se puede diseñar y comprobar su funcionamiento acorde a plazos de tiempo predestinados.

Por último, en términos de rentabilidad, para cumplir la viabilidad económica, este proyecto se enfoca simplemente en la prevención de colisión de acuerdo a la detección de vehículos. Para detectar peatones, ciclistas o motociclistas, se requeriría de una inversión adicional en sistemas de fusión de sensores, algoritmos mucho más complejos y un mayor tiempo de desarrollo. Básicamente el costo total del sistema superaría de tal forma que formaría una contradicción con la propia justificación del proyecto.

## 1.5 Marco teórico

### 1.5.1 Sistemas de prevención de colisión vehicular

Un sistema de prevención de colisión vehicular es aquel que obtiene información sobre diversos parámetros físicos del ambiente que rodea al vehículo, y analiza dicha información con la finalidad de tomar decisiones autónomas o de advertencia para prevenir o mitigar posibles colisiones. Cabe recalcar que a pesar de que se mencione la palabra “prevenir”, ninguno de estos sistemas tiene la capacidad de evitar colisiones en todas las situaciones que puedan surgir, pero se espera que en la mayoría de los casos se reduzca la magnitud del accidente lo mayor posible.

Existen tres fases sobre las que funciona un sistema de prevención de colisión.

- Percepción
- Decisión
- Acción

La fase de percepción equivale a la recopilación de información a base de sensores. La fase de decisión se refiere al análisis de los datos proporcionados por los sensores mediante un dispositivo de procesamiento, tal como un microcontrolador o un microprocesador, en las que un algoritmo de software decide si se requiere enviar una señal de intervención o advertencia. La fase de acción sucede cuando la señal recibida se emplea para generar algún tipo de alerta física para llamar la atención del conductor, la cual puede tener la intención de informar o advertir.

#### *1.5.1.1 Ejemplos de sistemas*

Existen varios sistemas de prevención de colisión con una implementación dirigida a diferentes escenarios de los cuales un vehículo puede estar expuesto. Los ejemplos que se muestran a continuación corresponden a acciones reactivas en base a alertas y advertencias.

- **Advertencia de colisión frontal (FCW):** Genera una alerta visual y/o sonora para indicar al conductor que se ha detectado un posible evento futuro de colisión. Por

lo general las alertas se clasifican en nivel dependiendo del nivel de riesgo de acuerdo a distancia o velocidad, desde seguro hasta crítico.

- **Advertencia de punto ciego (BSW):** Genera un indicador visual, usualmente instalado en los retrovisores de los lados, para indicar al conductor que un vehículo se encuentra en el punto ciego del carril adyacente. En caso de que se active el indicador de cambio de carril, se genera un indicador sonoro de alerta.
- **Advertencia de tráfico cruzado (CTW):** Una advertencia que se produce cuando el vehículo se encuentra en movimiento de reversa, y se detecta que se aproxima un vehículo en dirección perpendicular a este.

El segundo grupo de sistemas que se muestra a continuación corresponde a acciones activas que requieren el condicionamiento del vehículo desde su producción en fábrica, para que posea controles de aceleración o frenado tanto hidráulicos como electrónicos.

- **Control de crucero adaptivo (AAC):** Usa los mismos datos que un sistema FCW, pero en este caso el vehículo tiene control sobre el acelerador y los frenos para mantener una velocidad y distancia segura con el vehículo que tenga en frente.
- **Frenado de emergencia automático (AEB):** Equivale a un sistema FCW, pero con la funcionalidad adicional que en la generación de alerta crítica el vehículo aplica los frenos evitar la colisión o minimizar el impacto de forma autónoma.

#### *1.5.1.2 Disponibilidad en el mercado ecuatoriano*

Por lo general en el Ecuador sólo se ofrecen sistemas básicos, como asistente de parqueo o un sistema alerta en punto ciegos. Estos usan sensores ultrasónicos, y radar de corto alcance, respectivamente, los cuales son significativamente más económicos que los sensores empleados para un sistema de alerta de colisión frontal (FCW), el cual usaría un radar de medio o largo alcance, LIDAR o cámaras. Por este motivo, los vehículos que vienen de fábrica con este tipo de sistema son encontrados exclusivamente en modelos de gama media o alta.

A continuación, se presenta una lista de algunos modelos de vehículos y sus precios, ofrecidos en el país, que sí cuentan con un sistema FCW propio de cada marca.

**Tabla 1.1. Listado de algunos vehículos en Ecuador con sistema FCW.**

| Marca | Modelo | Precio |
|---|---|---|
| Toyota | Corolla Hybrid | $34,990 |
| Mazda | CX-5 | $36,990 |
| Nissan | Leaf | $39,990 |
| Chevrolet | Blazer RS | $69,990 |
| Mercedes Benz | Clase E | $89,990 |
| Porsche | Cayenne S | $149,990 |

Como se observa en el listado anterior, el vehículo más económico con un sistema FCW tiene un precio de $34,990, contrastando con que el promedio del costo de los 10 vehículos más vendidos en el país es de aproximadamente $20,500. De esta manera podemos asegurar que existe una brecha de mercado para un sistema FCW externo que ofrezca las mismas capacidades que el fabricante automovilístico.

### 1.5.2 Sistema de alerta de colisión frontal (FCW)

El sistema FCW en general consiste en un vehículo equipado con sensores, una plataforma informática para adquisición y procesamiento de datos, así como elementos indicadores y de alerta. El elemento más crucial del sistema es el sensor frontal, que adquiere información sobre la distancia y velocidad con respecto al vehículo delantero.

Existen otros elementos secundarios cuya función es la de suplementar a los datos obtenidos por el sensor frontal, crear redundancia y tratar de hacer más robusto al sistema. La implementación es estos elementos secundarios, que corresponde casi exclusivamente a cámaras o sistemas de visión, son tecnologías modernas asociadas a aprendizaje profundo y redes neuronales, las cuales requieren de mayor poder computacional y algoritmos complejos los cuales se escapan del enfoque de esta investigación.

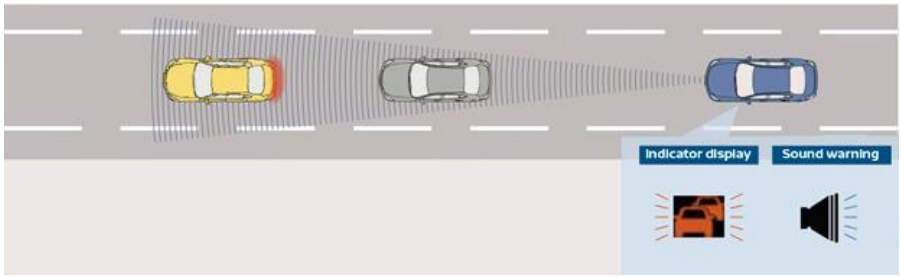



**Ilustración 1.1. Principio sistema FCW**

Todo sistema FCW, al detectar un posible evento de colisión frontal, alerta al conductor a una distancia adecuada tal que este pueda reaccionar a tiempo aplicando los frenos o cambiando la dirección del vehículo. Normalmente dicha alerta se transmite al conductor mediante algún tipo de interfaz humano máquina (HMI), en la cual un ejemplo es un aviso visual empleando una tira LED roja reflejada en el parabrisa, la cual imita las luces de freno del vehículo en frente. Se activa también una advertencia sonora cuando la acción de frenado debe de darse inmediatamente. También en ciertos modelos, el sistema se integra con la pantalla TFT LCD ubicada en el tablero del vehículo,


**Ilustración 1.2. Integración sistema FCW con pantalla TFT LCD del tablero vehicular.**

### 1.5.3 Obtención de variables para un sistema FCW

Las variables necesarias para el funcionamiento de un sistema FCW son las siguientes:

- Velocidad real: Velocidad del vehículo anfitrión con referencia al suelo
- Distancia: Espacio lineal entre el vehículo anfitrión al vehículo delantero.
- Velocidad Relativa: Velocidad del vehículo anfitrión con referencia al vehículo delantero.

- Aceleración: Variación de la velocidad del vehículo anfitrión con referencia al suelo.

Para obtener la **velocidad real** se emplea un módulo GPS y se calcula la velocidad de acuerdo a la tasa de cambio de posición con respecto a las coordenadas obtenidas.

Para obtener la **distancia** hacia el objeto en frente, así como la **velocidad relativa,** se emplea un sensor de distancia. Este sensor cuenta con un chip interno que procesa y filtra los datos, de acuerdo a su tecnología interna, para enviar una señal en la salida mediante protocolo serial con la información de la distancia obtenida.

Para un sistema FCW se utiliza un sensor de distancia de medio o largo alcance, que según [6] debería tener como requerimiento un rango de hasta 200m, en caso de que el sistema se diseña para funcionar en velocidades de hasta 65 mph (105 km/h). También como recomendaciones debería de tener una resolución menor a 1 metro, una tasa de rango menor a 0.25 m/s, una tasa de subida de datos mayor a 10 Hz, entre otros.

Para obtener datos de **aceleración** de acuerdo al accionamiento de los pedales del vehículo, se hace uso de un módulo acelerómetro-giroscopio.

### 1.5.4 Módulos y sensores para un sistema FCW

Para que un sistema FCW sea lo suficientemente robusto, aparte del sensor de distancia que es el elemento crucial, también se emplean diversos módulos y sensores tal que se pueda obtener información adicional tanto del propio vehículo, como de las condiciones del ambiente. Esto es debido a que ambas influyen de cierta forma en el comportamiento del conductor, su capacidad de reacción, y la distancia de frenado del automóvil.

#### *1.5.4.1 Sensor LIDAR*

Light Detection and Ranging (LIDAR), o Detección de Luz y Rango, es un dispositivo usado para obtener la distancia exacta de un objeto mediante el uso de láseres y del principio Time of Flight (TOF), o Tiempo de Vuelo. Es decir, que mediante una serie de pulsos, se calcula el tiempo de regreso del láser al receptor de acuerdo a la velocidad de la luz, tal que se obtiene un valor de distancia al objeto detectado.

Existen algunos tipos de LIDAR, desde el más complejo como lo es el LIDAR 3D 360° usado para mapeo y creación de entornos 3D de puntos virtualizados, hasta el más sencillo, tal como lo es el LIDAR “Single-Point Rangefinder” usado para obtener la distancia más cercana a un objeto de forma lineal y con un bajo campo de visión horizontal. Este último es el tipo elegido para el sistema propuesto FCW, por lo que se empleará el LIDAR TF-LUNA como un sensor equivalente para la comprobación de adquisición de datos, pero proponiendo el TF02-Pro o el TF03-100 para una implementación.

#### *1.5.4.2 Acelerómetro*

El acelerómetro es un sensor capaz de detectar las fuerzas de aceleración que actúan sobre el mismo. Para el sistema FCW, solo se requiere obtener la información de aceleración en el eje paralelo al vector de velocidad del vehículo. Fuerzas externas como la centrípeta no son de utilidad en los cálculos dentro de los algoritmos a utilizar. Se tomará en consideración los datos y comunicación del modelo MPU-6050, un acelerómetro giroscopio de 3 ejes y 6 grados de libertad.

#### *1.5.4.3 Módulo GPS*

El Sistema de Posicionamiento Global (GPS), es una tecnología capaz de brindar datos de posición, velocidad y altitud mediante la comunicación de un receptor hacia los satélites en órbita más cercanos a este. El módulo GPS pensado para una implementación sería el NEO-6M, un dispositivo miniaturizado de alta sensibilidad y bajo consumo energético. Es totalmente compatible con Raspberry Pi, en cuanto a su voltaje compatible de 3.3V y la presencia de librerías para la traducción de los datos recibidos por el módulo.

#### *1.5.4.4 Sensor de lluvia*

Para detectar la presencia de lluvia, se podrá usar cualquier sensor genérico de lluvia que produzca una señal digital alta en caso de obtener una lectura positiva. Estos sensores son colocados con adhesivo en el parabrisas del automóvil, o sujetados justo en el panel exterior debajo de este. Luego se traspasa el cable de poder y señal al controlador dentro del vehículo.

**Ilustración 1.3. Ejemplo de sensor de lluvia instalado en la base del parabrisas**

### 1.5.5 Algoritmos FCW basados en percepción

Los algoritmos actuales para alerta de colisión se encuentran clasificados en dos categorías, cinemática y percepción.

El modelo de percepción realiza una comparación entre el tiempo de colisión (TTC) de dos objetos, con un umbral de tiempo determinado para obtener el nivel de seguridad o peligrosidad. En estos algoritmos se ofrecen constantes o valores recomendados, producto de experimentaciones reales llevadas a cabo por los creadores de estos. El valor de TTC equivale a el tiempo restante para que ocurra una colisión entre dos vehículos en caso de que la diferencia de velocidad se mantuviera igual.

$TTC = \frac{D}{v_r}$, siendo D la distancia entre ambos vehículos y $v_r$ la velocidad relativa.

#### *1.5.5.1 Algoritmo de Honda*

Es una sencilla ecuación basada en datos obtenidos mediante una serie de experimentos elaborados por la compañía automovilística Honda. En este algoritmo se alerta al conductor cuando el TTC se iguale al tiempo crítico de 2.2 segundos. Por lo tanto, el rango de alerta se calcula de la siguiente manera:

$$R_a = 2.2\ v_r + 6.2$$

Siendo el valor de 6.2 una distancia de seguridad adicional.

Este algoritmo también presenta una parte de prevención (rango de prevención $R_p$), la cual corresponde a una función definida en partes, dependiendo de que si el tiempo de detención del vehículo delantero es mayor o menor al tiempo de reacción del conductor del vehículo anfitrión [11].

$$R_p = \begin{cases} v_r\tau_2 + \mathrm{t}_1\mathrm{t}_2\mathrm{a}_1 - 0.5a_1\mathrm{t}_1^2\,; & \dfrac{(v-v_r)}{a_2} \geq t_2 \\ vt_2 - 0.5a_1(t_2-t_1)^2 - \dfrac{(v-v_r)^2}{2a_2}; & \dfrac{(v-v_r)}{a_2} < t_2 \end{cases}$$

Los parámetros recomendados para esto caso son:

| Variable | Valor | Unidad |
|---|---|---|
| Aceleración vehículo anfitrión ($a_1$) | 7.8 | m/s^2 |
| Aceleración vehículo delantero ($a_2$) | 7.8 | m/s^2 |
| Tiempo $t_1$ | 0.5 | s |
| Tiempo $t_2$ | 1.5 | s |

### 1.5.6 Algoritmos FCW basados en cinemática

En el modelo cinemático se calcula la distancia mínima entre el vehículo ego y el delantero, tal que no sea sobrepasada para que no se produzca una colisión tomando en los parámetros cinemáticos en tiempo real tanto del vehículo ego como del vehículo delantero[9]. La precisión en la generación de alerta por parte de estos algoritmos es mayor, aunque requieren de un constante procesamiento de información de múltiples sensores que envíen datos fiables al sistema FCW.

#### *1.5.6.1 Algoritmo NHTSA – JHU/APL*

Es un algoritmo desarrollado por la Administración Nacional de Seguridad del Tráfico en las Carreteras (NHTSA) de los EE. UU., en colaboración con el Laboratorio de física aplicada (APL) de la universidad de John Hopkins (JHU). Utiliza algunas de las variables de los algoritmos ya vistos anteriormente, más la adición de otras que se verán más adelante. Emplea un parámetro ajustable para modificar la sensibilidad del sistema. Su

rendimiento ha sido examinado de acuerdo a contextos operativos, los cuales incluyen tres casos específicos:

- Vehículo ego se dirige a velocidad constante hacia un vehículo delantero estacionario.
- Vehículo ego se dirige a velocidad constante hacia un vehículo delantero a menor velocidad.
- Vehículo ego se dirige a velocidad constante hacia un vehículo delantero que se encuentre aplicando los frenos.

Con este algoritmo es posible tener 4 niveles de alerta, siendo el más bajo “sin advertencia”, y el más alto para “alto peligro de colisión”. Los 2 niveles restantes son advertencias intermedias para proveer alertas no intrusivas al conductor, en caso de que el vehículo ego requiera desacelerar a una tasa menor.

La siguiente tabla enlista todas las variables y parámetros requeridos por el algoritmo.

| **Variable** | **Símbolo** | **Obtenido por** |
|---|---|---|
| Velocidad vehículo anfitrión | $V_1$ | Sistema FCW |
| Aceleración vehículo anfitrión | $A_1$ | Sistema FCW |
| Velocidad vehículo delantero | $V_2$ | Suma de $v_1$ y $v_r$ |
| Rango | $R$ | Sistema FCW |
| Tasa de rango (Velocidad relativa) | $V_R$ | $\Delta R/\Delta t$ |
| Aceleración relativa | $A_R$ | $\Delta V_R/\Delta t$ |
| Aceleración vehículo delantero | $A_2$ | Suma de $A_1$ y $A_r$ |
| Tiempo de reacción | $T_R$ | Recomendado 1.5 [s] |
| Asumida capacidad máxima de frenado por vehículo anfitrión | $A_{1MAX}$ | 0.6g, a un factor de 1, 0.75 y 0.5 |
| Tiempo detención vehículo delantero | $T_{LS}$ | Algoritmo |
| Tiempo detención vehículo anfitrión | $T_{HS}$ | Algoritmo |

Primero se calcula $D_{tresh}$, o distancia de umbral, que tiene una componente fija de 2m y otra variable dependiendo de la velocidad del vehículo. Este valor corresponde a la

distancia máxima que se desea que quede entre el vehículo ego y el delantero en caso de actuar frente a una alerta.

$$D_{tresh} = 2\ [m] + (V_1)(0.1\ [s])$$

Para los tiempos de detención se aplican las siguientes ecuaciones:

$T_{LS} = -\frac{V_2}{A_2}$

$T_{HS} = T_R - \frac{V_1 + A_1 T_R}{A_{1MAX}}$

$D_{miss}$ o distancia de elusión, corresponde a la distancia máxima real que quedaría entre el vehículo ego y el delantero luego de actuar frente a una alerta de colisión. Es calculada de la siguiente manera:

**Para el caso $T_{LS} \geq T_R$:**

$D_{miss} = R + \Delta R1 + \Delta R2 + \Delta R3$

El algoritmo emplea la ecuación cinemática para desplazamiento: $\Delta x = v_0 t + \frac{1}{2} a t^2$

$\Delta R1 = V_R T_R + \frac{1}{2}(A_2 - A_1)(T_R)^2$

$\Delta R2 = (V_R + (A_2 - A_1)T_R)(T_{LS} - T_R) + \frac{1}{2}(A_2 - A_{1MAX})(T_{LS} - T_R)^2$

$\Delta R3 = [(V_R + (A_2 - A_1)T_R) + (A_2 - A_{1MAX})(T_{LS} - T_R)](T_{HS} - T_{LS}) + \frac{1}{2}(-A_{1MAX})(T_{HS} - T_{LS})^2$

Cada vez que se alarga la ecuación, es debido a que se está calculando el nuevo valor de velocidad ($v_0$) para el periodo de tiempo correspondiente.

Simplificando se obtiene:

$$D_{miss} = R + \frac{1}{2}(A_1 - A_{1max})(T_R)^2 - \frac{1}{2}(A_2)(T_{LS})^2 - (A_1 - A_{1\max})T_R T_{HS} + V_R T_{HS} + A_2 T_{HS} T_{LS} - \frac{1}{2}(A_{1max})(T_{HS})^2$$

Cuando el vehículo delantero no va a detenerse o su detención se da en un tiempo menor al tiempo de reacción, se simplifica el cálculo de la distancia de elusión eliminando un periodo de tiempo. Para este caso, se considera colisión evitada cuando la velocidad relativa llega a cero, a una distancia segura de umbral.

**Para el caso $T_{LS} < T_R$:**

$$D_{miss} = R + \Delta R1 + \Delta R2$$

$$\Delta R1 = V_R T_R + 0.5(A_2 - A_1)(T_R)^2$$

$$\Delta R2 = (V_R + (A_2 - A_1)T_R)(T_M - T_R) + 0.5(A_2 - A_{1MAX})(T_M - T_R)^2$$

$$T_M = \frac{(V_R + (A_2 - A_1)T_R)}{(A_{1MAX} - A_2)} + T_R$$

Simplificando se obtiene:

$$D_{miss} = R + V_R T_M + \frac{1}{2}(A_2 - A_{1max})(T_M)^2 - (A_1 - A_{1\max})T_M T_R + \frac{1}{2}(A_1 - A_{1max})(T_R)^2$$

En caso de que la distancia de elusión llegue a ser menor que la distancia de umbral ($D_{miss} < D_{umbral}$), el algoritmo propone que se genere una alerta sólo si esta condición se encuentre presente en 2 de los últimos 3 intervalos de tiempo. Es decir, el algoritmo genera un tiempo de activación adicional de 200 ms.

### 1.5.7 Descripción Raspberry Pi (RPi)

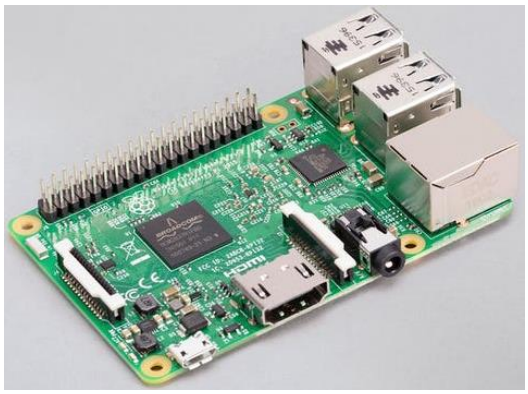


**Ilustración 1.4. Placa Raspberry Pi 3 Modelo B.**

El Raspberry Pi es un minicomputador elaborado por la fundación del mismo nombre, diseñado para ser empleado bajo un entorno abierto en la programación y elaboración de proyectos de todo tipo. Cuenta con un sistema operativo interno basado en Linux, y soporta una gran variedad de lenguajes de programación, siendo el principal Python, y con C/C++, así como Scratch, también presentes por defecto en el OS.

El modelo del cual se basa el presente proyecto es el Raspberry Pi 3 B, el cual posee una unidad de procesamiento central ARM Broadcom de cuatro núcleos, la cual funciona a una frecuencia de 1.2Ghz. Sus especificaciones mencionan una memoria RAM de 1GB, la inclusión de WIFI, Bluetooth, y varios puertos de comunicación serial y digital para la integración de una amplia gama de dispositivos.

Es un sistema embebido de bajo consumo energético, a tan solo 260 mA a 5 VDC. A pesar de esto, tiene la capacidad de alimentar dispositivos externos hasta un máximo de 2.5A, lo cual tiene la ventaja de que no es requerido recurrir a una fuente de alimentación externa para los dispositivos de entrada o salida que consumen en total una cantidad menor a dicho amperaje máximo.

### 1.5.8 Uso de los terminales GPIO de RPi

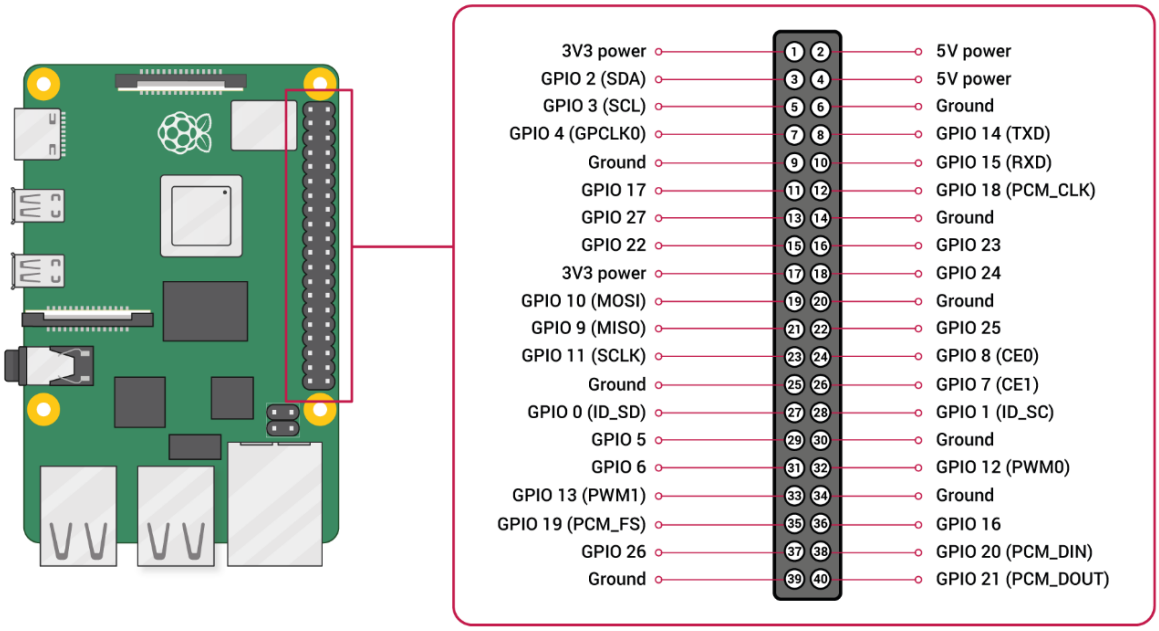



**Ilustración 1.5. Disposición y descripción de los terminales GPIO del Raspberry Pi 3 B.**

Una de las características más importantes que posee el Raspberry Pi, es su fila de terminales GPIO (General Purpose Input Output) o Entradas y Salidas de Propósito General, localizadas en el borde de la placa. Todos los modelos de Raspberry Pi desde el lanzamiento del Pi 1 Modelo B+ en 2014 traen consigo la misma cantidad de terminales con las mismas funciones, protocolos e interfaces.

Cualquiera de los pines GPIO puede ser designado mediante software a ser usado como entrada o como salida, a excepción de los pines de alimentación 3V3, 5V y GND. Cualquier pin GPIO que funcione como salida puede ser dispuesto como HIGH 3.3V o como LOW 0V. Asimismo, un pin GPIO que funcione como entrada detectará un estado lógico HIGH y LOW a 3.3V y 0V, respectivamente. Esta función es facilitada por la presencia de resistores de pull-up o pull-down configurables en la programación.

Tomando en consideración que ciertos terminales se encuentran nativamente configurados para ser compatibles con una función en particular, se detallará a continuación dichas funciones y los pines con las que son compatibles:

| Función | Categoría | Pines GPIO |
|---|---|---|
| PWM | Software | Todos |
| | Hardware | 12, 13, 18, 19 |
| SPI | SPI0 | 10, 9, 11, 8, 7 |
| | SPI1 | 20, 19, 21, 18, 17, 16 |
| I2C | Data | 2, 3 |
| | EEPROM Data | 0, 1 |
| Serial | TX | 14 |
| | RX | 15 |

**Ilustración 1.6. Funciones de pines específicos GPIO.**

### 1.5.9 Protocolos de comunicación serial para RPi

#### *1.5.9.1 UART*

El Transmisor Receptor Asíncrono Universal (UART) es un protocolo de comunicación serial usado para el intercambio de datos entre dispositivos periféricos y un controlador. Solo requiere de dos líneas de señales, llamadas transmisión (TX) y recepción (RX), siendo estas la entrada y salida del sistema, respectivamente. UART funciona mediante la transmisión de datos a partir de la conversión de información digital dispuesta en bytes recibida de circuitos paralelos, en una secuencia de bits, controlados mediante bits de inicio, parada y paridad. La recepción de datos sigue el mismo proceso de manera inversa.

#### *1.5.9.2 SPI*

La interfaz de periféricos serial (SPI) es un protocolo de comunicación usado en aplicaciones donde el dispositivo requiere de la presencia del reloj, en casos de que sea crítica la comunicación sincrónica en la transferencia de datos. Funciona bajo una configuración de maestro y esclavo, en la cual el maestro sería el controlador, y los esclavos, que pueden ser varios, serían los dispositivos de periferia tales como módulos o sensores. Los bits de información se envían mediante 4 líneas de señales, siendo una exclusiva para el reloj y su sincronización (SCLK), dos de estas para la comunicación

entre maestro y esclavo (MOSI, MISO), y la última empleada exclusivamente para seleccionar o activar un esclavo (SS).

#### *1.5.9.3 I2C*

El Circuito Inter-Integrado (I2C), tal como SPI, es un protocolo de comunicación serial sincrónico que utiliza una configuración maestro esclavo. Sin embargo, este protocolo fue diseñado para la comunicación con dispositivos periféricos de baja velocidad. También hace uso de dos líneas de transmisión, de la cual tan sólo una es para la transferencia de datos (SDA), mientras que la otra funciona como la señal de reloj y sincronización (SCL).

### 1.5.10 Entorno de programación para RPi

Python es el lenguaje de programación por defecto para su uso general en cualquier aplicación desarrollada en el Raspberry Pi. Es un lenguaje de alto nivel, creado específicamente para ser de fácil uso, escritura y lectura, con una sintaxis clara e inspirada por palabras clave del idioma inglés.

El IDE recomendado para usarse en programas escritos para el Raspberry Pi es Thonny, un entorno de desarrollo basado en Python 3. Su ventaja principal es la sencillez de uso, ideal para desarrollar programas sin la necesidad de tener conocimientos avanzados en programación. Presenta un depurador simple, evaluador de expresiones por paso, resaltado de errores de sintaxis, finalización automática de código, entre otras funciones que facilitan en gran medida el trabajo del programador.

Para este proyecto, la programación en Python se requerirá para todos los procesos de la parte de diseño del sistema FCW, que consiste en la fase de adquisición de datos, y la fase de decisión/acción basada en las ecuaciones del algoritmo NHTSA. Para la simulación del sistema se requerirá traducir el código Python de la fase de decisión al lenguaje del software de simulación a utilizar.

# CAPÍTULO 2

## 2. METODOLOGÍA

### 2.1 Alternativas de solución

Existen diversas alternativas de solución para el diseño de un sistema de prevención de colisiones, que varían con respecto a la clase de sistema embebido a utilizarse, los tipos de sensores y sus características, así como los algoritmos de decisión a emplear en la programación de la lógica. Asimismo, la simulación de la funcionalidad y efectividad del sistema comprende una gran variedad de enfoques distintos, ya que existen diferentes herramientas o software de terceros sobre los cuales se puede implementar un proceso de simulación y obtención de resultados.

#### 2.1.1 Alternativa sistema embebido

A pesar de que el proyecto actual se enfoca en la utilización del producto Raspberry Pi, previamente se consideró en un sistema embebido alternativo que pudiese haber sido empleado en el diseño del actual sistema. Es de suma importancia aclarar dicha opción, ya se proporcionará una explicación del porqué no fue considerada desde el inicio.

La primera alternativa fue el uso de la tecnología FPGA (Field Programmable Gate Arrays) o matriz de puertas lógicas programables, un circuito integrado de hardware reconfigurable capaz de procesar señales digitales.

El problema surge en que esta tecnología hace uso de lenguaje de descripción de hardware (HDL), en el que el programador describe el hardware conectado al circuito y define las funciones de los bloques lógicos, pero la solución propuesta requiere de la creación de software mediante uso de lenguaje de alto nivel (C, Python, etc.), tal que se describan variables internas, ecuaciones, algoritmos, así como importación de librerías para protocolos de comunicación con los módulos y sensores a emplear.

Asimismo, la ventaja de las altas velocidades de procesamiento y envío/recepción de señales, son limitadas por los dispositivos de adquisición de datos del sistema, e incluso por el propio usuario de la solución. Por ejemplo, la tasa de refresco de envío de datos de un sensor LIDAR es por lo general de 100Hz, y el módulo GPS de entre 1-5Hz, es

decir que una velocidad de procesamiento alta no representa un requerimiento crítico del sistema. Con respecto al usuario, la limitación es su propio tiempo de reacción ($T_R$), por lo que si $T_R$ al estar sumado al tiempo de respuesta total del sistema, no representa un porcentaje de aumento significativo, la propuesta actual se consideraría aceptable.

### 2.1.2 Alternativas módulos y sensores

En cuanto a dos de las variables necesarias para el sistema FCW, la distancia y la velocidad relativa, se pueden emplear sensores LIDAR, de radar, o cámaras.

Como el proyecto actual se desarrollará en base a la obtención de datos de un sensor LIDAR, se detallarán las razones por la cual se descartó el uso de las dos alternativas restantes.

El sensor de radar es la tecnología más usada en los sistemas de prevención de colisión, pero conlleva ciertos impedimentos dentro del enfoque del proyecto que impiden su utilización. En primer lugar, los costos de una solución en base a radar son relativamente altos, desde su compra directa con el fabricante hasta la propia instalación. Adicionalmente, utilizan el protocolo de comunicación CAN, que no es compatible nativamente con las alternativas de sistemas embebidos pensados para este proyecto. Por último, no hay disponible un producto a escala de menores características para adquirir y realizar la comprobación de la comunicación y adquisición de datos.

Con respecto al uso de cámaras y su integración en un sistema FCW, se conoce que por lo general tiene como finalidad reforzar la fiabilidad del sistema y generar redundancia, en lo que se conoce como fusión de sensores. Sin embargo, individualmente una única cámara no puede obtener distancias con precisión sin tener un marco de referencia fijo, lo cual no es posible en un entorno dinámico como el automovilístico. La solución a este problema, el uso de cámaras estereoscópicas (para capturar imágenes en tres dimensiones y obtener distancias), requiere de algoritmos con un alto nivel de complejidad y un sistema embebido de mejores características, o fabricado específicamente para el procesamiento de imágenes en tiempo real.

### 2.1.3 Alternativas entornos de simulación

Existen en el mercado varios productos de software para el desarrollo, realización de pruebas, y validación, para tecnologías y algoritmos asociados a sistemas de asistencia avanzada al conductor (ADAS). Las compañías Ansys y VI-Grade corresponden a dos opciones que ofrecen aquello, siendo ambas alternativas viables al proyecto actual.

La opción de Ansys ofrece el control manual de un vehículo virtual en un entorno real, empleando cualquier simulador de conducción a elección. El software contiene una amplia gama ajustes de características para las unidades de sensores que se deseen implementar, así como una librería enriquecida de funciones para el sistema vehicular en cuestión.

**Ilustración 2.1. Ejemplo del uso de un simulador de manejo integrado con el software de Ansys para simulación vehicular AV y ADAS.**

Por otra parte, la opción de VI-Grade aparte de presentar un producto similar al anterior, también ofrece como alternativa el uso de su entorno gráfico de alto realismo VI-WorldSim. Este permite realizar experimentaciones totalmente “fuera de línea”, sin el uso de un simulador real de manejo, incorporando un sistema de tráfico en base a IA, peatones, animales, iluminación, clima, y mucho más.

**Ilustración 2.2. Vistazo de un escenario en el software VI-WorldSim.**

Sin embargo, la razón principal por la cual estas opciones no fueron escogidas es debido a su alto costo de adquisición. Estos entornos de simulación son dirigidos a un público corporativo, siendo promocionados bajo un esquema de entrega de datos de contacto y la solicitud formal de una demostración, para de esta manera concretar con la adquisición del producto.

## 2.2 Diseño de la solución

Básicamente el diseño de la solución consiste en la elección de los dispositivos, módulos y sensores a utilizar, el procedimiento de conexión y comunicación con el sistema embebido para la adquisición de datos, la programación necesaria para la conversión de los datos obtenidos en variables del sistema, y la elaboración del algoritmo de FCW en código Python.

El desarrollo de la solución está conformado por una parte de comprobación de hardware, y otra parte de simulación de software. Lo primero estará dirigido a comprobar que los datos de sensores en tiempo real se obtengan satisfactoriamente, tal que los valores obtenidos sean consistentes y que la variable asociada presente mínima variación con respecto a su valor real. Lo segundo estará dirigido a comprobar la funcionalidad y efectividad del algoritmo de alerta de colisión frontal, mediante simulación de un entorno cinemático en el contexto automovilístico, tal que se pueda visualizar las interacciones de distancia, tiempo y velocidad entre vehículos, con respecto a los instantes de alertas que genera el algoritmo.

### 2.2.1 Conexiones físicas módulos y sensores con Raspberry Pi

Las conexiones de los módulos y sensores se dan a cabo mediante los puertos USB del Raspberry Pi, o mediante las terminales de entrada/salida GPIO.

Cabe recalcar que este procedimiento no es parte de una implementación, ya que a pesar de que se demostrará la correcta comunicación y adquisición de datos en pruebas reales con los equipos y dispositivos, el funcionamiento completo del sistema de prevención de colisión será demostrado en un entorno de simulación que se realizará más adelante.

El módulo GPS empleado para la adquisición de los datos de velocidad del vehículo anfitrión, corresponde al modelo NEO-6M, el cual requiere de una comunicación serial UART con el Raspberry Pi. Ya que solo se requiere la obtención de datos, las únicas terminales que se requieren conectar son las de alimentación (VCC, GND) y de transmisión (TXD). Tanto el módulo GPS como el sensor LIDAR se conectan al RPi mediante USB, pero para este último se emplea un adaptador USB-TTL. Esto es debido a que en los pines GPIO solo está disponible la conexión de un único dispositivo UART, mientras que por USB hay disponibilidad de hasta 4 dispositivos que empleen este tipo de comunicación.

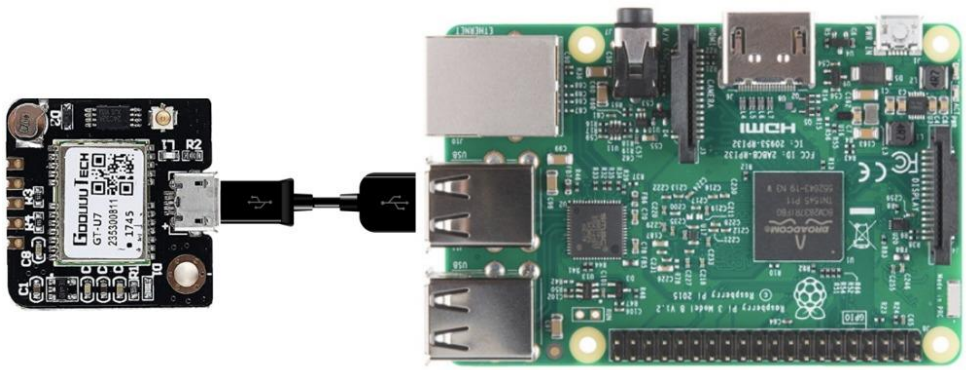


**Ilustración 2.3. Conexión Módulo GPS con Raspberry Pi**

El sensor LIDAR empleado para la adquisición de datos de distancia y velocidad relativa, corresponde al modelo TF-Luna. Este sensor ha sido seleccionado, como un modelo a escala del LIDAR que se usaría para una implementación real. Posee el mismo protocolo de comunicación serial (UART) que su contraparte de alto rango, y características

similares tales como el nivel de comunicación (TTL 3.3V), una tasa de refresco disponible de 100Hz, entre otros.

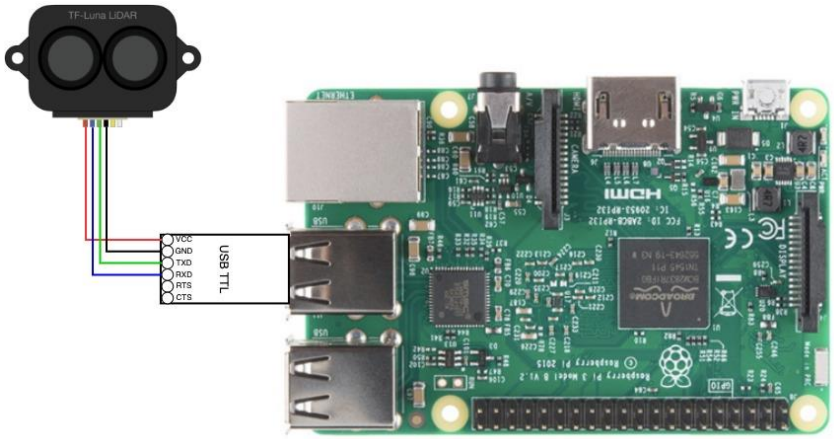



**Ilustración 2.4. Conexión sensor LIDAR con Raspberry Pi**

El display TFT LCD empleado para la salida de alertas corresponde al módulo ST7735S 3.3V. Su pantalla es a todo color, con una resolución de 128x160 pixeles y un tamaño de 1.8 pulgadas. El módulo hace uso de la interfaz SPI para la comunicación serial con el Raspberry Pi. Además de sus 2 terminales de alimentación, requiere de la utilización de 5 pines GPIO para la recepción de imágenes.

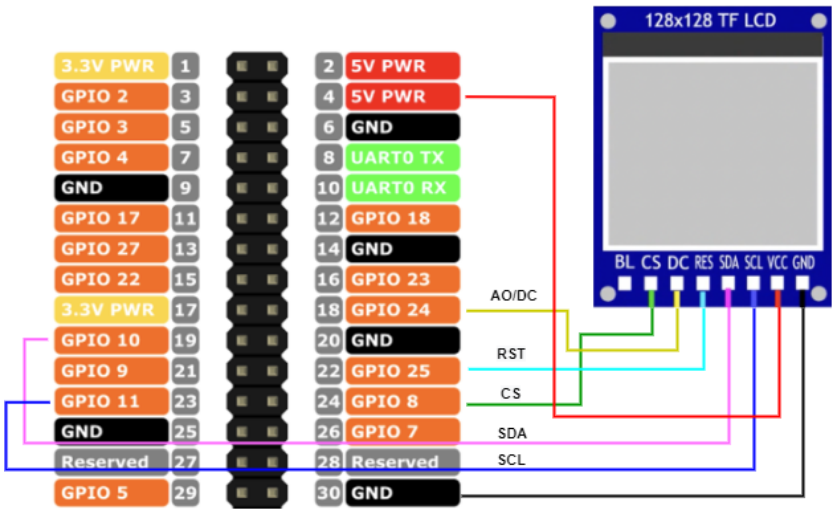



**Ilustración 2.5. Conexión pantalla TFT LCD con terminales GPIO del Raspberry Pi.**

Para la adquisición de la variable de aceleración, se emplea un módulo acelerómetro giroscopio de modelo GY-521, el cual es compatible con alimentación 3.3V y se comunica con el Raspberry Pi mediante el protocolo de comunicación I2C.

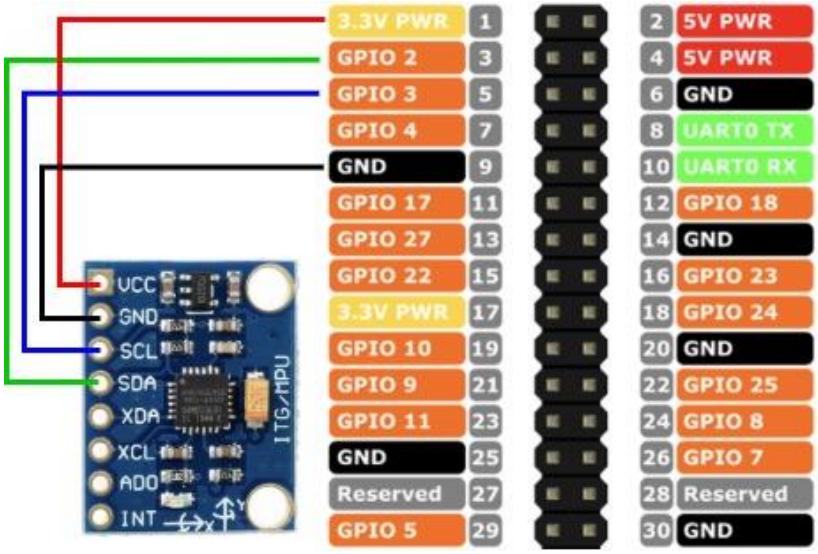



**Ilustración 2.6. Conexión acelerómetro con terminales GPIO del Raspberry Pi.**

Tanto el sensor de luz como el sensor de lluvia funcionan con alimentación y salida lógica de 3.3V, compatible con Raspberry Pi. Cada módulo posee salidas digitales, de las

cuales obtendremos por lógica binaria la detección de cada elemento (luz/agua) correspondiente a su sensor.

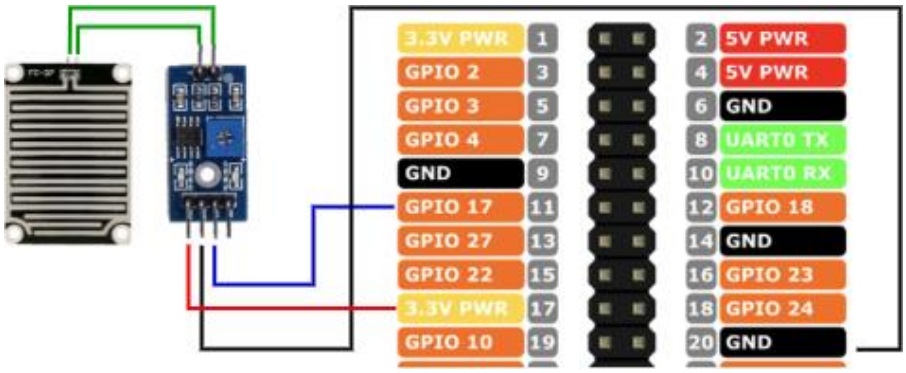



**Ilustración 2.7. Conexión de los sensores ambientales con los terminales GPIO del Raspberry Pi.**

### 2.2.2 Integración con pantalla TFT LCD

Para comprobar el uso de una pantalla TFT LCD que muestre gráficamente el estado de la alerta de colisión al vehículo, se propuso un display de 128x160 RGB que presenta un circuito integrado ST7735. Este modelo de hardware ya presenta librerías dentro del repositorio del sistema operativo Raspberry OS, por lo que se procede a elegir una librería perteneciente a la compañía Adafruit Industries, con propiedad intelectual al autor Tony DiCola.

Su instalación se la realiza directamente desde la ventana de terminal, escribiendo la siguiente línea:

```
sudo pip3 install adafruit-circuitpython-rgb-display
```

Posteriormente se descargan automáticamente todos los paquetes y archivos necesarios para realizar el programa sobre el cual se deben importar todas las imágenes al módulo LCD asociadas a los 4 niveles de alerta del algoritmo del sistema FCW.

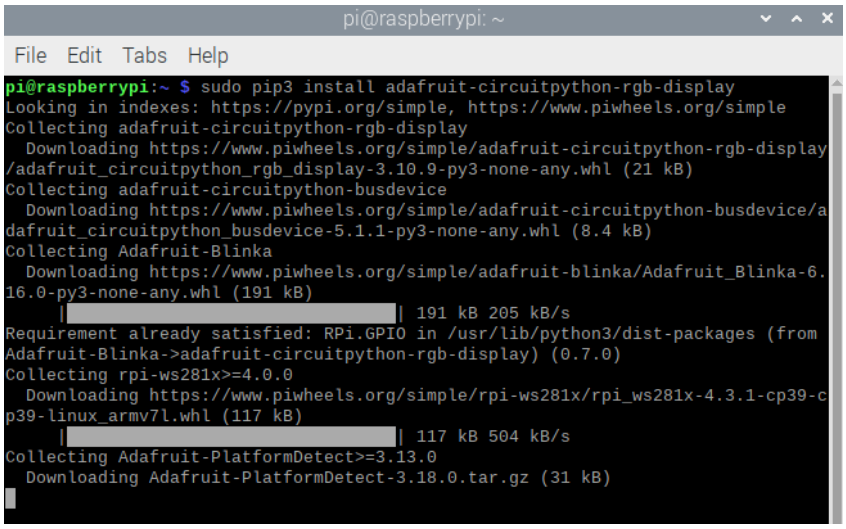



**Ilustración 2.8. Instalación de la librería Adafruit RGB Display mediante**

Una vez instalada y leída la documentación correspondiente, se procede a escribir en un programa Python TFT_LCD.py la configuración inicial. Se realizó la importación de librerías, se especificó el tamaño de los pixeles de la pantalla a usar, su modelo, así como otros parámetros de configuración por defecto que se encuentran detallados en la documentación de la librería en cuestión.

```
from PIL import Image
import ST7735 as TFT
import Adafruit_GPIO as GPIO
import Adafruit_GPIO.SPI as SPI

WIDTH = 128
HEIGHT = 160
SPEED_HZ = 4000000

# Configuración de Raspberry Pi.
DC = 24
RST = 25
SPI_PORT = 0
SPI_DEVICE = 0

# Creación de clase para display TFT LCD.
disp = TFT.ST7735(
    DC,
    width=WIDTH,
    height=HEIGHT,
    rst=RST,
    spi=SPI.SpiDev(
        SPI_PORT,
        SPI_DEVICE,
        max_speed_hz=SPEED_HZ))
```

Posterior a esto, se inicializa el display y se cargan imágenes de formato '.jpg' localizadas en la carpeta Home del Raspberry Pi. Se cambia el tamaño de la imagen y se la rota para que coincida con la pantalla, y por último se envía a dibujar la imagen cargada al hardware del display.

```
disp.begin()
image = Image.open('image.jpg')
image = image.rotate(90).resize((WIDTH, HEIGHT))
disp.display(image)
```

Las imágenes a continuación son de mi autoría y se crearon para los cuatro estados posibles de alerta en el algoritmo de NHTSA.

| **Algoritmo NHTSA – JHU-APL** | |
|---|---|
| **Estado de Alerta** | **Imagen** |
| Sin alerta |  |
| Alerta leve |  |
| Alerta media |  |

| | |
|---|---|
| Alerta máxima |  |
| **GPS no válida** | |
| Sin alerta |  |

### 2.2.3 Creación código Python para algoritmo FCW

Básicamente este proceso requiere la conversión del algoritmo propuesto por NHTSA, a código Python. Se crea un programa FCW.py sobre el cual se programará bajo un bucle 'while' principal que garantice la ejecución continua. Antes del bucle también se importa la librería time y se igualan a cero algunas variables internas. Asimismo se enceran todas las variables cinemáticas, hasta que se obtengan más adelante mediante la simulación a ser realizada.

```
while True:
    # Variables obtenidas de los módulos y sensores
    rango = 0  # Lidar
    vr = 0  # dr/dt
    vh = 0  # GPS
    ah = 0  # Acelerometro o dvh/dt
    ar = 0  # dvr/dt
    GPS = True
    Acelerometro = True
    Lluvia = False
    # Aceleración máxima de frenado
    if Lluvia is False:
        ah_max = -0.6*9.81
    else:
        ah_max = -0.4*9.81
    # ALGORITMO NHTSA - JHU-APL
    if GPS:
        vl = vr+vh
        al = ah + ar
        if -0.1 <= al <= 0.1:
            tls = 0
        else:
            tls = -vl / al

        # Distancia treshold
        d_tresh = 2 + vh*0.1
```

El algoritmo NHTSA será siempre el prioritario, y se ejecutará siempre y cuando la señal del GPS se encuentre activa. Se escriben las ecuaciones asociadas, y se crean condicionales para evitar errores en el programa debido a divisiones para cero. Luego, para obtener la distancia de elusión, se decidió colocar dichas ecuaciones en dos funciones, de acuerdo a las especificaciones del algoritmo. Nótese la variable Lluvia, la cual corresponde a la entrega digital del sensor de dicho parámetro, y modifica el parámetro de aceleración máxima de frenado de 0.6g a 0.4g.

```
        # Distancia miss cuando TLS>TR, es decir, TLS>1.5
        def solve_d_miss_1(am):
            ths = 1.5 - (vh + ah * 1.5)/am
            delta_r1 = (vr*1.5)+0.5*(al-ah)*(1.5**2)
            delta_r2 = (vr+1.5*(al-ah))*(tls-1.5) + 0.5*(al-am)*((tls-
1.5)**2)
            delta_r3 = (vr+1.5*(al-ah)+(al-am)*(tls-1.5))*(ths-
tls)+0.5*(-am)*((ths-tls)**2)
            d_miss_1 = rango + delta_r1 + delta_r2 + delta_r3
            return d_miss_1

        # Distancia miss cuando TLS<=1.5
        def solve_d_miss_2(am):
            tm = 1.5 + (vr+1.5*(al-ah))/(am-al)
            delta_r1 = (vr*1.5)+0.5*(al-ah)*(1.5**2)
            delta_r2 = (vr+1.5*(al-ah))*(tm-1.5) + 0.5*(al-am)*((tm-
1.5)**2)
            d_miss_2 = rango + delta_r1 + delta_r2
            return d_miss_2
```

Estas funciones, tal y como está expresado en el algoritmo original, se encuentra dividido dependiendo del resultado de la variable $T_{LS}$ (Tiempo de detención del vehículo delantero). De acuerdo a esto se genera una condicional adicional en las que se llama a la función, de acuerdo a los tres niveles de alerta esperados.

```
        if tls >= 1.5:
            d_miss_high = solve_d_miss_1(ah_max)
            d_miss_med = solve_d_miss_1(ah_max*0.75)
            d_miss_low = solve_d_miss_1(ah_max*0.5)
        else:
            d_miss_high = solve_d_miss_2(ah_max)
            d_miss_med = solve_d_miss_2(ah_max*0.75)
            d_miss_low = solve_d_miss_2(ah_max*0.5)
```

Por último, se realiza la comparación de las distancias de umbral y de elusión, pero añadiendo condicionales y tomando control sobre el tiempo. Esto se lo realiza debido a que deben pasar al menos un periodos de tiempo en que se cumpla la alerta interna de frenado, equivalente a 250 ms, para que se pueda generar una alerta real externa. De la misma manera, esta alerta está pensada para mantenerse activa por un tiempo mínimo de 1 segundo, y desactivarse automáticamente cuando el sistema no detecte peligro de colisión en la comparación de distancias.

A continuación se muestra el código para la activación de la alerta máxima, pero tomando en cuenta que para las dos alertas restantes se sigue exactamente el mismo proceso.

```
        # Activacion Alerta Máxima
        if d_tresh > d_miss_high:
            time_high = time.time() + 0.2
            if time.time() > time_high:
                alert_high = 1
        else:
            time_high_2 = time.time()+1
            if time.time() > time_high_2:
                alert_high = 0
```

Cabe recalcar que, al asociar este código directamente con los programas a realizarse a continuación para la adquisición de datos de sensores, representa lo que el minicomputador Raspberry Pi ejecutará en cada encendido para completar la parte del software del sistema de prevención de colisión. Sin embargo, este será posteriormente modificado, en pequeña proporción, para ajustarse a trabajar con los datos de las variables cinemáticas proporcionadas por la herramienta de simulación que ha sido desarrollada y se explicará más adelante.

## 2.3 Desarrollo de la solución

Esta sección se basa principalmente en escribir en Python para el Raspberry Pi, métodos de adquisición de datos y conversión a parámetros cinemáticos para cada uno de los sensores del sistema. Igualmente trata de la programación del software asociado a el sistema de prevención de colisión, así como también de la generación de diversos escenarios de simulación que permitirán poner a prueba los algoritmos empleados y el rendimiento de los datos recopilados.

### 2.3.1 Integración de sensores con Raspberry Pi

Para la adquisición de los datos de sensores a utilizar, primeramente, se debió de configurar el Raspberry Pi, comenzando por la instalación de su sistema operativo, y culminando con la activación de los protocolos para la comunicación de acuerdo a los sensores a ser usados. Para todos los procesos a continuación, se realizarán en un nuevo archivo Python en blanco.

### 2.3.2 Adquisición de datos - Acelerómetro

Para el acelerómetro, basta solamente con importar la librería del integrado MPU6050 usado en el módulo para adquirir los datos transmitidos por la comunicación I2C del RPi. Sin embargo, el único paso externo que se requiere es la búsqueda del valor hexadecimal asociado a la dirección del módulo, el cual se consigue mediante comandos de detección ingresados en el terminal. La dirección obtenida, que en mi caso correspondió a 0x68, se entrega a la librería importada como entrada, y los datos de aceleración de salidas se obtienen llamando a una función de esta misma.

```
from mpu6050 import mpu6050
mpu = mpu6050(0x68)
while True:
    accel_data = mpu.get_accel_data()
    ax = float(accel_data['x'])
    ay = float(accel_data['y'])
    az = float(accel_data['z'])
    # Valores obtenidos experimentalmente
    if 9.32<=az<=10.3:
        acc = True
```

Cabe recalcar que estos datos de aceleración son usados en el algoritmo siempre y cuando el sistema físico esté orientado en la posición predeterminada descrita en la

descripción del módulo en el anterior capítulo, y que el eje z se encuentre entre un rango estrecho de valores asociados solamente al efecto de la gravedad.

### 2.3.3 Adquisición de datos - Sensor Lidar

Para el sensor Lidar, se realizó el mismo procedimiento de búsqueda de dispositivos en el que se obtuvo la dirección “/dev/ttyUSB0” para la lectura de datos. En este caso se lee la sentencia con una velocidad de baudios de 115200, que es el valor por defecto para el sensor mencionado por el fabricante.

```
import serial,time
import numpy as np

ser = serial.Serial("/dev/ttyUSB0", baudrate=115200, timeout=0)
```

Mediante el uso del manual de usuario proporcionado por el fabricante, se escribe el código de lectura de datos de acuerdo al formato predeterminado de salida serial. Este se filtra mediante una condicional que determina si los números alfanuméricos asociados a los bytes 0 y 1 corresponden al valor 0x56. Entonces, se realiza la conversión necesaria de los bytes 2 y 3 de dicha sentencia para obtener los datos de la distancia en metros.

```
def read_lidar():
    while True:
        counter = ser.in_waiting
        if counter > 8:
            bytes_serial = ser.read(9)
            ser.reset_input_buffer()
            if bytes_serial[0] == 0x59 and bytes_serial[1] == 0x59:
                distance = bytes_serial[2] + bytes_serial[3]*256
                return distance/100.0
```

Sobre el dato de distancia obtenido, el sistema requiere también las variables de velocidad y aceleración relativa. Ya que el sensor real presenta una variación entre los datos medidos, que gráficamente es observado como ruido en la señal, se eligió una recolección de datos equivalente a un 10% de frecuencia de su variable contigua. Esto quiere decir que la distancia se actualiza acorde a 100Hz, la velocidad relativa acorde a 10Hz, y la aceleración relativa acorde a 1Hz. La efectividad de este método se comprobará más adelante.

Para realizar este procedimiento, se crean listas antes del bucle principal, para guardar momentáneamente los valores de cada variable en el periodo de tiempo correspondiente. Se obtiene la distancia llamando a la función read_lidar() creada anteriormente, lo que indirectamente también genera un tiempo de espera de 0.01 segundos en cada iteración. Esto es debido a que, al momento de llamar a dicha función, dentro de esta también hay un bucle, que termina solamente cuando logra obtener la condición de bytes asociada al valor medido de distancia. De esta manera, un valor será agregado a una lista acorde a dicho tiempo, lo que permite la diferencia de tiempo necesaria en el cálculo de velocidad y aceleración. Para obtener la velocidad relativa, se ha escrito el siguiente código:

```
d_lista = []
vr_lista = []
while True:
    # Para obtener distancia
    distancia = read_lidar()
    # Para obtener velocidad relativa
    d_lista.append(distancia)
    d_len = len(d_lista)
    if d_len == 11:
        d1 = d_lista[0] # Valor en la primera posición
        d2 = d_lista[10] # Valor en la última posición
        vr = (d2-d1)/0.1 # 100 Hz para 10+1 valores es 0.1s
        vr_lista.append(vr) # Lista vr, para luego calcular ar
        d_lista.clear() # Borrar los contenidos de la lista de
distancia
        d_lista.append(d2) # El ultimo valor de la lista i-1 pasa a
ser el primero en la lista i
```

Naturalmente se realiza un procedimiento idéntico para calcular la aceleración relativa, y tomando en cuenta que la lista de velocidad relativa aumenta su tamaño cada 0.1 segundos, al tener un tamaño de 11 la diferencia de tiempo será de 1 segundo, haciendo que la aceleración relativa tenga una tasa de refresco de 1Hz.

```
    # Para obtener aceleracion relativa
    vr_len = len(vr_lista)
    if vr_len == 11:
        vr1 = vr_lista[0]
        vr2 = vr_lista[10]
        ar = (vr2-vr1)/1 # 10 Hz para 10+1 valores es 1s
        v_lista.clear()
        v_lista.append(vr2)
```

### 2.3.4 Adquisición de datos - Módulo GPS

Para el módulo GPS se solicitó al terminal del RPi, por medio de comandos, el listado de los dispositivos USB conectados a este, de modo que se obtuvo una dirección interna "/dev/ttyACM0" con la cual se leen los datos mediante el protocolo serial correspondiente desde el programa en Python.

Los datos se obtuvieron llamando a la componente serial con una velocidad en baudios de 9600 y un valor timeout de 0.5, de acuerdo con las especificaciones del fabricante para el módulo empleado. Se importan las librarías requeridas, se crea una variable booleana para representar el estado del GPS, y una lista que servirá para obtener un valor suplente para la aceleración lineal del vehículo.

```
import serial
import time

port="/dev/ttyACM0"
ser = serial.Serial(port,baudrate=9600,timeout=0.5)
gps = False
v_lista = []
```

Luego de esto, dentro del bucle principal, se filtra solo la línea de datos recibida con la especificación $GPRMC acorde a las sentencias NMEA. Esta línea, al separarla por comas, permite obtener de la octava posición de la lista creada el valor de velocidad horizontal (en nudos), calculado por el propio módulo a partir de la tasa de cambio de las coordenadas obtenidas, es decir, de la posición del vehículo a través del tiempo. Si el GPS no recibe señal, no habrá ningún número o carácter en el espacio de la línea de datos donde estuviera el valor de velocidad.

```
while True:
    data = ser.readline()
    newdata = str(data)
    if newdata[2:8] == "$GPRMC":
        gps = True
        # Para obtener velocidad del vehiculo
        string = newdata.split(",") # Separamos los bytes de datos
        v = string[7] # De acuerdo a la normativa el 7mo byte de
$GPRMC corresponde a velocidad
        if v == ""
            gps = False # El GPS no recibe señal
        else:
            gps = True
            v = float(v)*0.5144 # Convertir velocidad de nudos a m/s
            v_lista.append(v)
```

Luego, se obtiene el valor de aceleración cada segundo con el mismo proceso de recopilación de datos de velocidad durante un periodo de tiempo conocido, tal como se lo realizó para el sensor Lidar.

```
        # Para obtener un valor suplente para la aceleracion
        if v_lista == 6:
            v1 = v_lista[0]
            v2 = v_lista[5]
            a_gps = (v2-v1)/1 # velocidad se actualiza a 5Hz, 6 datos
de muestra para -> 1Hz
            v_lista.clear()
            v_lista.append(v2)
```

### 2.3.5 Adquisición de datos – Sensor de lluvia

Para el sensor ambiental de lluvia, simplemente se requirió conocer el estado lógico de los pines físicos sobre el cual está físicamente conectado. Para este procedimiento se importa la librería de RPi.GPIO al programa, de forma que se configura el pin GPIO17 como entrada y se pregunta si posee el estado lógico HIGH o LOW. Esto se lo realiza con el siguiente código:

```
import RPi.GPIO as GPIO

# Definir la convención de numeración de pines
GPIO.setmode(GPIO.board)
# Configurar el pin GPIO 17 como entrada
GPIO.setup(17, GPIO.IN)
# Valor de la entrada
lluvia = GPIO.input(17)
```

### 2.3.6 Creación de escenarios de conducción en Matlab

En esta sección se hizo uso del programa Matlab R2021a para crear diversos escenarios de conducción, los cuales han sido adaptados para abarcar situaciones posibles en el contexto previo a una posible colisión automovilística. De estos escenarios se genera un sistema de lazo abierto sobre el cual se corre una simulación para obtener arreglos de datos en función de tiempo, tal que generen la información tanto de los sensores empleados como de los parámetros cinemáticos de los vehículos.

Una vez ingresado a la ventana principal de Matlab, se procedió a hacer uso de la herramienta Driving Scenario Designer (DSD), la cual es accesible al escribir la siguiente línea en la ventana de comandos.

```
drivingScenarioDesigner()
```

En la ventana emergente, se puede apreciar una barra de herramientas sobre la cual se eligen opciones para comenzar a generar el escenario.

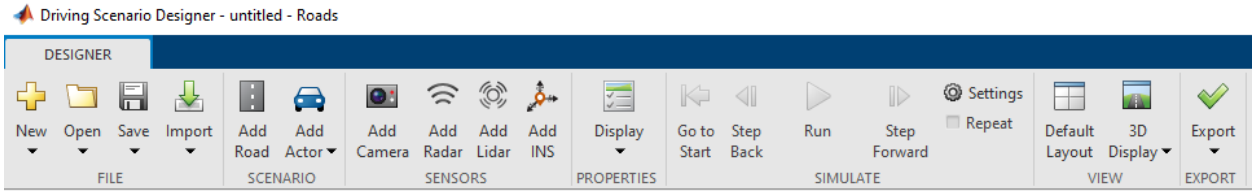



**Ilustración 2.9. Barra de herramientas del programa Matlab, Driving Scenario Designer.**

Para comenzar a crear el escenario se escogen las opciones dentro de “SCENARIO” y “SENSORS”. La primera consiste en la colocación de calles y actores (vehículos), mientras que la segunda consiste en la colocación y modificación de características técnicas de sensores de visión, radar o Lidar.

La calle no tiene ningún impacto en la simulación, ya que sólo sirve de referencia para colocar los vehículos y crear sus trayectorias. De todos modos, se crea una calle estándar de 4 carriles, con un ancho de 3.6m cada uno. También se crea el primer actor, el cual siempre es el vehículo ego, y se lo coloca en la posición inicial.

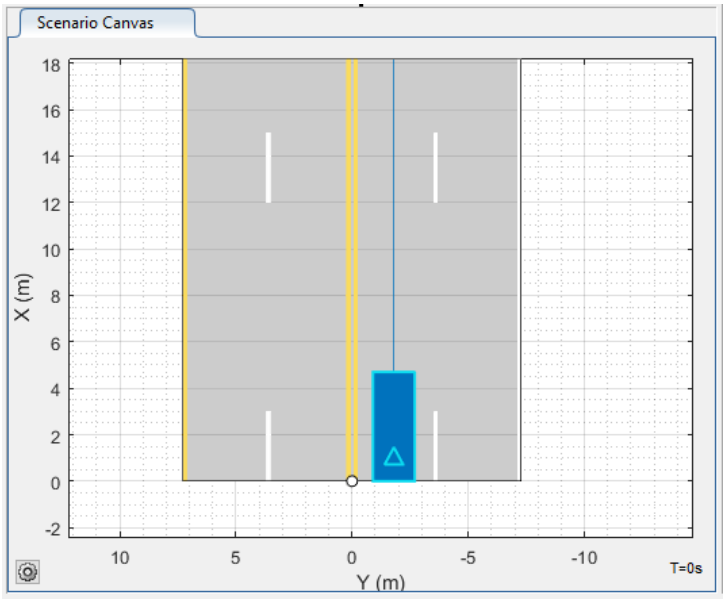



**Ilustración 2.10. Vista del comienzo de la creación de un escenario; vehículo ego es colocado en la posición inicial de la calle.**

Con respecto al sensor escogido, este es del tipo radar, pero con sus parámetros modificados de tal manera que se asemeje lo mayor posible a los dos modelos de Lidar con los cuales se ha diseñado el sistema. Estos parámetros se muestran en la tabla a continuación.

**Tabla 2.1. Modificación de parámetros del sensor radar en Matlab DSD.**

| Parámetro | Valor | Observación |
|---|---|---|
| Posición (x,y,z) | (1.9,0,1) | Con respecto a las coordenadas del vehículo. |
| Probabilidad de Detección | 1 | El sensor nunca deja de detectar al objecto en frente. |
| False Alarm rate | 1e-07 | Probabilidad que retorne un valor aleatorio entre 0 y la distancia real. |
| Field of View (FOV) | 3° / 0.5° | Lidar TF02-Pro / TF03 |
| Max Range | 40m / 100m | Lidar TF02-Pro / TF03 |
| Limit # of detections | 1 | Entrega un único valor de distancia del objeto más cercano dentro del área de detección. |
| Has Noise | No | Se le añade manualmente al recopilar los datos. |
| Has False Alarms | Si | Usado para comprobar que un dato falso en un solo instante de tiempo no activa ninguna alarma. |

Este sensor se coloca en una de las zonas predeterminadas del vehículo llamada “Front Window”, que es la que tiene mayor relación con el lugar real sobre el cual el sistema está diseñado en ser colocado.

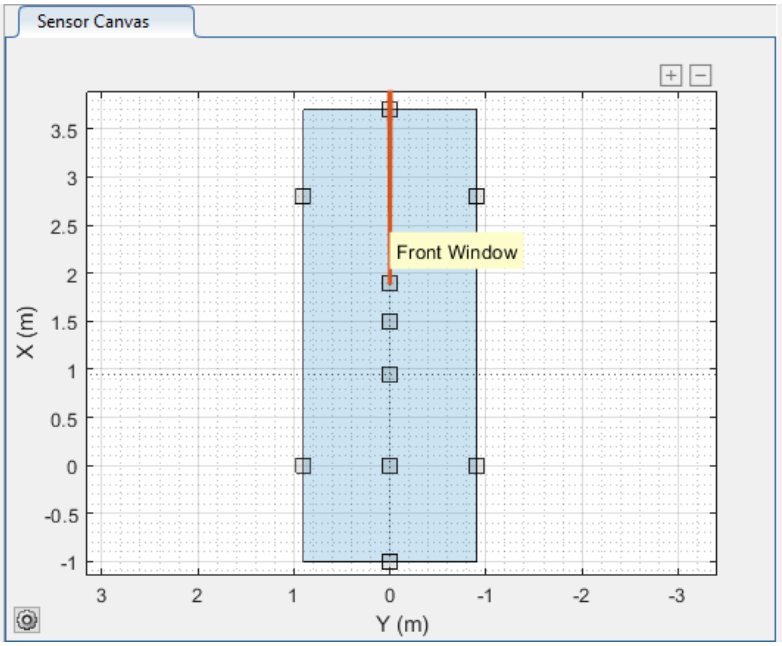



**Ilustración 2.11. Vista de la colocación del sensor sobre las coordenadas del vehículo.**

Según las dimensiones del vehículo de simulación, el sensor se encuentra a una distancia de 1.8 metros de la parte frontal. Esta compensación de distancia ya ha sido añadida anteriormente al algoritmo FCW.

Cabe recalcar que todos los actores a incluirse son vehículos de tamaño estándar (proporcionado por el DSD), de dimensiones 4.7 x 1.8 x 1.4 m (largo x ancho x altura). La trayectoria del vehículo ego siempre se dará a cabo en línea recta sobre su propio carril, mientras que las velocidades usadas están dentro de un rango real en términos de ciudad y carreteas, pero dependen de cada escenario y tipo de sensor a usar. En cambio, para los vehículos delanteros, tanto la trayectoria como la velocidad del vehículo dependerán del escenario.

Se elaboraron 4 escenarios, los cuales convergen en varios casos sobre los cuales pueden cambiar tres aspectos. El modelo de Lidar usado, el rango inicial entre ambos vehículos y la velocidad a la que viajan. Estos escenarios se detallan a continuación:

- **Escenario 1: Acercamiento a vehículo detenido.** Se coloca un vehículo ego en la posición inicial, y un vehículo delantero a una distancia máxima de 100m. El vehículo ego se dirige en curso de colisión a velocidad constante. El vehículo delantero no tiene trayectoria ni movimiento alguno.

**Tabla 2.2. Parámetros cinemáticos para escenario 1**

| Caso | Tipo de Lidar | Rango inicial (m) | Velocidad Vehículo ego (km/h) |
|---|---|---|---|
| 1 | TF03 | 100 | 60 |
| 2 | | | 110 |

- **Escenario 2: Acercamiento paulatino.** Se coloca un vehículo ego en la posición inicial, así como un vehículo delantero a 30m de distancia, Ambos van a velocidad constante pero el delantero se encuentra 30 km/h por debajo del ego.

**Tabla 2.3. Parámetros cinemáticos para escenario 2**

| Caso | Tipo de Lidar | Rango inicial (m) | Velocidad vehículo ego (km/h) | Velocidad vehículo delantero (km/h) |
|---|---|---|---|---|
| 1 | TF02-Pro | 30 | 90 | 60 |

- **Escenario 3: Frenado repentino.** Se coloca un vehículo ego en la posición inicial, así como un vehículo delantero a diferentes rangos de distancia. Cada vehículo va inicialmente a la misma velocidad constante, y luego de un tiempo determinado, el vehículo delantero frena inesperadamente con una aceleración de 0.6g.

**Tabla 2.4. Parámetros cinemáticos para escenario 3**

| Caso | Tipo de Lidar | Rango inicial (m) | Velocidad vehículos (km/h) | Desaceleración vehículo delantero (m/s^2) |
|---|---|---|---|---|
| 1 | TF02-Pro | 40 | 70 | 0.6 |
| 3 | | 30 | | 0.6 |

- **Escenario 4: Cruce espontáneo.** Se coloca un vehículo ego en la posición inicial, un segundo vehículo a un rango de 50m en el mismo carril, y un tercer vehículo en el carril adyacente. Después de un tiempo determinado, este tercer vehículo se cambia al carril del vehículo ego, y sigue en una trayectoria recta por el mismo carril. No existe ningún riesgo de colisión en este escenario. Las velocidades son de 50 km/h para el vehículo ego, y de 70 km/h para el resto.

**Tabla 2.5. Parámetros cinemáticos para escenario 4**

| Caso | Tipo de Lidar | Rango inicial (m) | Velocidad Vehículo ego (km/h) | Velocidad vehículo delantero (km/s) |
|---|---|---|---|---|
| 1 | TF03/02-Pro | No aplica | 50 | 70 |

- **Escenario 5: Conducción cercana.** Ambos vehículos viajan a la misma velocidad constante de 70 km/h y a una distancia inicial de 20 metros.

| Caso | Tipo de Lidar | Rango inicial (m) | Velocidad Vehículos (km/h) |
|---|---|---|---|
| 1 | TF03/02-Pro | 20 | 70 |

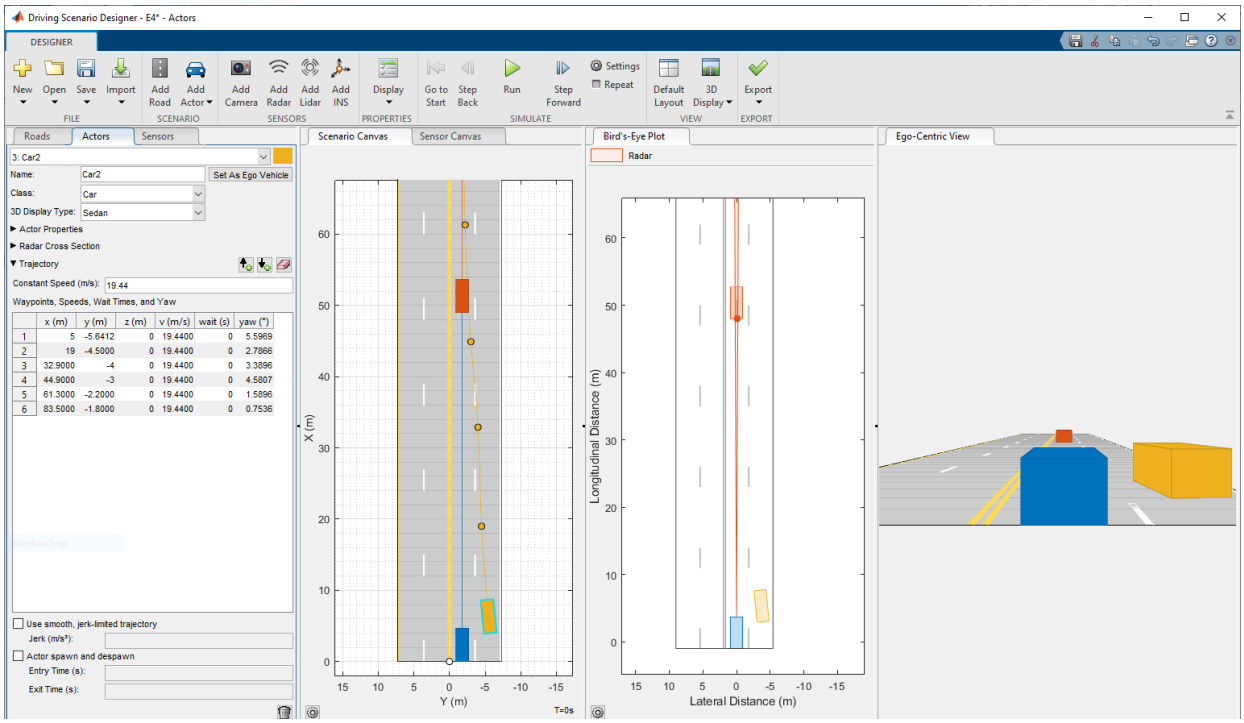
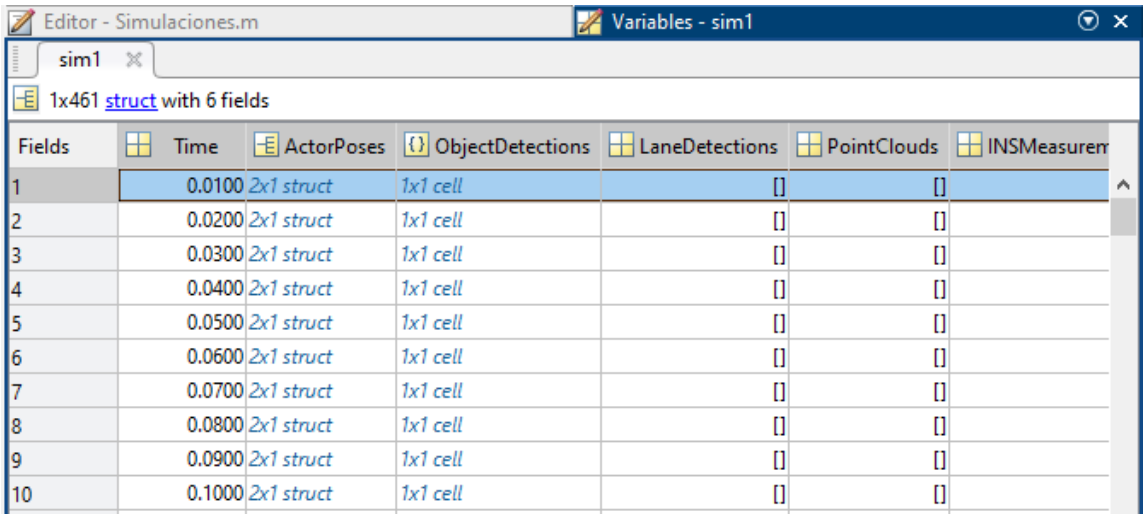



**Ilustración 2.12. Captura de ventana de Matlab DSD, Escenario 4.**

Cabe recalcar que, apegándose al enfoque de este proyecto, no se requiere realizar un completo análisis de los algoritmos empleados. Tomando en cuenta que cada escenario puede realizarse a diversos rangos, y cada rango a su vez a diferentes velocidades, comprobar todas las posibilidades de rangos y velocidades aumentaría el número de simulaciones de forma exponencial. Lo que se desea es comprobar el rendimiento de la adquisición de datos, el tiempo de respuesta del sistema, el efecto del error debido al ruido en los valores de distancia, la comparación de ambos algoritmos de alerta, y por último, comprender el uso de un tipo de sensor sobre el otro en los diferentes escenarios.

### 2.3.7 Obtención de datos – Simulación Matlab

Los datos de cada simulación se exportan al Workspace en una estructura 1 x n de 6 campos, siendo n proporcional al tiempo total de acuerdo a $n = 0.01 * t$.




| Fields | Time | ActorPoses | ObjectDetections | LaneDetections | PointClouds | INSMeasurem |
|---|---|---|---|---|---|---|
| 1 | 0.0100 | 2x1 struct | 1x1 cell | [] | [] | |
| 2 | 0.0200 | 2x1 struct | 1x1 cell | [] | [] | |
| 3 | 0.0300 | 2x1 struct | 1x1 cell | [] | [] | |
| 4 | 0.0400 | 2x1 struct | 1x1 cell | [] | [] | |
| 5 | 0.0500 | 2x1 struct | 1x1 cell | [] | [] | |
| 6 | 0.0600 | 2x1 struct | 1x1 cell | [] | [] | |
| 7 | 0.0700 | 2x1 struct | 1x1 cell | [] | [] | |
| 8 | 0.0800 | 2x1 struct | 1x1 cell | [] | [] | |
| 9 | 0.0900 | 2x1 struct | 1x1 cell | [] | [] | |
| 10 | 0.1000 | 2x1 struct | 1x1 cell | [] | [] | |

**Ilustración 2.13. Estructura de datos provenientes de la simulación de un escenario.**

Los datos provenientes de “ActorPoses” incluyen la posición y velocidad de cada vehículo en el escenario. En cambio, los datos de “ObjectDetections” incluye el valor de distancia medido por el sensor, y otros parámetros asociados. Ingresando manualmente a cada uno de estos permite obtener la dirección en relación a la estructura principal sobre la cual es posible realizar la extracción de dichos datos.

Se requiere extraer las variables de posición, velocidad, velocidad relativa y aceleración de cada vehículo, a través de los datos de actores. Al mismo tiempo, se requiere extraer la variable de distancia a través de los datos de sensor. Se debe recordar que en el proceso de adquisición de datos la variable de velocidad relativa no la entrega el sensor, si no que se la calcula a partir del cambio de distancia a través del tiempo. En este caso se extrae la velocidad relativa real para compararla posteriormente con los valores del cálculo en base a $\Delta x/\Delta t$.

Se crea un nuevo script de Matlab para realizar la extracción de los datos de simulación.

Todas las variables se recopilan en forma de arreglos, extrayendo los datos a partir de su dirección correspondiente en la estructura de simulación llamada “sim1”.

En primer lugar, creamos un arreglo de ceros para el tiempo de simulación, dándole el mismo tamaño que la estructura y reemplazando dichos ceros por el valor correspondiente mediante el uso de un lazo for.

```
%Para obtener el tamaño de líneas de datos
n1 = size(struct2table(sim1),1);
%Para obtener un array de los datos de tiempo
tiempo = zeros(n1,1);
for i=1:n1
    t = sim1(i).ObjectDetections{1, 1}.Time;
    tiempo(i,1)=t;
end
```

Luego obtenemos el valor de distancia de dos maneras, una “teórica” y otra “práctica”. La manera teórica corresponde al valor real de distancia sin ruido ni error, mientras que la manera práctica corresponde al valor de distancia tomando en consideración el error de exactitud de ±5cm (0.1 – 5m) y de ±1% (5-40m). Se le añade dicho valor haciendo uso de valores aleatorios y probabilidad.

```
%Array de los datos de distancia
distancia = zeros(n1,1);
for i=1:n1
    x = sim1(i).ObjectDetections{1,1}.Measurement(1);
    distancia(i,1)=x;
end
```

```
%Array de los datos de distancia con ruido por error de exactitud
for i=1:n1
    x = sim1(i).ObjectDetections{1,1}.Measurement(1);
    rn = rand(1);
    %Condición 1: distancia>5m
    %Condicion 2: probabilidad de 50% para aumentar +1% de x
    if x>5 && rn>0.5
        xr = x + 0.01*x;
    %Condición 1: distancia<=5m
    %Condicion 2: probabilidad de 50% para aumentar +0.05 de x
    elseif x<=5 && rn>0.5
        xr = x + 0.05;
    else
        xr = x;
    end
    distancia(i,1)=xr;
end
```

A continuación se obtienen las velocidades reales de cada vehículo, para los escenarios donde existan solamente 2 de estos. Se realiza un procedimiento similar a los anteriores, ya que se crean arreglos de ceros y luego mediante un lazo se ingresan uno a uno los datos al arreglo.

```
%% Para obtener velocidades reales (v, v_rel)
velocidad = zeros(n1,1);
vel_relativa = zeros(n1,1);
for i=1:n1
    %Velocidad vehiculo ego
    v1_struct = sim1(i).ActorPoses;
    v1_cell = struct2cell(v1_struct);
    v1_cell2 = v1_cell(3,1);
    v1_array = v1_cell2{1,1};
    v1 = v1_array(1,1);
    velocidad(i,1)= v1;
    %Velocidad relativa (V2-Vego)
    v2_struct = sim1(i).ActorPoses;
    v2_cell = struct2cell(v2_struct);
    v2_cell2 = v2_cell(3,2);
    v2_array = v2_cell2{1,1};
    v2 = v2_array(1,1);
    v_rel = v2 - v1;
    vel_relativa(i,1)=v_rel;
end
```

Para el escenario 4 donde se simulan 3 vehículos, más adelante se crea otro script ligeramente modificado para extraer correctamente los datos de dicho tercer actor, ya que tiene una dirección diferente en la estructura.

Para obtener la aceleración lineal teórica realizamos directamente una diferenciación al arreglo de datos de velocidad. Esto es exclusivamente para comparación de valores, ya que la aceleración lineal práctica se obtiene a partir del cambio de velocidad a través del tiempo ($\Delta v/\Delta t$).

```
acc = diff(velocidad)/0.01;
```

Por último, se extraen las posiciones de ambos vehículos para así posteriormente generar una gráfica a través del tiempo y observar el tiempo en que colisionan. Cabe recalcar que estas posiciones requieren una compensación de distancia de acuerdo a que se desea saber la posición de la parte frontal del vehículo ego con respecto a la parte trasera del vehículo delantero. Siguiendo las medidas de vehículo acorde a la ilustración 2.10, se obtiene que de un valor de posición dado, la parte frontal se encuentra a +2.7m, mientras que la parte trasera se encuentra a -2m.

```
%% Posición de vehiculo 1,2 vs tiempo
pos1 = zeros(n1,1);
pos2 = zeros(n1,1);
for i=1:n1
    %Posicion vehiculo anfitrion
    pos1_struct = sim1(i).ActorPoses;
    pos1_cell = struct2cell(pos1_struct);
    pos1_cell2 = pos1_cell(2,1);
    pos1_array = pos1_cell2{1,1};
    p1 = pos1_array(1,1);
    %Compensada la diferencia de coordenadas terreno vs vehiculo
    pos1(i,1)= p1+2.7;

    %Posicion vehiculo delantero
    pos2_struct = sim1(i).ActorPoses;
    pos2_cell = struct2cell(pos2_struct);
    pos2_cell2 = pos2_cell(2,2);
    pos2_array = pos2_cell2{1,1};
    p2 = pos2_array(1,1);
    %Compensada la diferencia de coordenadas terreno vs vehiculo
    pos2(i,1)= p2-2;
end
```

Por último, ya que se requieren usar todos los datos de los arreglos en un programa escrito en Python, se exportan desde la ventana de comandos de Matlab como archivos de texto con formato de separación por nueva línea. Se usan las siguientes líneas de código:

```
new_txt=sprintf('%.15f\n',array)
fid=fopen('file1.txt','w');
new_txt=fprintf(fid,'%.15f\n',array)
fclose(fid)
```

Se cambia la variable “array” por el arreglo que se desee exportar, y ‘file1.txt’ por el nombre de archivo de texto que se desee. Por cuestiones de organización los archivos de texto llevan el siguiente formato: “Escenario” + “Caso” + “Variable”, ej. “E1C1-R” para el rango del caso 1 del escenario 1.

El código completo para la obtención de datos de la simulación, incluido la generación de gráficas, se puede observar en el Apéndice B del presente documento.

### 2.3.8 Ejecución de algoritmo FCW con datos simulados

Para esta sección, el algoritmo FCW previamente escrito debió de ser modificado en el sentido que todas las variables deben de extraerse de los arreglos de datos a ser importados desde los archivos de texto. Se crearon tres archivos:

- **FCW_Teorico.py**: Se simulará el sistema con todas las variables a una misma tasa de refresco simulada de 100 datos por segundo virtual. Esto se realiza para observar la respuesta del sistema, si no existieran retrasos por la adquisición de datos y cálculo de variables cinemáticas.
- **FCW_Practico.py**: Se simulará el sistema empleando los mismos métodos de obtención de variables usados en la adquisición de datos de sensores, e igualando cualquier variable a su tasa de refresco real. Se añade ruido artificial sobre la señal de distancia de acuerdo a condiciones normales del sensor Lidar.
- **FCW_Ruido.py**: Se añade ruido artificial sobre la señal de distancia de acuerdo a condiciones desfavorables del sensor Lidar.

Cabe recalcar que 1 segundo virtual en la simulación equivale a 100 datos en los arreglos de valores de la simulación de Matlab. Es decir, cada iteración del bucle principal representa 10ms, que se tendrá para el control del tiempo a través de los cálculos realizados y alertas generadas en el algoritmo.

Para los datos teóricos, la primera adición con respecto al código original FCW.py es la importación de arreglos de un archivo de texto externo, pero ubicados en la misma carpeta del sistema operativo en la que se encuentran los archivos Python.

```
# IMPORTACION ARRAYS DE MATLAB -> TXT -> PYTHON
import numpy as np
import time

t1 = time.time()

rango_array = np.genfromtxt("E1C1-R.txt", delimiter='\n')
vr_array = np.genfromtxt("E1C1-VR.txt", delimiter='\n')
ar_array = np.genfromtxt("E1C1-AR.txt", delimiter='\n')
vh_array = np.genfromtxt("E1C1-V.txt", delimiter='\n')
ah_array = np.genfromtxt("E1C1-A.txt", delimiter='\n')
i1 = len(rango_array)
```

Adicionalmente, el bucle principal termina cuando se llega al final del arreglo de datos, y en cada iteración de este se extraen cada una de las variables cinemáticas.

```
i = 0
# BUCLE PRINCIPAL
while i <= i1:
    tiempo = i * 0.01  # Se conoce que el tiempo es de 10 ms entre
cada dato de los arreglos
    # Obtencion de datos de variables en cada arreglo
    rango = rango_array[i]
    vr = vr_array[i]
    vh = vh_array[i]
    # Ya que al diferenciar en Matlab se perdieron 2 unidades de
tiempo
    if i < i1-2:
        ah = ah_array[i]
        ar = ar_array[i]
```

En cuanto a las ecuaciones que calculan el riesgo de colisión, no hay cambios mayores con excepción a la generación de la alerta externa, ya que ahora no se basa en el tiempo que transcurra, si no por las mismas iteraciones del bucle.

### 2.3.9 Comprobación de no colisión posterior a la alerta máxima

Una vez obtenidos para ambos vehículos los datos instantáneos de distancia, velocidad y aceleración en cada una de las alertas, se procede a crear las funciones cinemáticas que demuestren teóricamente si luego de la alerta, se evita la colisión. Esto tomando en cuenta los dos periodos posteriores a una alerta:

- Periodo de reacción: 1.5 [s]
- Periodo de frenado: -0.6g [m/s^2]

```
%% TRAYECTORIA VEHICULO EGO (Host Vehicle)

% Intervalo de tiempo sin reacción del conductor
x1a = (v1*t)+0.5*a1*(t.^2);

% Intervalo de tiempo con frenado máximo
d1 = v1*(1.5)+0.5*a1*((1.5).^2); %distancia recorrida en 1.5s
v1a = v1 + a1 * 1.5; %velocidad al culminar los 1.5 seg
x1b = d1 + v1a*(t-1.5) + 0.5*(am)*((t-1.5).^2); % Función desplazada 1.5s
% Se crea una función por partes
x1 = piecewise(t<=1.5, x1a, t>1.5, x1b);
```

Se emplean las mismas ecuaciones cinemáticas asociadas al tiempo de detención, pero realizando ciertas modificaciones para garantizar la continuidad en la función por partes. Primero se escribe la función para el periodo de reacción, sobre la cual luego se calcula la distancia que se recorre y la velocidad al terminar el periodo. Dicha distancia se suma a la función correspondiente al periodo de frenado, y la nueva velocidad ahora se incluye. También se desplaza en el tiempo a la variable t para compensar los 1.5 segundos que ya sucedieron acorde a los cálculos del periodo anterior.

Ya que se asume que luego de la alerta el vehículo delantero mantendrá sus parámetros cinemáticos, esta conlleva a una sola función. Se suma la distancia inicial con respecto al vehículo ego a la función cinemática estándar. Adicional, para obtener la distancia mínima entre ambas, simplemente se restan ambas funciones.

```
%% TRAYECTORIA VEHICULO DELANTERO (Lead vehicle)
x2 = r + v2*t + 0.5*a2*(t.^2);
```

Por último, se creó una última función por partes, para obtener los datos de velocidad de ambos vehículos igualmente en ambos periodos.

```
%% Velocidad vehículos
vh1 = @(t)v1 + a1*t;
vh2 = vh1(1.5) + am*(t-1.5);
vh = piecewise(t<=1.5, vh1, t>1.5, vh2);
vl = v2 + a2*t;
```

## 2.4 Especificaciones técnicas del producto

**Tabla 2.6. Especificaciones técnicas del sistema físico FCW**

| Parámetro | Valor | Observaciones |
|---|---|---|
| Corriente máxima | 1.5 A | |
| Voltaje de Operación | 12V Alimentación<br>5V Dispositivos | Se conecta a la toma de voltaje DC del vehículo |
| Precisión Rango | Min: ~1cm<br>Max: ~11cm | Depende de la intensidad de señal del Lidar |
| Precisión Velocidad Relativa | Min: ~0.5km/h<br>Max: ~3km/h | |
| Rango de Detección | Modelo 1: 40m<br>Modelo 2: 100m | Modelo 1: Lidar TD02-Pro<br>Modelo 2: Lidar TF03 |
| Tiempo de reacción del sistema | Max: ~250ms | |

# CAPÍTULO 3

## 3. RESULTADOS Y ANÁLISIS

### 3.1 Análisis de la magnitud de ruido en la adquisición de datos del Lidar TF-Luna

El dispositivo más importante del sistema FCW corresponde al sensor de distancia, ya que su rendimiento en cuanto a la adquisición de datos de distancia es lo que se encuentra más estrechamente relacionado con la fiabilidad de todo el sistema en sí. Con esto me refiero a que, en caso de que dicho sensor presente baja precisión en la toma de datos, se volverían inviables los cálculos realizados en el algoritmo en el sentido que se distanciarían demasiado de su contraparte teórica.

Afortunadamente, la ficha técnica de los dos modelos de Lidar elegidos para el diseño del sistema (TF02-Pro & TF03) presentan en su ficha técnica un buen nivel de precisión tal que no debería presentar inconvenientes de acuerdo al uso dado en este proyecto. Sin embargo, para comprobar la veracidad de estos datos, se adquirió un modelo de Lidar a escala perteneciente del mismo rango de productos (TF-Luna), con el cual se adquirieron datos a diferentes niveles de intensidad de señal para comprobar la variación del error medido, con el error mencionado en su ficha técnica.

Cabe recalcar que la precisión de cada Lidar presenta un valor fijo en su ficha de datos correspondiente, pero este puede variar de acuerdo a la intensidad de señal y a la reflectividad del objeto medido. Un valor de intensidad superior a 2000 representa una buena señal, mientras que un valor menor a 400 presenta una calidad de señal disminuida. La tabla presentada por la ficha de datos del TF-Luna compara la intensidad de señal con la desviación estándar del valor de distancia medido.

**Tabla 3.1. Relación de Intensidad y precisión para Lidar TF-Luna**

| **Intensidad** | **100** | **200** | **400** | **1000** | **>2000** |
|---|---|---|---|---|---|
| **STD: 1σ** | >±3cm | ±3cm | ±2cm | ±1cm | ±0.5cm |

Se realizan mediciones a tres objetos distintos, las dos primeras a la misma distancia pero con un objeto de alta luminosidad y otro de baja. La tercera a una distancia corta pero con un objeto de muy baja luminosidad.

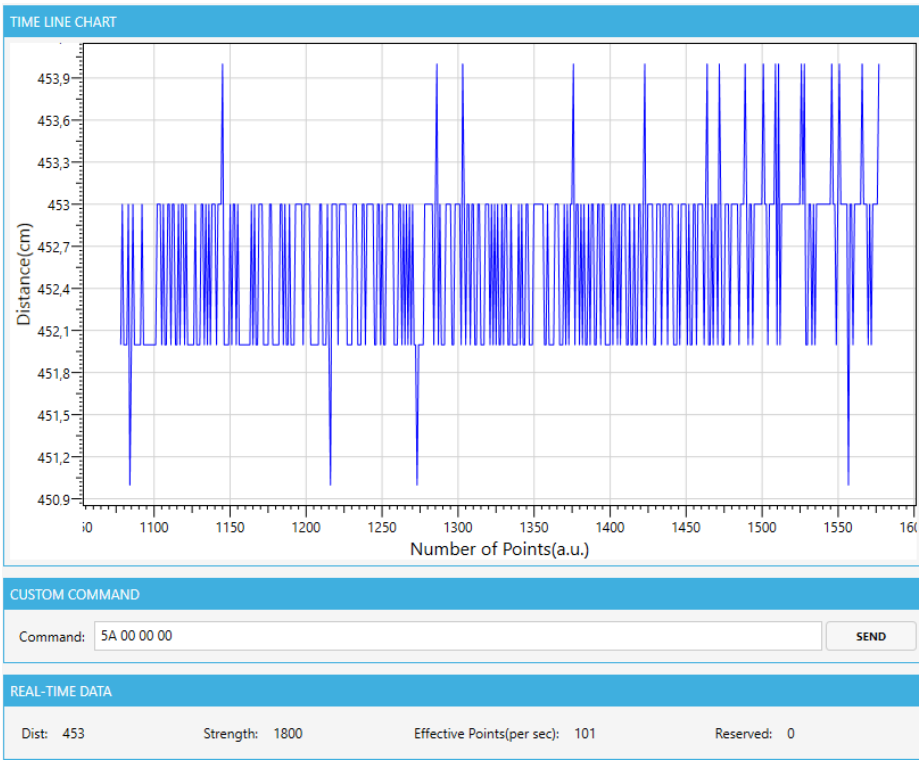



**Ilustración 3.1. Adquisición de datos Lidar TF-Luna con fuerza de señal de 1800**

La primera medición se tomó a 4.53 metros de distancia, aproximadamente un 57% del rango máximo del sensor, obteniendo una intensidad de 1800. Existe un ruido prevalente en la señal de distancia, oscilando entre los valores de 453 a 452cm, con picos en 454 y 451 cm. Esto es consistente con la tabla de relación de intensidad y precisión, ya que en la gran mayoría de tiempo se obtiene una precisión de ±0.5cm, mientras que a menor cantidad se obtiene una precisión de ±1cm.

Para la segunda medición se obtienen precisiones variadas, prevaleciendo la de ±0.5cm pero con una mayor aparición de picos de ±1cm y ±2cm. En cuanto a la tercera medición, a pesar de que la distancia de 1.68 metros es considerablemente menor a las dos anteriores, la intensidad de señal es la más baja, resultando en un valor promedio de 300. Asimismo, aparecen nuevos picos que equivalen a un decrecimiento de precisión, obteniendo desde ±3 hasta ±4cm. Cabe recalcar que una precisión de ±4cm puede conllevar a valores de hasta 8cm de diferencia entre mediciones de distancia consecutivas, y al aplicar diferenciación para encontrar velocidad y aceleración, puede magnificarse dicho error afectando el rendimiento del sistema.

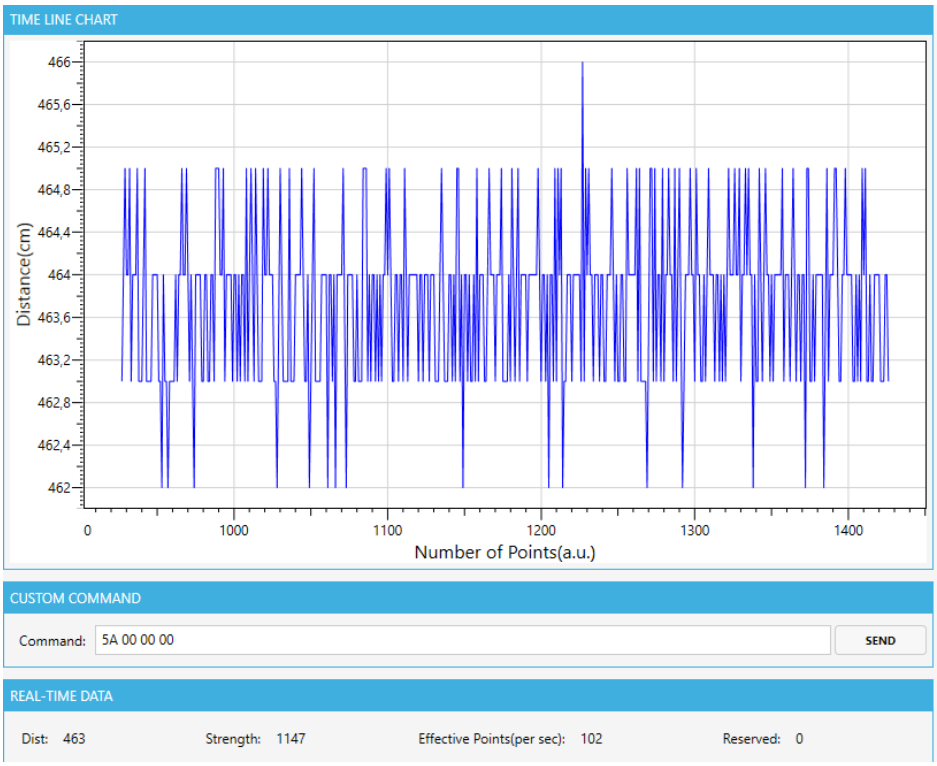



**Ilustración 3.2. Adquisición de datos Lidar TF-Luna con fuerza de señal de 1100**

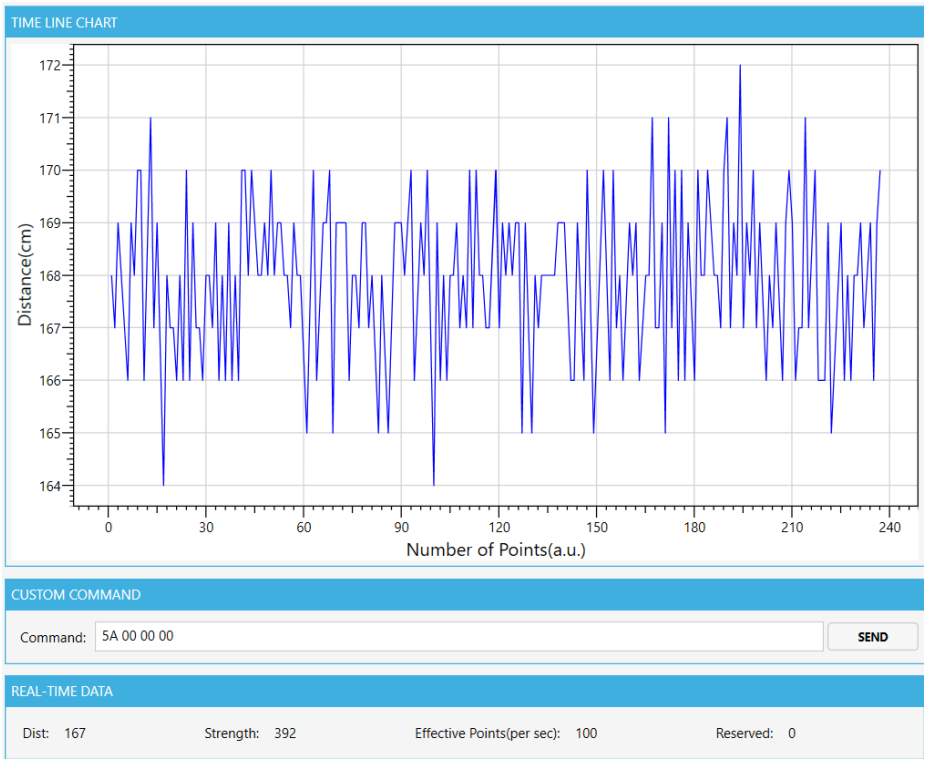



**Ilustración 3.3. Adquisición de datos Lidar TF-Luna con fuerza de señal de 300**

Las gráficas anteriores demuestran que la distancia hacia el objeto influye poco en la aparición de una mayor magnitud de ruido en la señal, ya que depende directamente de la intensidad lumínica del objeto influyendo en el valor de intensidad.

La siguiente imagen muestra la superficie usada en la primera y tercera medición, en la cual se observa claramente el contraste entre ambas luminosidades, por lo que se entiende la razón de su efecto en el ruido de la señal.

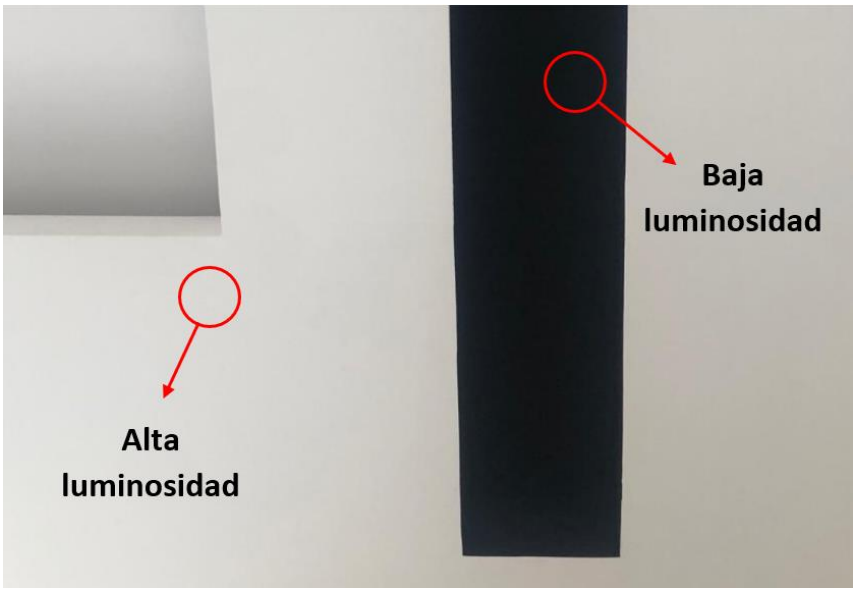



**Ilustración 3.4. Superficie usada en parte de las mediciones de intensidad de señal y ruido del sensor Lidar.**

De acuerdo a los valores obtenidos, se formularon magnitudes de ruido para la salida de datos de sensor en las simulaciones de escenarios de conducción, de acuerdo a condiciones normales de operación, y a condiciones desfavorables en las que el objeto detectado presenta muy poca luminosidad. Estas magnitudes de ruido aparecen de acuerdo a los porcentajes asociados a una curva de distribución normal asociada a la desviación estándar.

**Tabla 3.2. Magnitudes pico a pico de ruido para distintas condiciones de señal del Lidar.**

| | 1σ = 68.2% | 2σ = 27.2% | >3σ = 4.6% |
|---|---|---|---|
| Condición Normal | 1cm | 3cm | 5cm |
| Condición Desfavorable | 6cm | 8cm | 11cm |

## 3.2 Análisis del uso de distintas opciones de tasas de refresco en la obtención de velocidad y aceleración relativa.

La búsqueda de una óptima tasa de refresco para la obtención de velocidad y aceleración relativa surge de la necesidad de maximizar el tiempo de respuesta del sistema, y al mismo tiempo, minimizar la magnificación del error presente como ruido en los datos de distancia del sensor Lidar, hacia las variables mencionadas. Se tendrá presente la magnitud de ruido tanto en condición normal como en condición desfavorable.

Para comprobar el tiempo de respuesta del sistema se simulará el primer caso del escenario 4 de conducción. Ya que dicho escenario trata de una acción que sucede rápida y espontáneamente, la efectividad del sistema dependerá exclusivamente de qué tan rápido puede generar una alerta ante tal eventualidad.

En aquel mismo escenario, también se pretende comprobar que el error magnificado sea lo suficientemente pequeño como para no generar alertas falsas en periodos de tiempo donde el peligro de colisión es nulo, y también para no influir en las ecuaciones significativamente como para crear datos imprecisos.

Teniendo en cuenta que la tasa de refresco de 100Hz del sensor Lidar es fija, se ha preparado una tabla que indica las diferentes combinaciones de tasas de refresco para velocidad y aceleración relativa que se desean poner a prueba, así como el factor que ha decrementado con respecto a la variable que le antecede, y la cantidad de tiempo que demoraría cada variable en cambiar su valor.

**Tabla 3.3. Opciones de tasas de refresco para velocidad y aceleración relativa.**

| Variables | Opción 1 | | Opción 2 | | Opción 3 | | Opción 4 | |
|---|---|---|---|---|---|---|---|---|
| | Hz | ms | Hz | ms | Hz | ms | Hz | ms |
| Velocidad Relativa | 20 | 50 | 3.33 | 300 | 6.67 | 150 | 4 | 250 |
| Aceleración Relativa | 6.67 | 200 | 1.67 | 600 | 3.33 | 300 | 4 | 250 |

Con respecto a la primera opción, se eligió los valores priorizando el tiempo de respuesta del sistema. Para la segunda opción, se priorizó la minimización del error. Para la tercera y cuarta opción, se intentó encontrar un balance óptimo entre rapidez y error. De acuerdo a la mejor de estas opciones se emplearán los valores asociados para el análisis del resto de escenarios y casos de conducción.

### 3.2.1 Simulación y valores teóricos de algoritmo FCW para Escenario 3, Caso 1

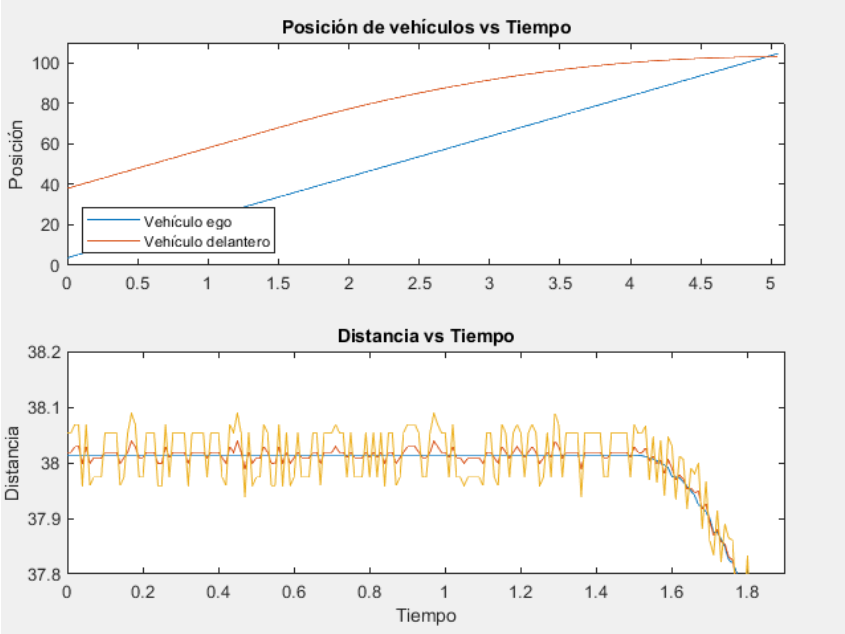



**Ilustración 3.5. Graficas de Posición y Distancia vs Tiempo de Escenario 4, Caso 1**

Tal y como sucede con la mayoría de los escenarios de conducción diseñados, para este caso está prevista una colisión inminente a los 5 segundos, evidenciado por la gráfica de posición de vehículos con respecto al tiempo. Asimismo, se puede observar la magnitud de los tres niveles de ruido en la gráfica de distancia con respecto al tiempo, la cual fue acercada a décimas alrededor del valor real para apreciar la diferencia entre niveles.

Una vez ingresado los datos en el algoritmo FCW que recibe exclusivamente los valores teóricos de cada variable usada en el sistema, se obtuvo la siguiente salida:

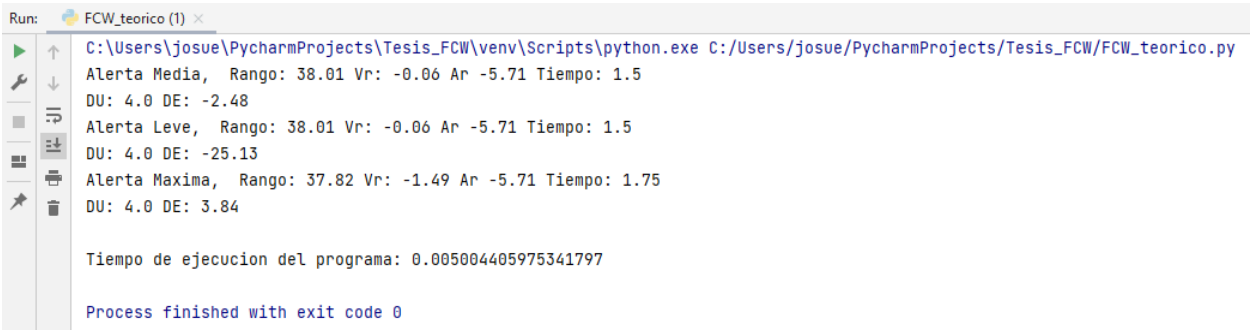



**Ilustración 3.6. Salida de alertas algoritmo FCW teórico para Escenario 4, Caso 1**

Las alertas leve y media se activaron simultáneamente a los 1.5 segundos, sin embargo, frenar acorde al valor de desaceleración asociado a dichas alertas no se podrá impedir colisión. A los 1.75 segundos se activa la alerta máxima, la cual calcula que si se podrá evitar la colisión con una distancia de elusión de 3.84 metros. Esto corresponde a un comportamiento normal del sistema, ya que en estos tipos de escenarios críticos donde el vehículo delantero ha accionado los frenos con una aceleración cercana al máximo del vehículo ego (0.6g), este último debe reaccionar con una acción similar.

### 3.2.2 Primera Opción: Maximización del tiempo de respuesta

La maximización del tiempo de respuesta consiste en usar un diferencial de tiempo con un valor relativamente bajo para obtener tanto velocidad como aceleración relativa, tal que el algoritmo FCW pueda actualizar sus cálculos y emitir una alerta en un tiempo muy corto. Los valores usados son aquellos mencionados en la tabla 3.3, para Vr se tiene 20Hz y para Ar 5Hz.

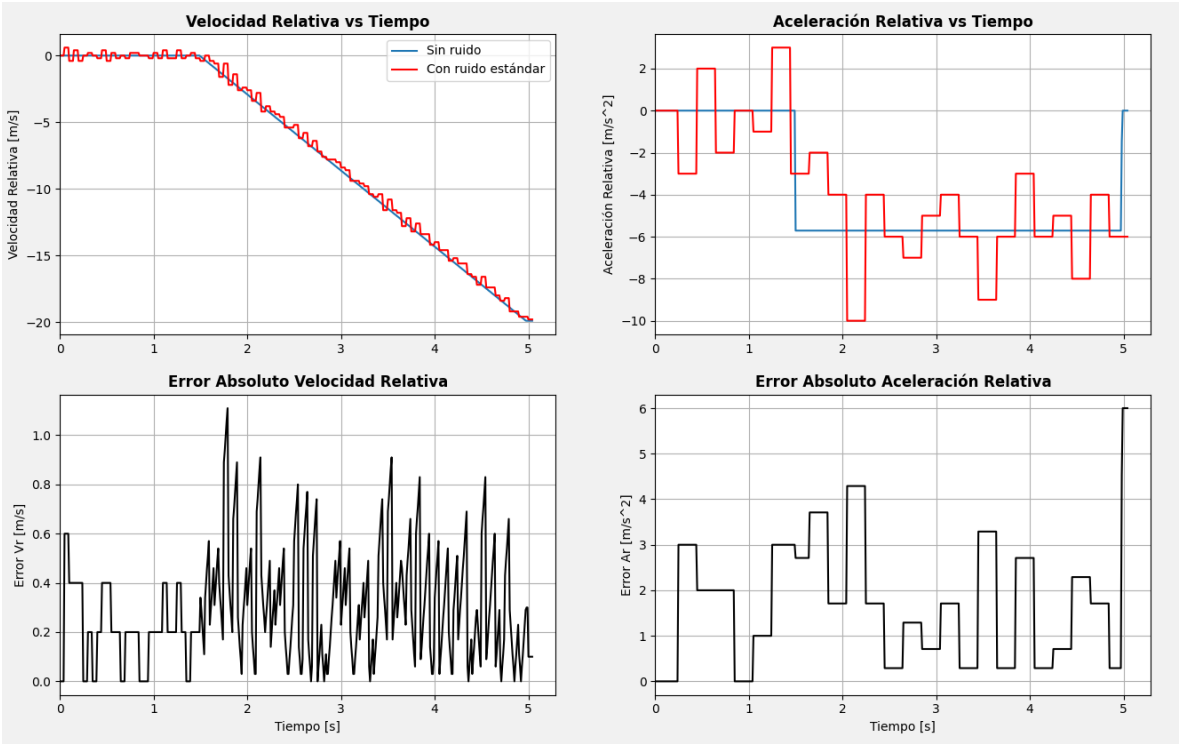



**Ilustración 3.7. Simulación del Escenario 4 - Caso 1, TR: Op.1, Ruido normal**

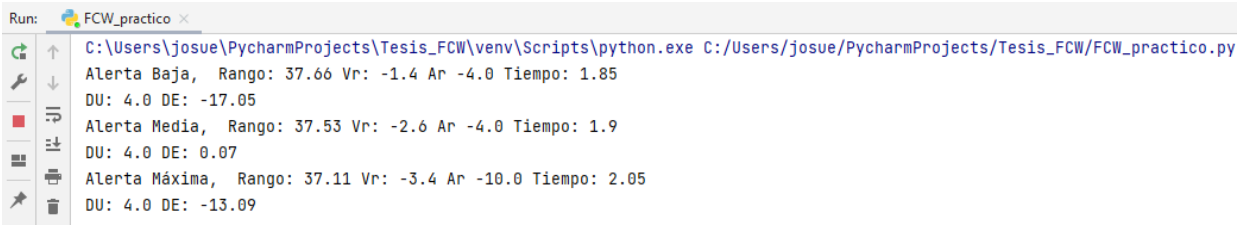



**Ilustración 3.8. Salida de alertas de algoritmo FCW para Escenario 4 - Caso 1, TR: Op.1, Ruido normal**

A pesar de que el objetivo de elegir una alta tasa de refresco era para maximizar el tiempo de respuesta, se obtuvo un resultado contraproducente en el sentido que el error es lo suficientemente grande como para influir negativamente en la generación de la alerta. Poco después de los 1.75 segundos, momento en que teóricamente la alarma debió de activarse, se aprecia un error en la aceleración que aumenta positivamente su valor al sentido opuesto del valor real, retrasando aún más la generación de la alerta a 2.05 segundos.

Con respecto a los errores absolutos, para la velocidad relativa este llega al 1.1m/s, lo cual es un valor lo suficientemente bajo como para considerarse aceptable. Sin embargo, la aceleración llega a tener un error de hasta 4.3m/s^2, que corresponde a un valor lo suficientemente grande como para volver inválida a la variable para su uso en las ecuaciones del sistema. Para el mismo escenario anterior ahora simulado con datos de distancia desfavorables se obtuvieron las siguientes gráficas:

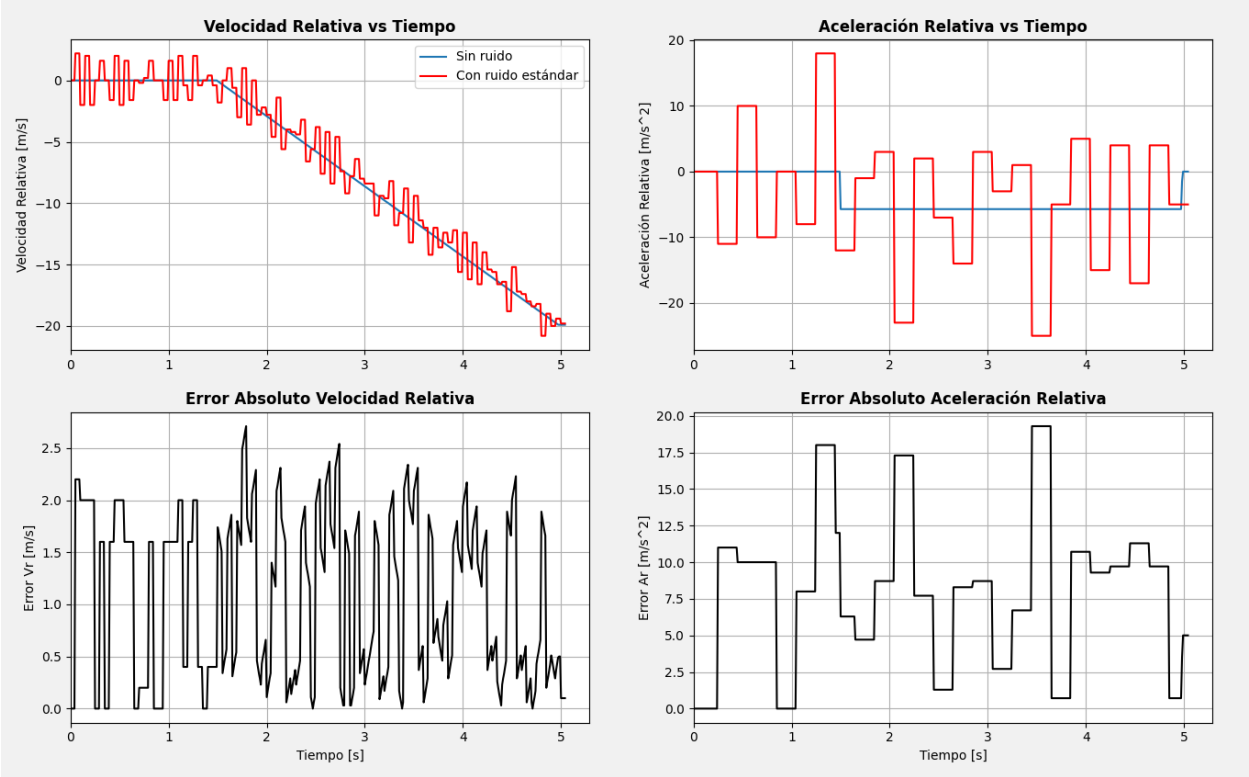



**Ilustración 3.9. Simulación del Escenario 4 – Caso 1, TR: Op.1, Ruido desfavorable**

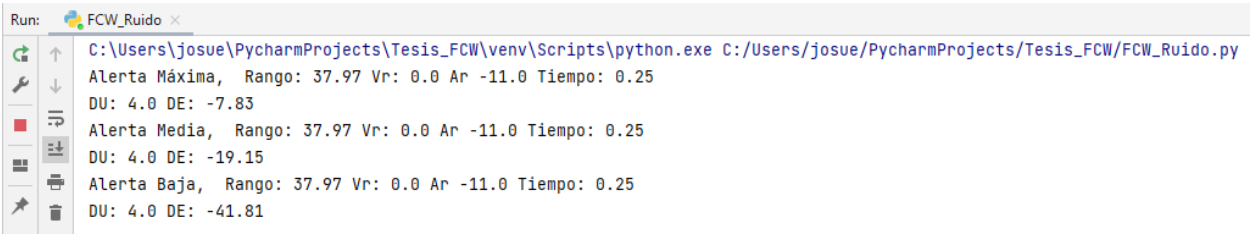



**Ilustración 3.10. Salida de alertas de algoritmo FCW para E4C1, TR: Op.1, Ruido desfavorable**

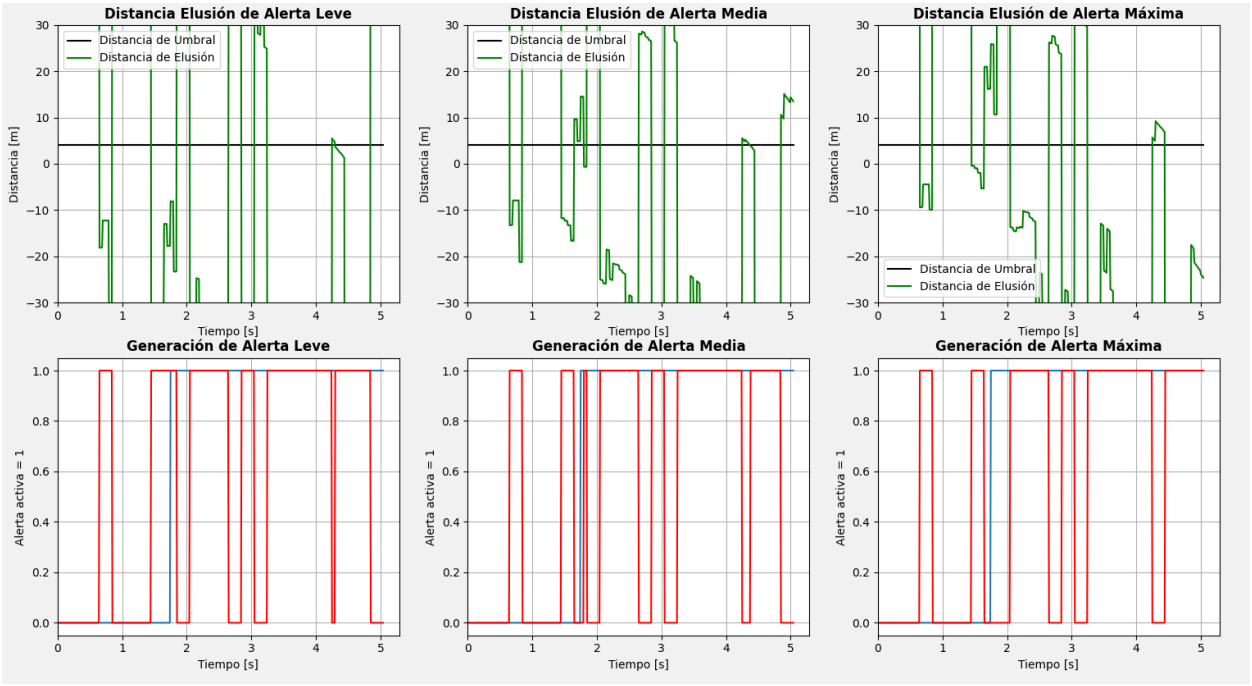



**Ilustración 3.11. Gráficas de alertas de algoritmo FCW para E4C1, TR: Op.1, Ruido desfavorable**

Una vez que se ejecuta el algoritmo FCW con las condiciones desfavorables de error, se pierde completa fiabilidad en el sistema, ya que este se comporta erráticamente a lo largo de todo el tiempo de simulación, y genera alertas aleatoriamente en tiempos no adecuados. Por lo tanto, sólo elegir maximizar el tiempo de respuesta no es el enfoque correcto para mejorar la fiabilidad del sistema.

### 3.2.3 Segunda Opción: Minimización del error

La maximización del tiempo de respuesta consiste en usar un diferencial de tiempo con un valor relativamente alto para obtener tanto velocidad como aceleración relativa, tal que el algoritmo FCW pueda recibir datos con poco error y emitir alertas confiables, sin falsas alarmas. Los valores usados son aquellos mencionados en la tabla 3.3, para Vr se tiene 5Hz, y para Ar 1.67Hz.

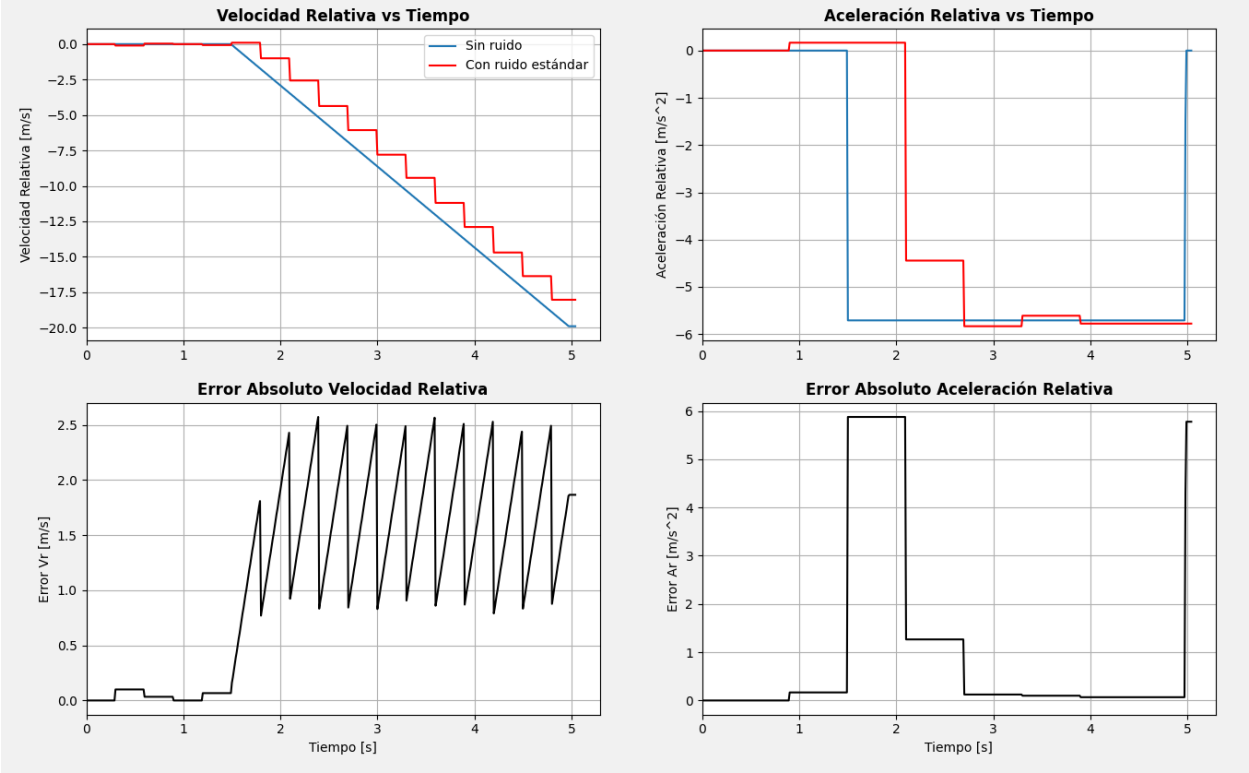



**Ilustración 3.12. Simulación del Escenario 4 - Caso 1, TR: Op.2, Ruido normal**

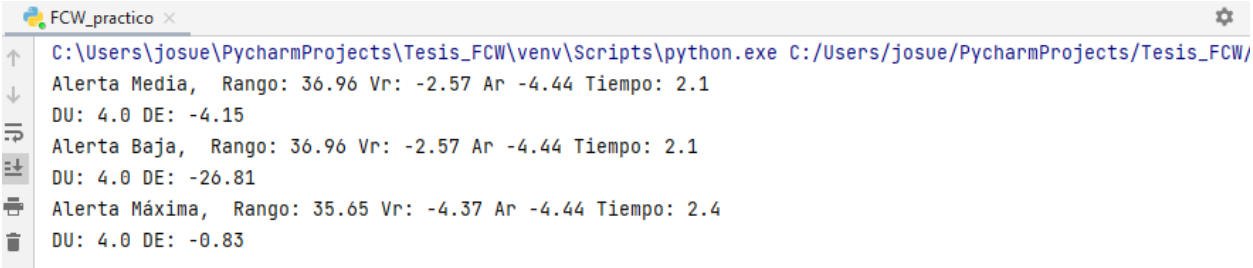



**Ilustración 3.13. Salida de alertas de algoritmo FCW para E4C1, TR: Op.2, Ruido normal**

La ilustración 3.14 muestra un aparente mayor error máximo tanto en velocidad como aceleración relativa, pero este suceso es producto simplemente del retraso en el tiempo que tiene la variable con su contraparte. El error verdadero es aquel que se mide cuando la variable se encuentra en un periodo de tiempo sin cambios en su valor. Por lo tanto, se puede apreciar que para la velocidad relativa el error es menor a 0.1 m/s y para la aceleración es de 0.2 m/s^2. Esto representa una mejora significativa con respecto a la anterior opción, pero lamentablemente, lo que se gana en minimizar el error en los datos, se pierde en el hecho que al sistema le toma hasta 650ms en generar una alerta para una eventualidad de frenado brusco.

Con respecto a la simulación del sistema empleando la magnitud de error desfavorable en los datos de distancia, se obtienen resultados muy similares. En síntesis, el error de las variables se vuelve despreciable, pero el tiempo de respuesta se mantiene igual de elevado.

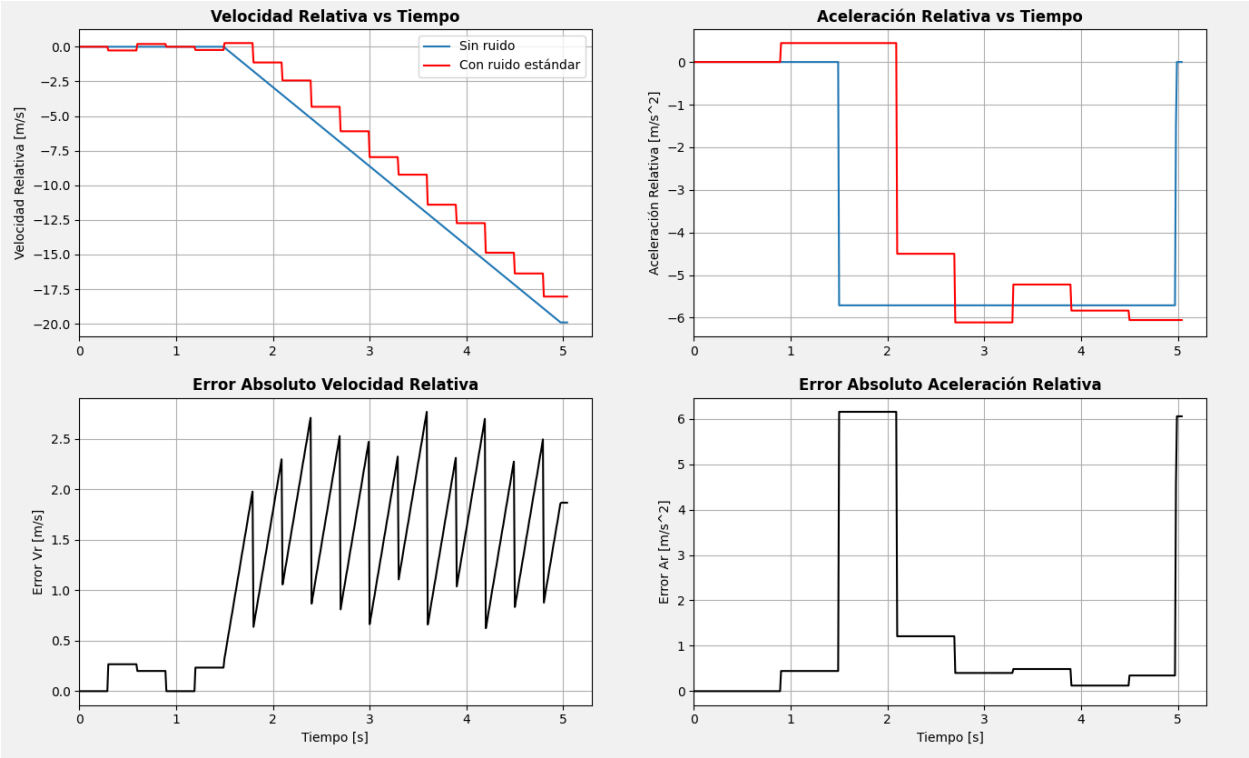



**Ilustración 3.14. Simulación del Escenario 4 - Caso 1, TR: Op.2, Ruido desfavorable**

### 3.2.4 Tercera Opción: Balance entre tiempo de respuesta y error, TR doble para aceleración relativa.

Buscando un mejor tiempo de respuesta que la opción 2, se disminuye a la mitad cada uno de los tiempos de actualización de cada variable de acuerdo a las utilizadas en dicha opción, tal que las tasas de refresco ahora son de 6.67Hz para la velocidad relativa, y 3.33Hz para la aceleración relativa.

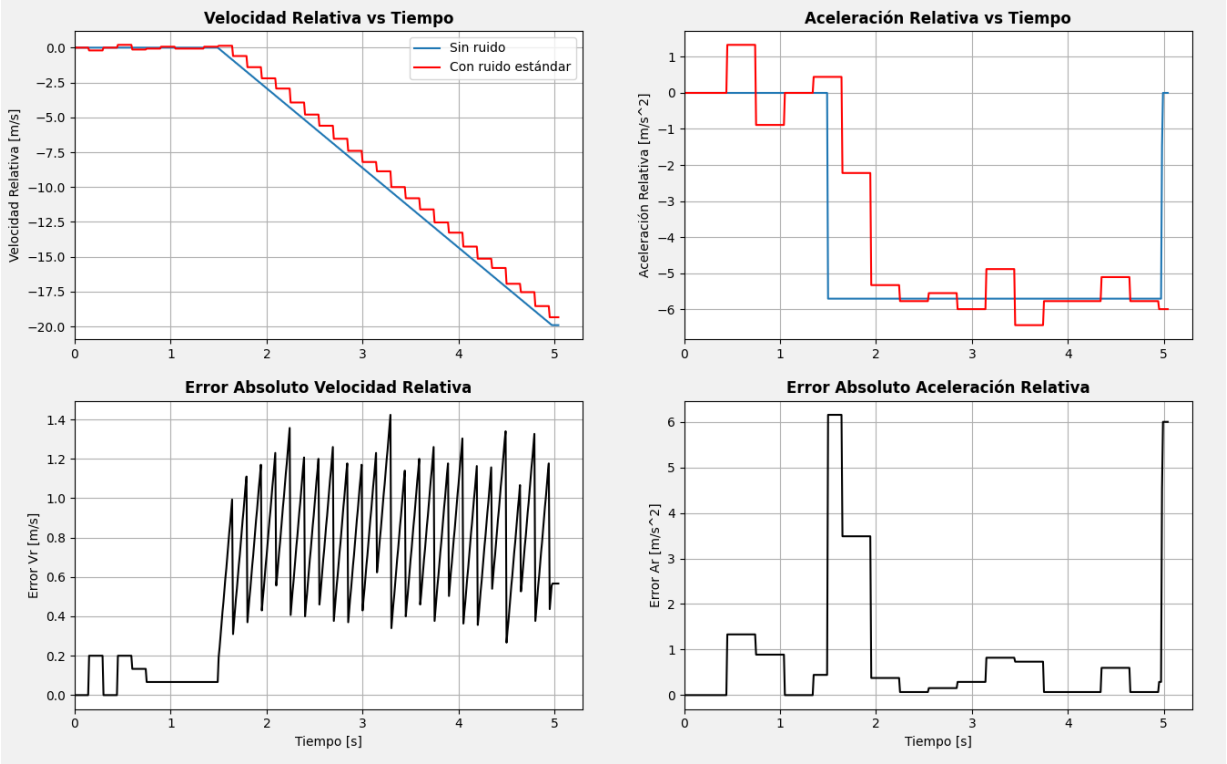



**Ilustración 3.15. Simulación del Escenario 4 - Caso 1, TR: Op.3, Ruido normal**

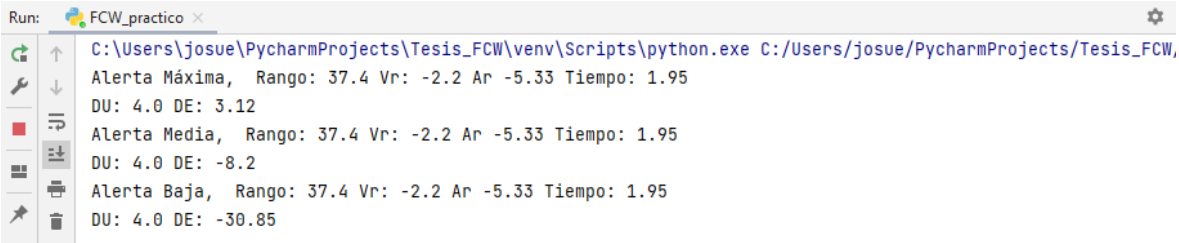



**Ilustración 3.16. Salida de alertas de algoritmo FCW para E4C1, TR: Op.3, Ruido normal**

Para esta opción, todas las alertas se activaron al mismo tiempo a los 1.95 segundos, tan solo 200ms después del tiempo teórico de 1.75 segundos. El error de la velocidad relativa es de apenas 0.2m/s en tiempo estable, mientras que la aceleración, en la mayoría de los casos es inferior a 1m/s^2. Por ahora, esta opción es la más aceptable de todas las analizadas hasta el momento. Sin embargo, también debe ser igual de efectivada al obtener datos con un error desfavorable, lo que se muestra en las siguientes gráficas.

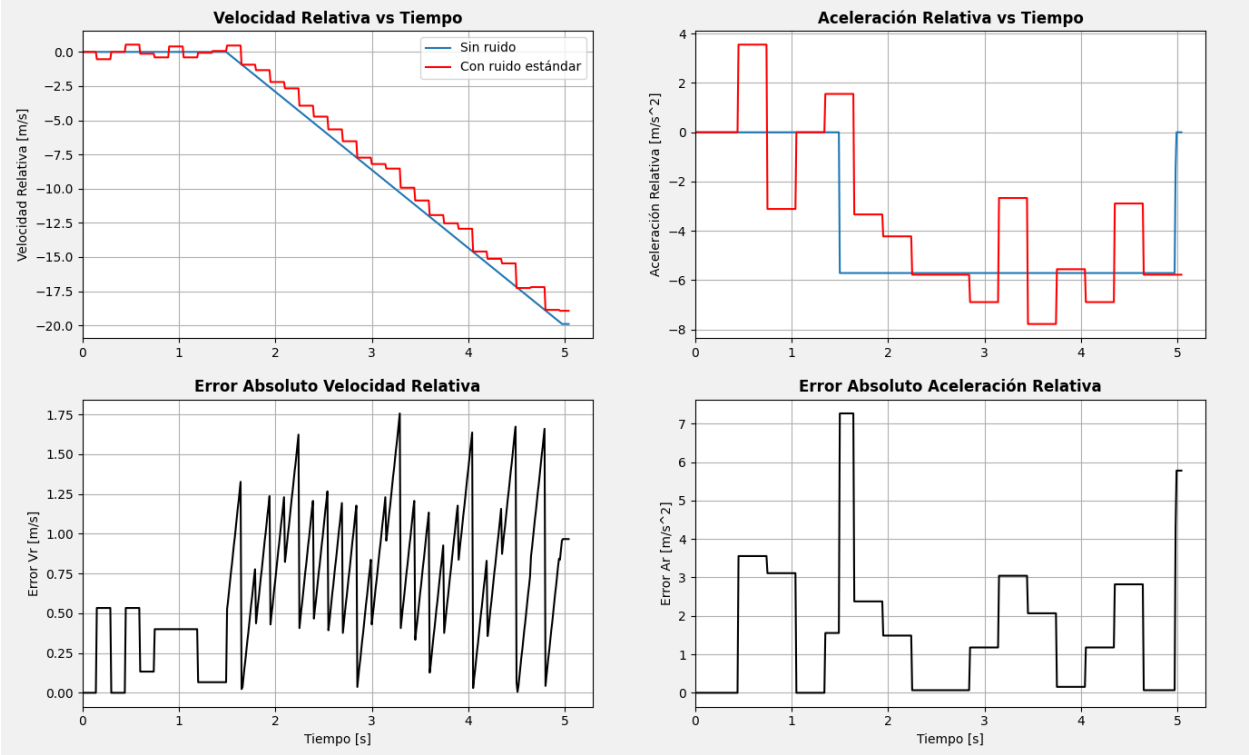



**Ilustración 3.17. Simulación del Escenario 4 - Caso 1, TR: Op.3, Ruido desfavorable**

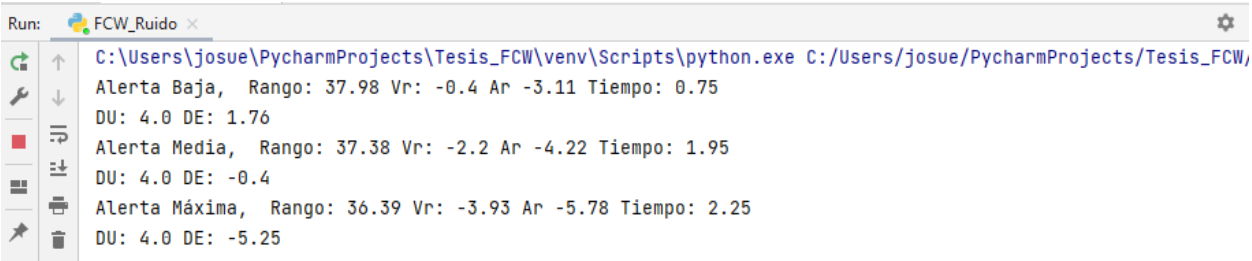



**Ilustración 3.18. Salida de alertas de algoritmo FCW para E4C1, TR: Op.3, Ruido desfavorable**

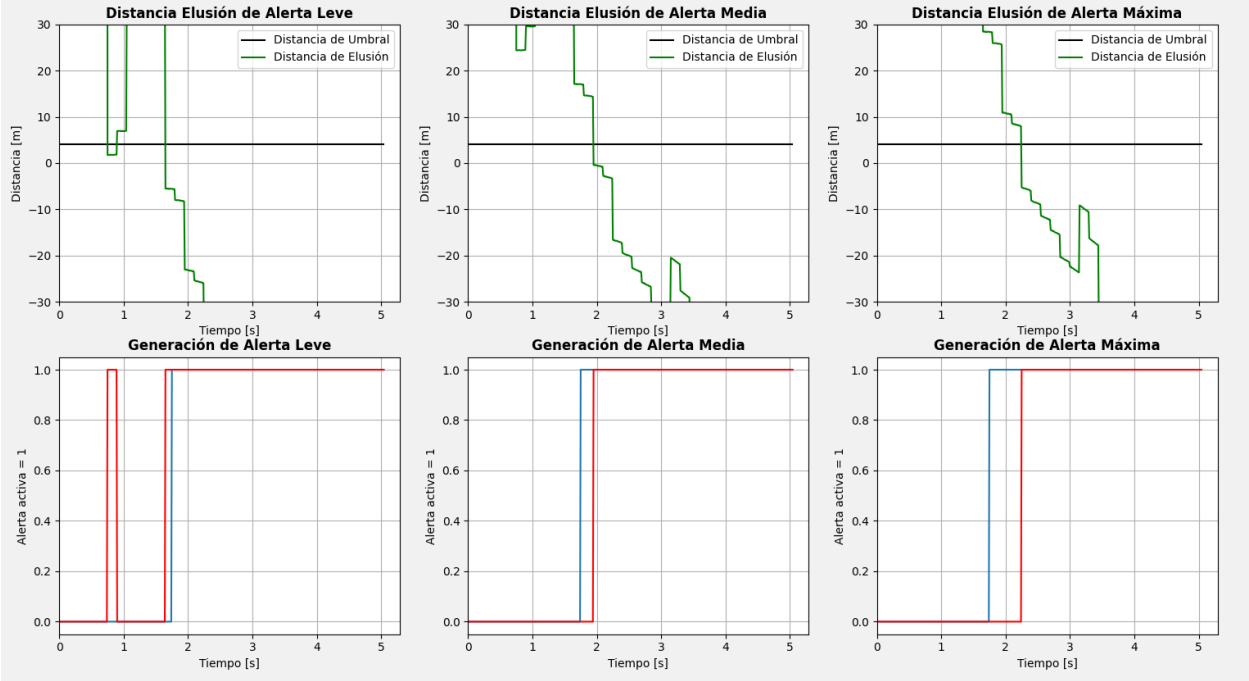



**Ilustración 3.19. Gráficas de alertas de algoritmo FCW para E4C1, TR: Op.3, Ruido desfavorable**

Para la misma simulación pero con error desfavorable, la alerta media presenta el mismo tiempo de reacción de 1.95 segundos. Sin embargo, la alerta máxima aparece a los 2.25 segundos, es decir que se retrasa 300ms debido al efecto del error en la aceleración relativa, el cual tiene una magnitud considerable que supera en ocasiones los 3m/s^2. Por esta misma razón, la alerta leve se vuelve inconsistente e inutilizable, en el sentido que en momentos sin riesgo de colisión, esta se activa tan sólo por el cálculo de datos con error significativo.

### 3.2.5 Cuarta Opción: Balance entre tiempo de respuesta y error, igual TR para velocidad y aceleración relativa.

A pesar del buen rendimiento del sistema en la tercera opción bajo la simulación de un error normal, es necesario mejorar el rendimiento del sistema con ruido desfavorable, aumentando la TR para la velocidad a 4Hz, e igualando ese dato para la aceleración. Esto se consigue realizando una diferenciación dato a dato a los valores de velocidad. Ambas variables entonces actualizarán sus datos al algoritmo FCW cada 250ms.

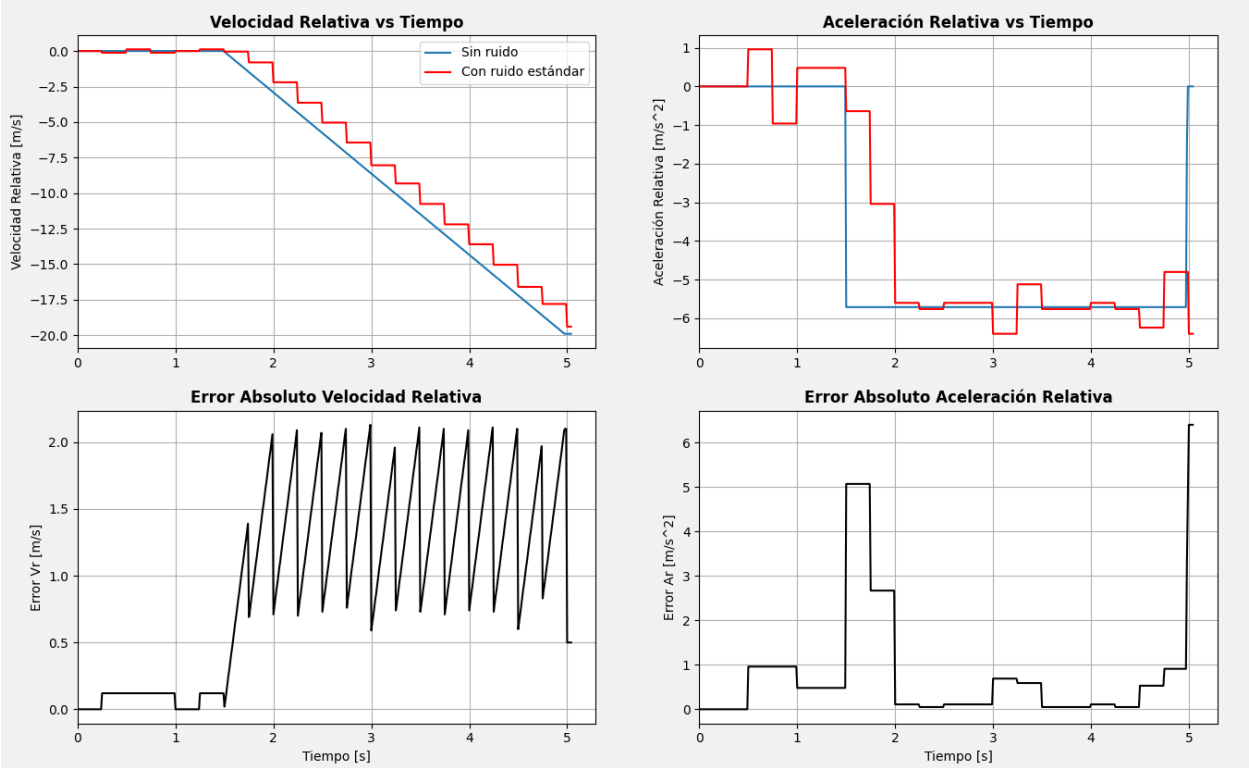



**Ilustración 3.20. Simulación del Escenario 4 - Caso 1, TR: Op.4, Ruido normal**

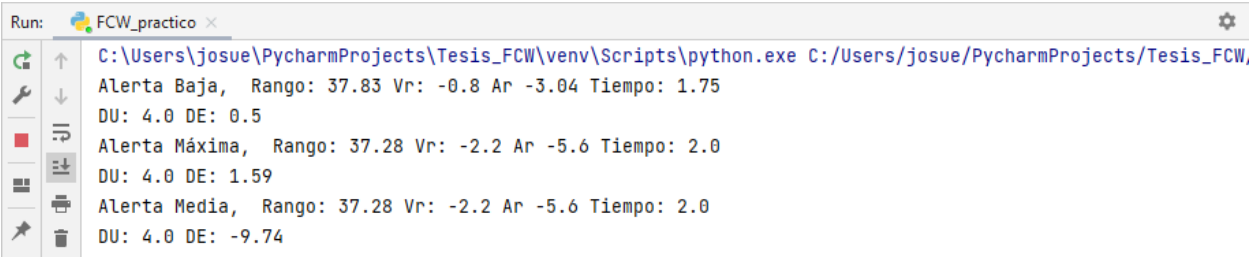



**Ilustración 3.21. Salida de alertas de algoritmo FCW para E4C1, TR: Op.4, Ruido normal**

Para esta opción, se mejora ligeramente el error con una penalización de 50ms al tiempo de respuesta de las alertas máxima y media. Se encuentran menos picos de error en la gráfica de aceleración relativa, estando muy cerca al valor real en la mayor parte del tiempo del estado estable. Entonces, entre la tercera opción y cuarta opción no se ve un cambio considerable al simular el sistema con un error normal. Sin embargo, la razón de haber creado esta cuarta opción fue para mejorar el rendimiento con un error desfavorable, lo cual se muestra en las gráficas a continuación.

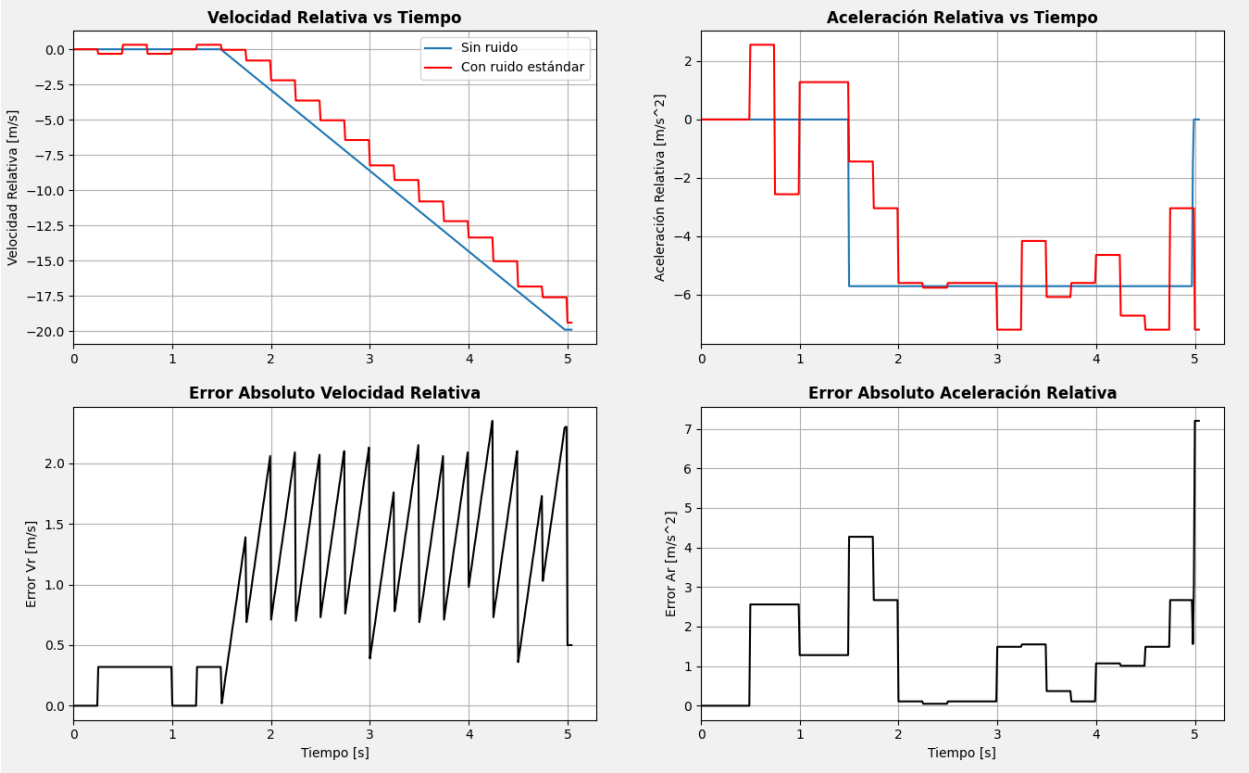



**Ilustración 3.22. Simulación del Escenario 4 - Caso 1, TR: Op.4, Ruido desfavorable**

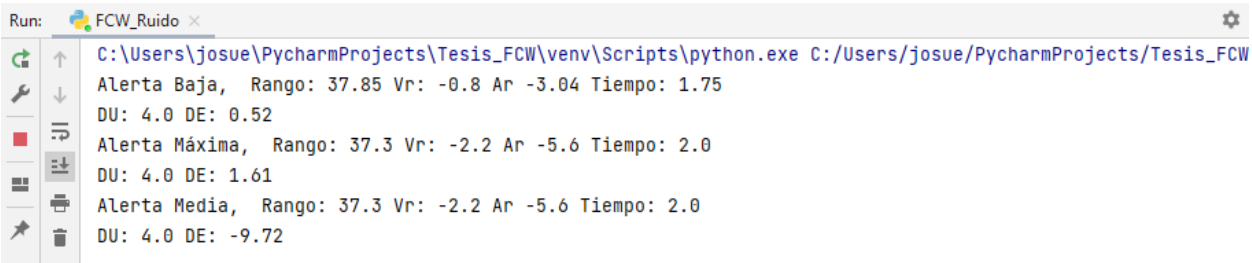



**Ilustración 3.23. Salida de alertas de algoritmo FCW para E4C1, TR: Op.4, Ruido desfavorable**

En cuanto al rendimiento del sistema bajo un ruido desfavorable, la opción 4 de tasa de refresco tuvo una mejora considerable, igualando por completo los tiempos de alerta presentes con ruido normal, y presentando como promedio de error en la aceleración relativa un valor menor a 1.6 m/s^2. También, un problema que se presentó en la opción anterior es la aparición de falsas alertas leves debido a la magnitud del error, cuestión que no ocurre de acuerdo a la siguiente gráfica.

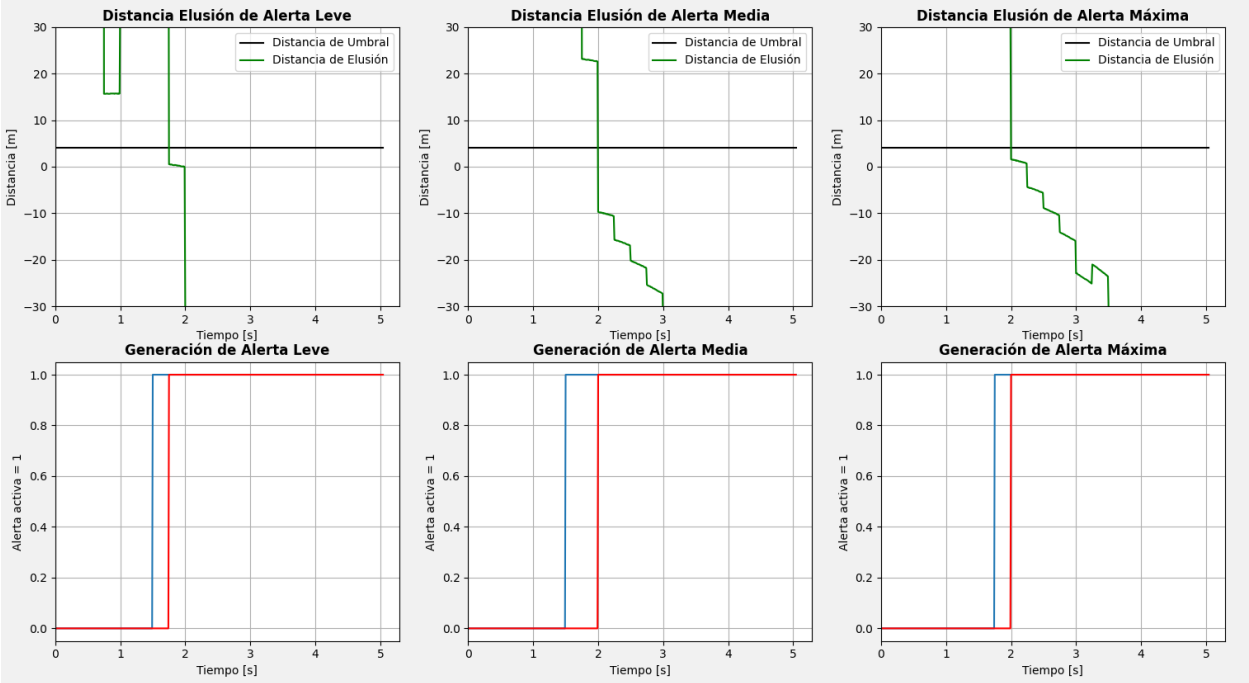



**Ilustración 3.24. Gráficas de alertas de algoritmo FCW para E4C1, TR: Op.4, Ruido desfavorable**

### 3.2.6 Comparación de resultados obtenidos

Se realizó una tabla recopilando todos los resultados asociados a las 4 opciones de tasas de refresco para la velocidad y aceleración relativa, usando la diferencia del tiempo de respuesta obtenido y el teórico, el error absoluto en estado estable y la generación de alertas falsas, como métodos para determinar la opción idónea para emplear en la simulación del resto de escenarios de simulación creados para este proyecto. Un tiempo de respuesta es inválido (Inv.) cuando es producto de una alerta falsa o pico de error.

**Tabla 3.4. Comparación de rendimiento de sistema FCW para las opciones de TR.**

| Opción | Ruido | Diferencia Tiempo de Respuesta [ms] | Error Vr [m/s] | Error Ar [m/s^2] | Generación de alertas falsas |
|---|---|---|---|---|---|
| 1 | Normal | Inv | ~0.6 | ~4.3 | Leve |
| | Desfavorable | Inv | ~2 | ~19 | Todas |

| | | | | | |
|---|---|---|---|---|---|
| 2 | Normal | 650 | ~0.1 | ~0.2 | No |
| | Desfavorable | 650 | ~0.3 | ~0.5 | |
| 3 | Normal | 200 | ~0.2 | ~1.3 | |
| | Desfavorable | 500 | ~0.6 | ~3.6 | Leve |
| 4 | Normal | 250 | ~0.11 | ~1.0 | No |
| | Desfavorable | 250 | ~0.55 | ~2.5 | |

Ya que la opción 4, en la cual tanto la velocidad como la aceleración relativa se actualizan acorde a una tasa de refresco de 4Hz, presentan la mejor combinación de tiempo de respuesta y de error absoluto en las variables, tal que fue la elegida para ser usada para las simulaciones en el resto de los escenarios de conducción. Por lo tanto, las características presentadas pasan a ser parte del sistema FCW en general, teniendo un retraso máximo de tan sólo 250ms, un error máximo de velocidad y aceleración relativa de 0.11m/s y 1.0m/s^2 en condiciones normales. Sin embargo, en condiciones desfavorables, se sigue teniendo un error un error relativamente alto de estas mismas variables de 0.55m/s y 2.5m/s^2, por lo que para esta situación se requiere modificar la tasa de refresco dinámicamente en caso de que se detecte poca intensidad de señal.

Esto se resuelve añadiendo una condicional al código, que dependiendo el nivel de intensidad de la señal Lidar, cambia datos a valores que corresponden al número de datos por tiempo a calcular tanto para velocidad como para aceleración relativa, en otras palabras, se modifica la tasa de refresco.

```
    if intensidad > 500:
        a,b,dt1,dt2 = 25,1,0.25,0.25
    else:
        a,b,dt1,dt2 = 30,2,0.3,0.6
```

También, para cuando estas variables tengan un valor cercano a cero, simplemente se elimina el ruido aplicando una condicional al código, tal que por debajo del valor de error ya conocido, la variable se mantenga en cero. El error se conoce mediante rangos de valor de intensidad obtenidos del Lidar.

## 3.3 Ejecución del código Python de FCW para cada escenario de conducción y comprobación de no-colisión posterior a la alerta máxima.

Ya que se ha determinado la efectividad de los datos de variables cinemáticas de acuerdo al procedimiento realizado anteriormente, ya no es necesaria su visualización en conjunto con su respectivo error. En cambio, se muestra exclusivamente la posición de ambos vehículos en el curso de colisión original, la generación de alertas, así como la gráfica de predicción de la posición y velocidad futuras en caso de reaccionar a los 1.5 segundos y desacelerar a la capacidad de frenado asociado a cada alerta. De esta manera se demuestra la distancia mínima entre ambos vehículos en caso de no existir colisión, o la velocidad en el instante de colisión, en caso de que hubiese.

### 3.3.1 Simulación Escenario 1, Caso 1: Vehículo Detenido

Recordando los parámetros de este escenario, se tiene un vehículo delantero totalmente inmóvil, y el vehículo ego dirigiéndose a este a velocidad constante. El caso 1 trata del uso del Lidar de larga distancia, a una velocidad media de 60km/h. La trayectoria de colisión se observa en la siguiente gráfica.

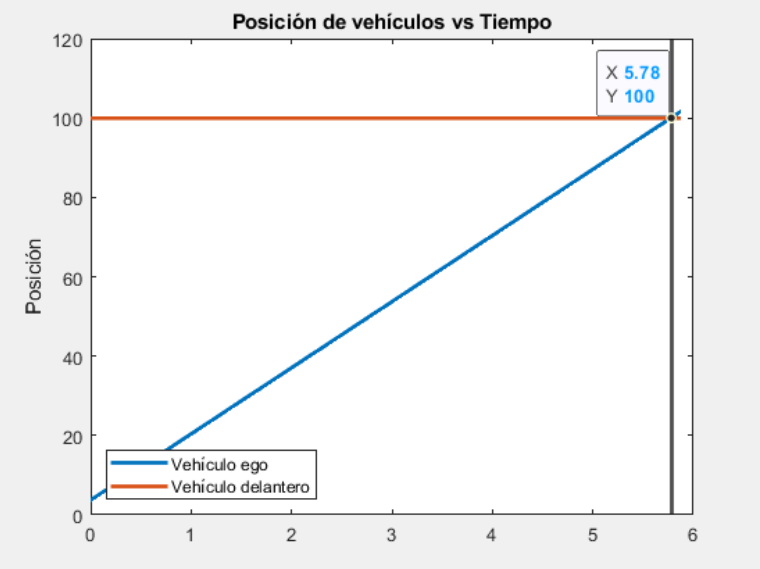



**Ilustración 3.25. Gráfica de posición de Escenario 1, Caso 1**

Tal y como se observa en la ilustración 3.27, el vehículo ego se acerca linealmente a un vehículo que no cambia su posición en el tiempo, es decir, se encuentra detenido. La colisión se produce a los 5.78 segundos. Los datos de la simulación se importan al

algoritmo FCW de Python, y se obtienen los siguientes valores con respecto a la activación de alertas.

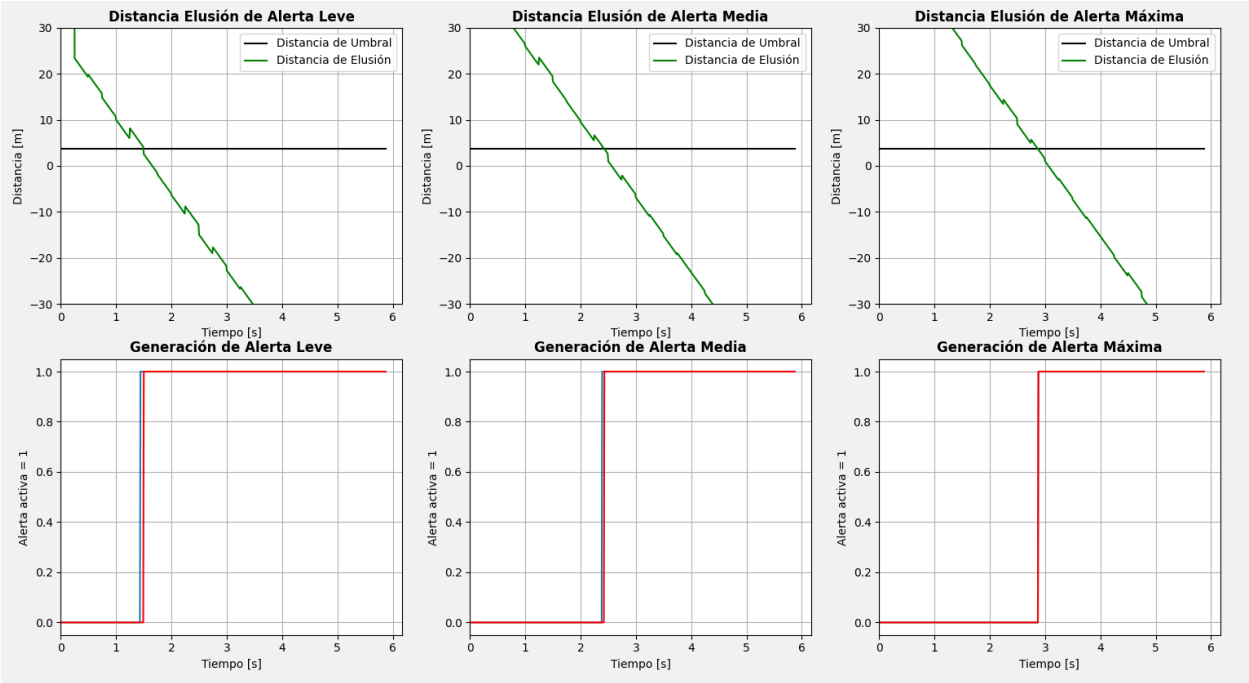



**Ilustración 3.26. Gráfica de generación de alertas Escenario 1, Caso 1**

Los gráficos de activación de alertas demuestran que, en escenarios donde el valor de las variables es poco cambiante, el retraso del sistema es mínimo, ya que no se necesita esperar la actualización del valor de la variable de velocidad o aceleración (que ocurre cada 250ms), ya que de todos modos el nuevo valor será igual al anterior. Por lo tanto, lo único que influye en el cálculo de la alerta es la actualización de la distancia (100Hz) y el error presente en las variables.

**Tabla 3.5. Tiempos de generación de alertas Escenario 1, Caso 1**

| | Alerta Leve [s] | Alerta Media [s] | Alerta Máxima [s] | Tiempo entre Al. Max. y colisión [s] |
|---|---|---|---|---|
| Real | 1.44 | 2.39 | 2.86 | 2.92 |
| Sistema | +0.06 | +0.04 | +0.02 | |

Posteriormente, se ha comprobado la prevención de colisión de acuerdo a dos posibilidades:

1) El conductor reacciona acorde a lo sugerido en el algoritmo: Tiempo de reacción de 1.5 segundos y frenado máximo de 0.6g.
2) Reacción temprana del conductor: Tiempo de reacción de 1 segundo y frenado máximo de 0.6g.

Para esto se usa el tiempo exacto de la generación de la alerta máxima, sumado con el retraso del sistema, y se recopila todas las variables cinemáticas en ese instante de tiempo. Estas son ingresadas al código de Matlab de Comprobación, y se observa la nueva gráfica de posición entre vehículos.

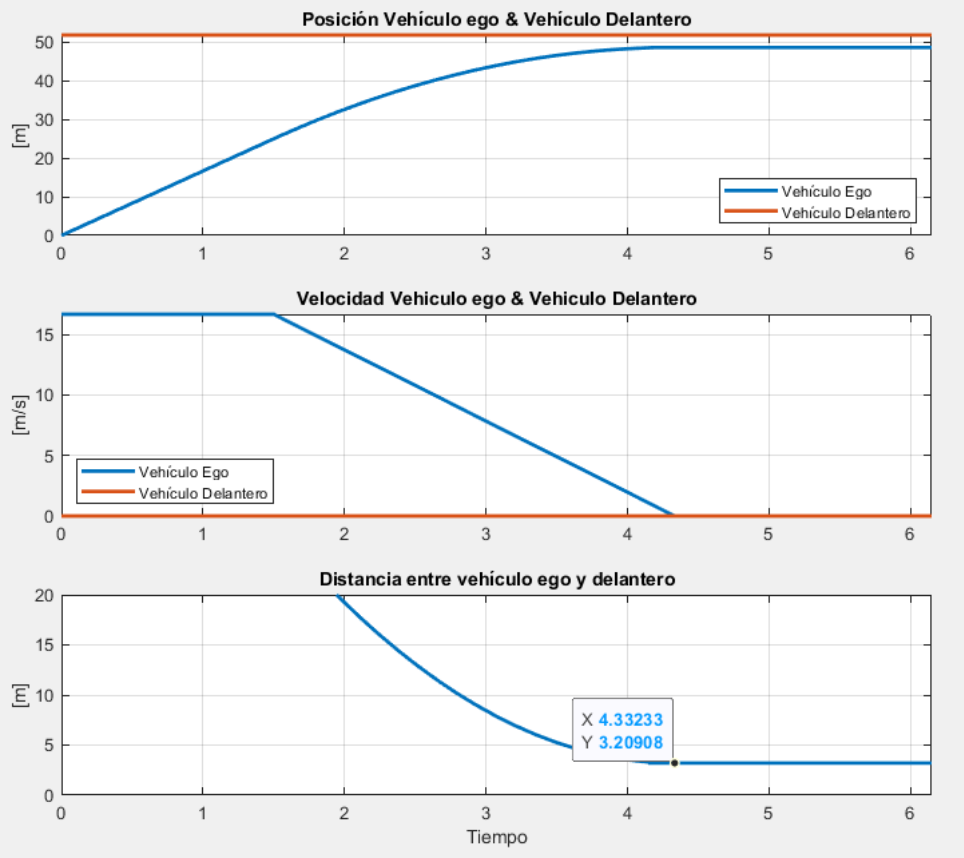



**Ilustración 3.27. Comprobación de colisión con reacción estándar, Escenario 1, Caso 1**

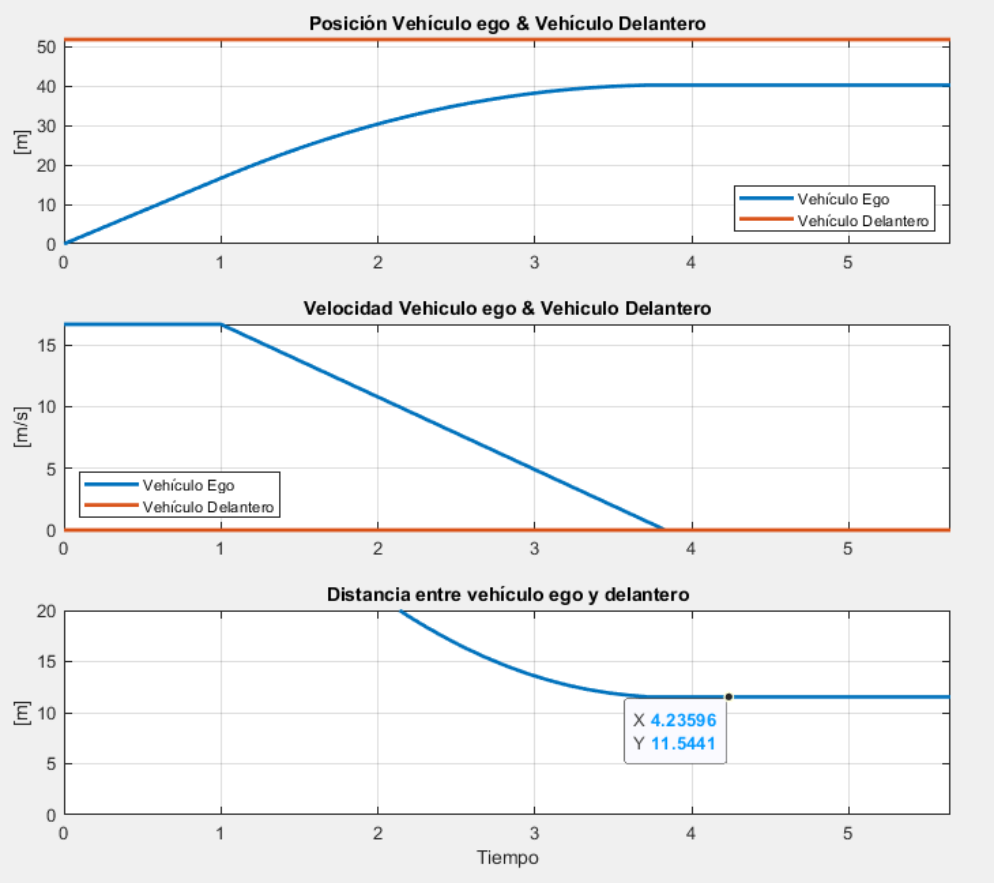



**Ilustración 3.28. Comprobación de colisión con reacción temprana, Escenario 1, Caso 1**

En ambos escenarios se observa claramente como la colisión pudo ser efectivamente prevenida, con una distancia final entre ambos vehículos de 3.2 metros en caso de que el conductor reaccione de acuerdo a lo estimado por el algoritmo, y de 11.54 metros en caso de reaccionar medio segundo antes.

### 3.3.2 Simulación Escenario 1, Caso 2: Vehículo Detenido

Recordando los parámetros de este escenario, se tiene un vehículo delantero totalmente inmóvil, y el vehículo ego dirigiéndose a este a velocidad constante. El caso 1 trata del uso del Lidar de larga distancia, a una velocidad media de 110km/h. La trayectoria de colisión se observa en la siguiente gráfica.

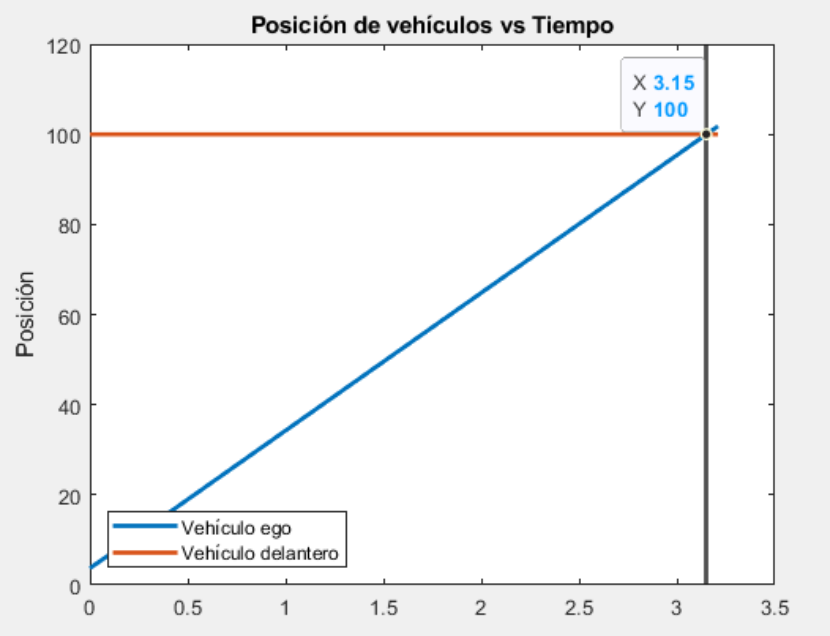



**Ilustración 3.29. Gráfica de posición de Escenario 1, Caso 2**

Tal y como está diseñado el escenario, el vehículo se acerca a gran velocidad al vehículo delantero, recorriendo los 100 metros de rango del Lidar en tan solo 3.15 segundos, momento en el cual ocurre la colisión. Los datos de la simulación se importan al algoritmo FCW de Python, y se obtienen los siguientes valores con respecto a la activación de alertas.

**Tabla 3.6. Tiempos de generación de alertas, Escenario 1, Caso 2**

| | Alerta Leve [s] | Alerta Media [s] | Alerta Máxima [s] | Tiempo entre Al. Max. y colisión [s] |
|---|---|---|---|---|
| Real | 0 | 0 | 0 | 2.9 |
| Sistema | +0.25 | +0.25 | +0.25 | |

Descontando el tiempo de reacción estándar de 1.5 segundos, a la velocidad de 110 km/h sobre la cual fue construido este caso, solo queda 1.4 segundos para frenar o tomar, lo cual no es suficiente para evitar colisión y se demuestra en las gráficas siguientes.

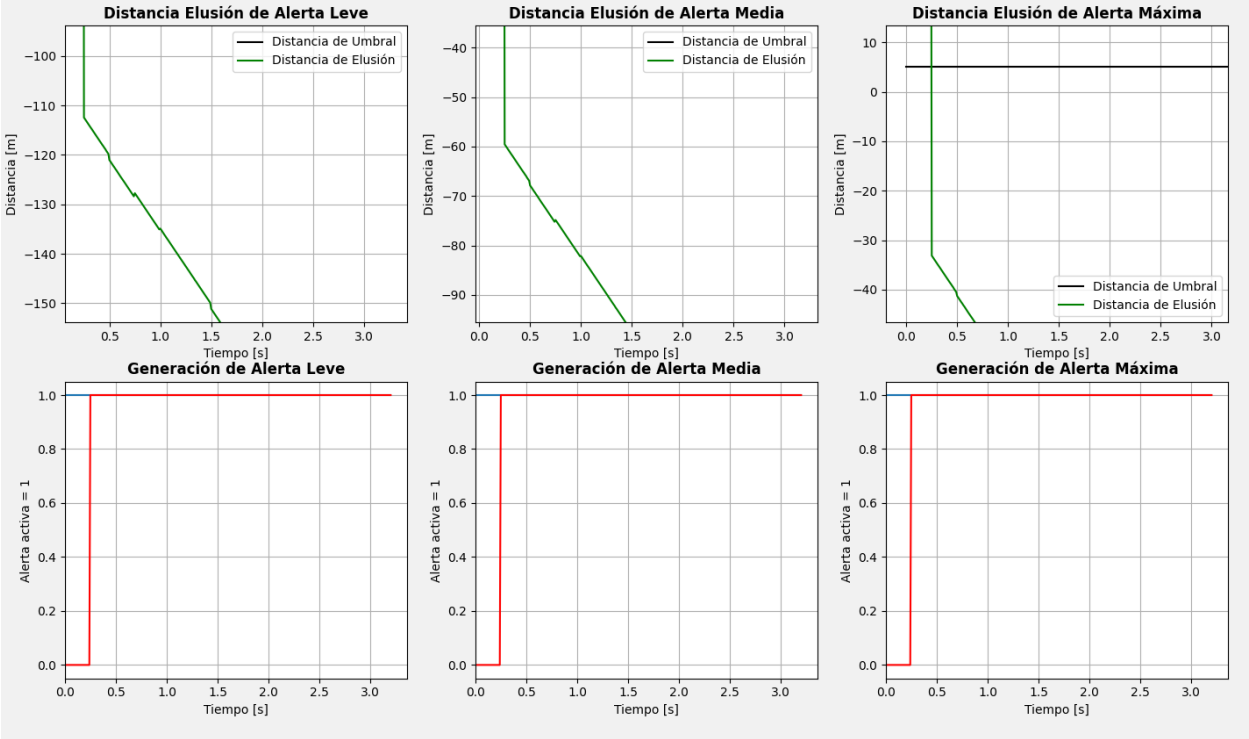



**Ilustración 3.30. Gráficas de alertas Escenario 2, Caso 2**

La ilustración 3.32 demuestra que a velocidades muy por encima del límite, se detectará la colisión sólo dentro del rango máximo del Lidar utilizado, por lo que en estos casos las alertas se activan teóricamente al instante, y en la práctica a los 250ms de detectar el vehículo enfrente (que es el tiempo de respuesta máximo del sistema), pero las distancias de elusión muestran que no es posible evitar la colisión con un tiempo de reacción de 1.5 segundos y desaceleración de 0.6g.

En este caso, se comprobará la prevención de colisión de acuerdo a dos posibilidades:

1) El conductor reacciona acorde a lo sugerido en el algoritmo: Tiempo de reacción de 1.5 segundos y frenado máximo de 0.6g.
2) El vehículo tiene control sobre los frenos: Tiempo de reacción de 0.25 segundos y frenado máximo de 0.6g.

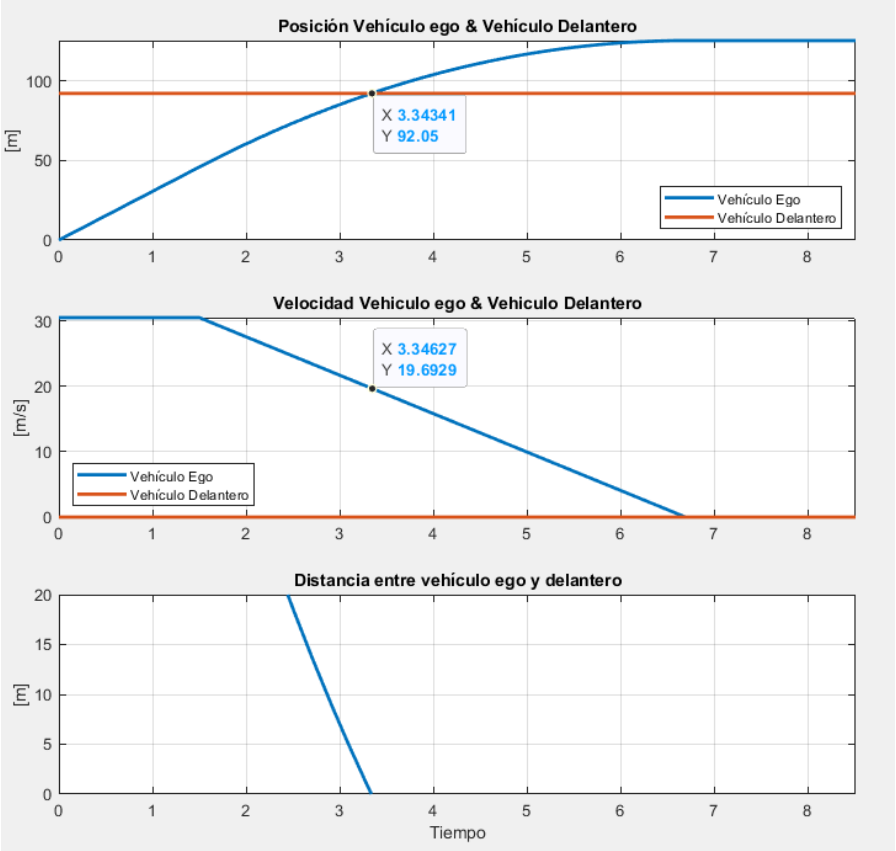



**Ilustración 3.31. Comprobación de colisión con reacción estándar, Escenario 1, Caso 2**

En la ilustración 3.23 se observa que la colisión se da a cabo 3.34 segundos luego de haberse emitido la alerta, y la velocidad de impacto disminuyo de 30m/s a 19.7m/s. Esto representa “el peor de los casos”, ya que en una situación así tienen que coincidir que, el conductor está distraído, conduce a una velocidad muy por encima de la capacidad del sistema FCW, y el vehículo en frente está completamente detenido en una carretera.

Sin embargo, analizando la posibilidad hipotética 2, en la cual el sistema está directamente conectado al frenado eléctrico del vehículo, se obtienen los siguientes resultados.

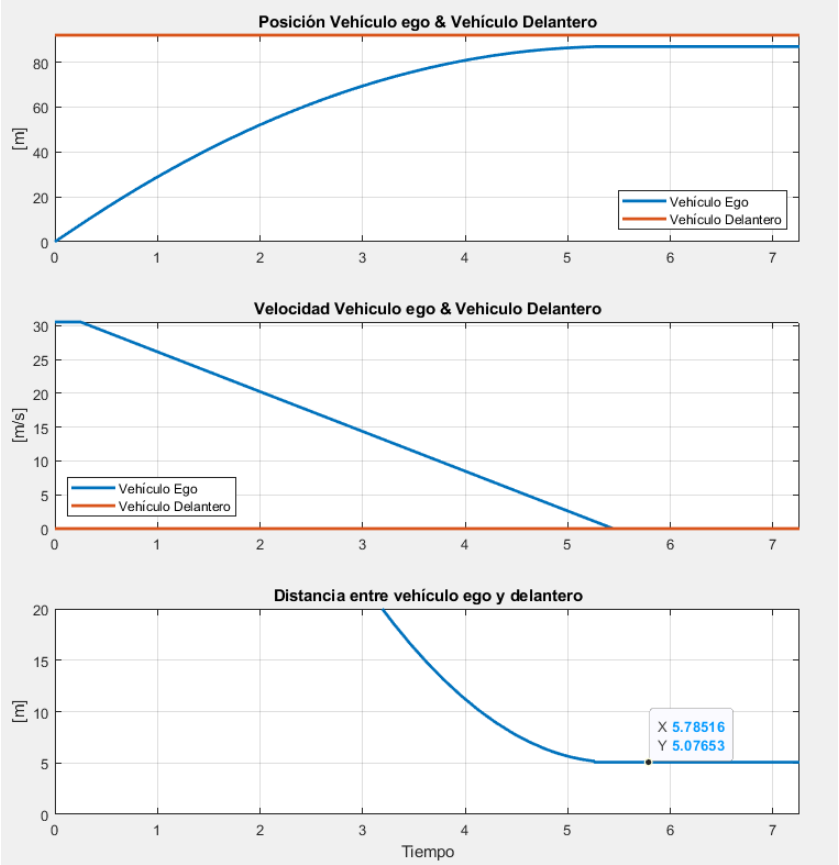



**Ilustración 3.32. Comprobación de colisión con reacción del sistema, Escenario 1, C2**

Sin intervención humana, la rapidez del sistema es capaz de prevenir una colisión, simplemente reaccionando 1.25 segundos antes que el conductor. Claro, esto no podría suceder ya que el sistema actual está diseñado para alertar al conductor, mas no controlar el vehículo. Aún así, esto demuestra las capacidades del sistema y su efectividad por sí solo en la prevención de colisiones.

### 3.3.3 Simulación Escenario 2: Vehículo a menor velocidad

Recordando los parámetros de este escenario, se tiene un vehículo delantero viajando a 30 km/h menos que el vehículo ego, el cual viaja a 90km/h. Ambos tienen un rango inicial de 30m. Al simular este escenario se observa una colisión a los 3.8 segundos, y se infiere la diferencia de velocidades de ambos vehículos de acuerdo al ángulo de sus pendientes en la siguiente gráfica.

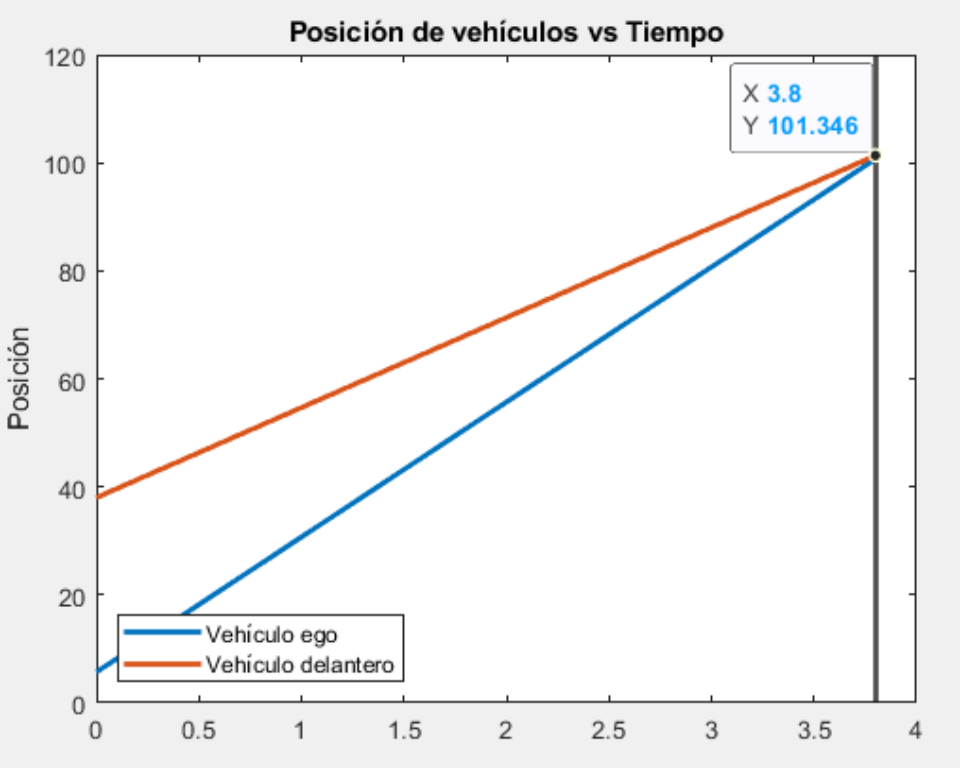



**Ilustración 3.33. Gráfica de posición de Escenario 2**

Los datos de la simulación se importan al algoritmo FCW de Python, y se obtienen los siguientes valores con respecto a la activación de alertas.

**Tabla 3.7. Tiempos de generación de alertas para escenario 2**

| | Alerta Leve [s] | Alerta Media [s] | Alerta Máxima [s] | Tiempo entre Al. Max. y colisión [s] |
|---|---|---|---|---|
| Real | 0.08 | 0.13 | 0.15 | 3.55 |
| Sistema | +0.17 | +0.12 | +0.10 | |

Las alertas se generan en un tiempo muy cercano ya que en estos tipos de escenarios donde las variables cambian constantemente, una vez que se detecta un peligro de colisión solo la desaceleración máxima es la requerida para evitarla.

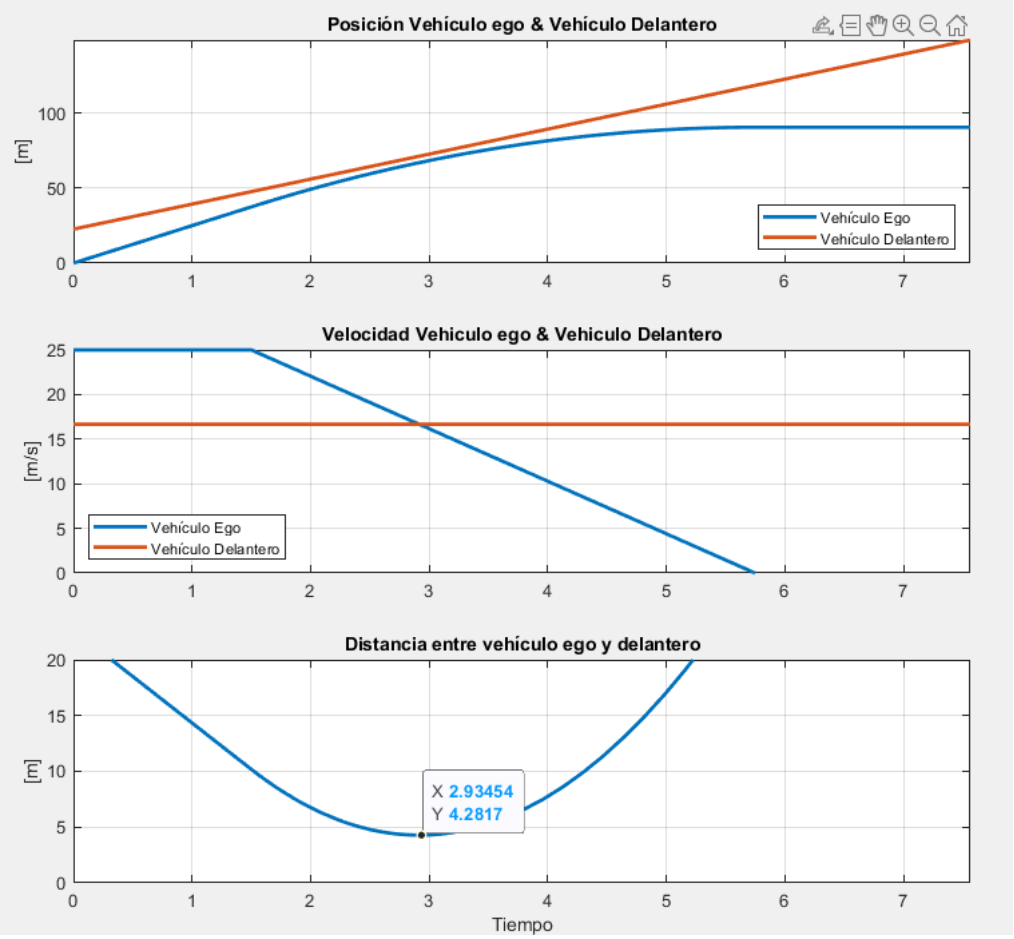



**Ilustración 3.34. Comprobación de colisión con reacción del conductor estándar.**

El algoritmo de Matlab que predice la posición final del vehículo ego termina de generar la gráfica una vez que la velocidad de este sea cero. Sin embargo, realísticamente una vez que se igualen las velocidades, el peligro de colisión será nulo, incluso el propio algoritmo está diseñado para dejar de generar alerta cuando esto suceda. Entonces lo sucedido es que luego de reaccionar por 1.5 segundos, el conductor tiene la justa cantidad de tiempo para disminuir la velocidad de 90 a 60km/h, siempre y cuando frene acorde a lo calculado. Luego de llegar a 60km/h con una distancia al vehículo delantero de 4.3 metros, podrá dejar de frenar y seguir a velocidad constante.

### 3.3.4 Simulación Escenario 3, Caso 2: Frenado repentino

El caso 1 de este escenario ya fue simulado y comprobado al mismo tiempo que se realizaban las pruebas de tasa de refresco para las variables de velocidad y aceleración relativa. La colisión se pudo prevenir acorde al tiempo de reacción y frenado estándar del conductor, calculado por el algoritmo. El caso 2 emplea el mismo frenado abrupto del conductor delantero, pero con la diferencia que esta vez ambos vehículos se encuentran a una menor distancia, aproximadamente 30 metros entre sí.

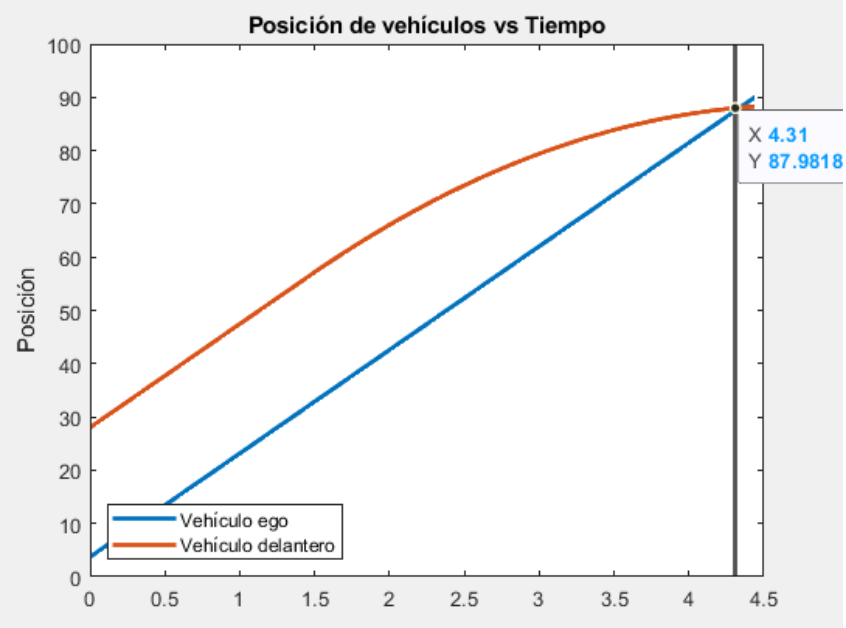



**Ilustración 3.35. Gráfica de posición para Escenario 3, Caso 2**

Ocurre la colisión a los 4.31 segundos de iniciada la simulación. Los datos de la simulación se importan al algoritmo FCW de Python, y se obtienen los siguientes valores con respecto a la activación de alertas.

**Tabla 3.8. Tiempos de generación de alertas para Escenario 3, Caso 2**

| | Alerta Leve [s] | Alerta Media [s] | Alerta Máxima [s] | Tiempo entre Al. Max. y colisión [s] |
|---|---|---|---|---|
| Real | 1.46 | 1.46 | 1.46 | 2.56 |
| Sistema | +0.29 | +0.29 | +0.29 | |

La gráfica de prevención de colisión muestra que tan solo una diferencia de 10 metros y con los mismos valores cinemáticos que el caso anterior, esta vez no fue posible prevenir la colisión con un tiempo de reacción y frenado estándar.

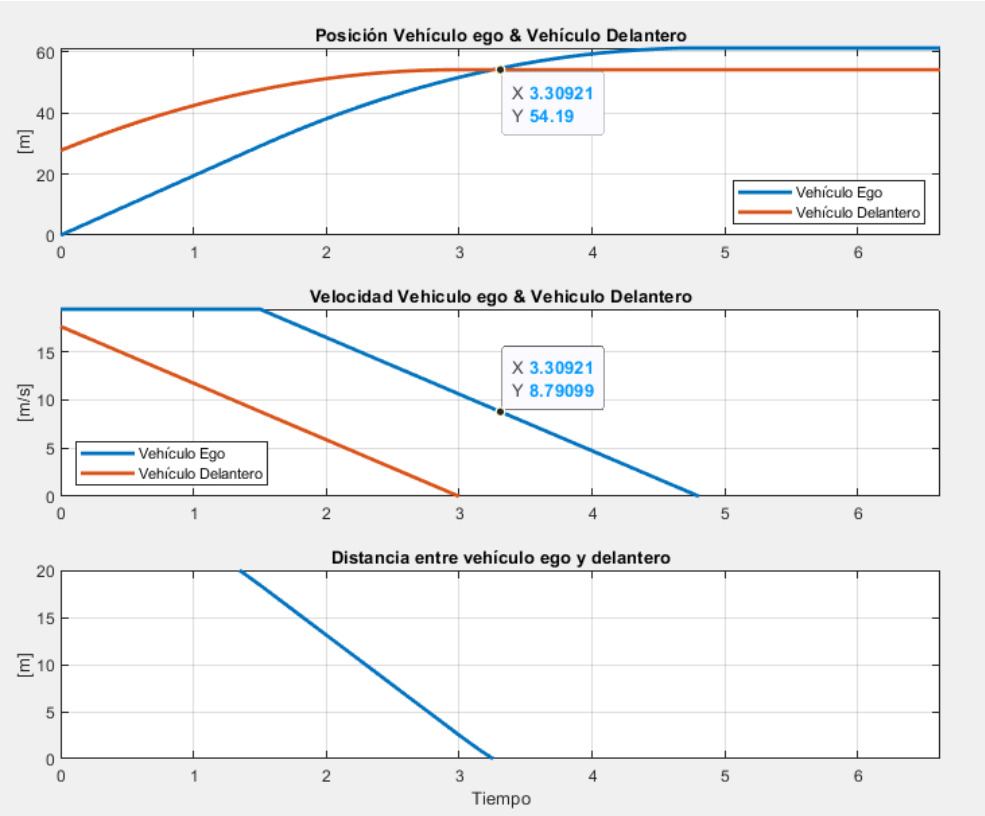



**Ilustración 3.36. Comprobación de colisión Escenario 3, Caso 2**

La colisión se produce 3.3 segundos luego de haberse activado la alerta, a una velocidad de 8.8 m/s. Es decir, que substrayendo el tiempo de reacción estándar del conductor de 1.5 segundos, le sobran 1.8 segundos para realizar maniobras evasivas para intentar prevenir la colisión inminente.

### 3.3.5 Simulación Escenario 4: Cruce espontáneo

Este escenario fue creado para comprobar la reacción del sistema ante el cruce espontáneo de un vehículo delante de su campo de visión. Esto es debido a que aquel cambio de distancia repentino significaría un cálculo de velocidad y aceleración alto por

un periodo de tiempo, por lo que en estos casos se desea implementar una condición al código tal que no surjan alertas falsas.

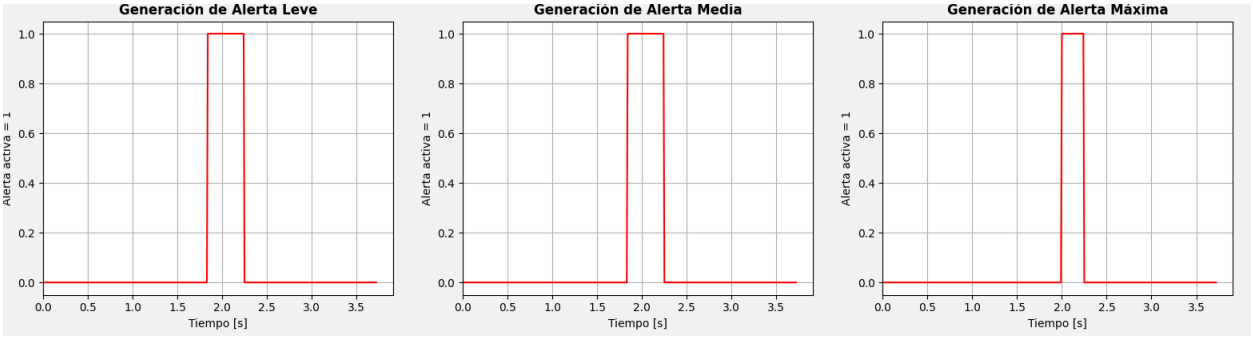



**Ilustración 3.37. Gráfica de generación de alertas para Escenario 4.**

Se activan las 2 primeras alertas por el simple hecho de detectar una disminución brusca en la distancia hacia el vehículo delantero, mientras que la alerta máxima se activa ya que en el primer instante que aparece el vehículo, se detecta el capó frontal, mientras que una vez que se cambio totalmente de carril, se detecta la parte trasera.

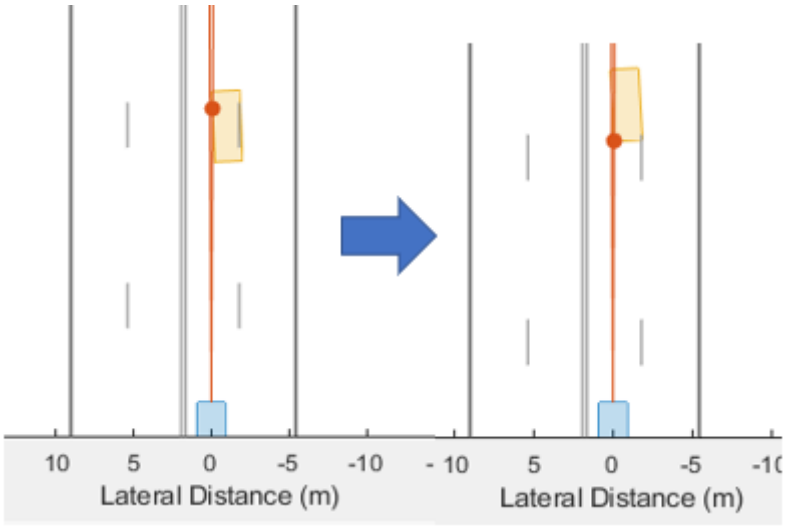



**Ilustración 3.38. Captura de momento de detección en cambio de carril.**

Por esta razón, se propone la inclusión en el código de una condicional tal que, si la distancia detectada por el sensor Lidar cambia bruscamente, se tenga una pausa en la generación de alertas los cálculos para evitar falsos positivos, de al menos 0.25 segundos para la alerta máxima, y 0.5 segundos para la mediana y leve.

### 3.3.6 Simulación Escenario 5: Conducción cercana

En este escenario, ambos vehículos siempre mantienen la misma velocidad, y por ende, la misma distancia entre sí, entonces tanto la velocidad como aceleración relativa son siempre cero a lo largo de la simulación. Se realizó un acercamiento a la gráfica de distancia para diferenciar las tres magnitudes de ruido sobre la señal.

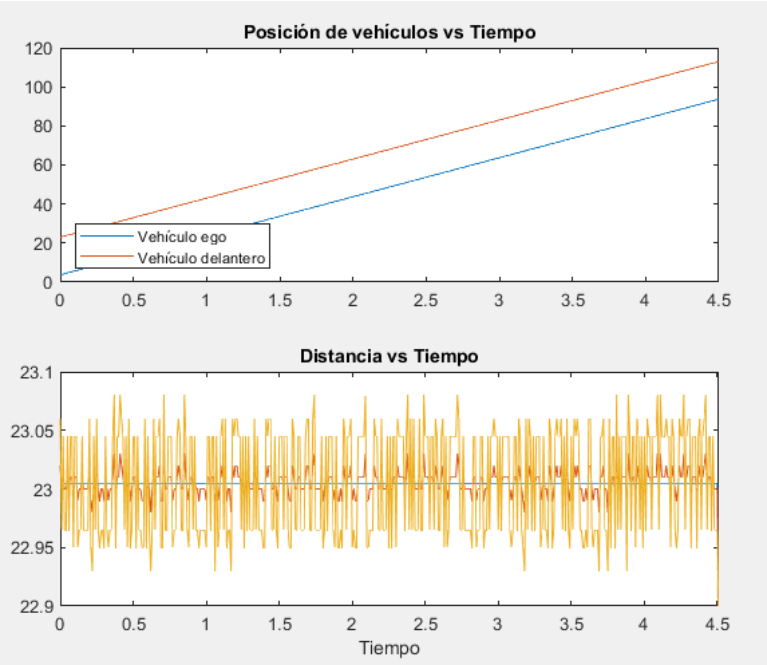



**Ilustración 3.39. Gráficas de Posición y Distancia vs Tiempo de Escenario 5**

Al ingresar los datos en el algoritmo FCW con ruido normal, no se obtiene ninguna alerta a lo largo de todo el tiempo que dura la simulación. Esto demuestra que el error latente en la velocidad y aceleración relativa no es suficiente para generar alertas falsas.

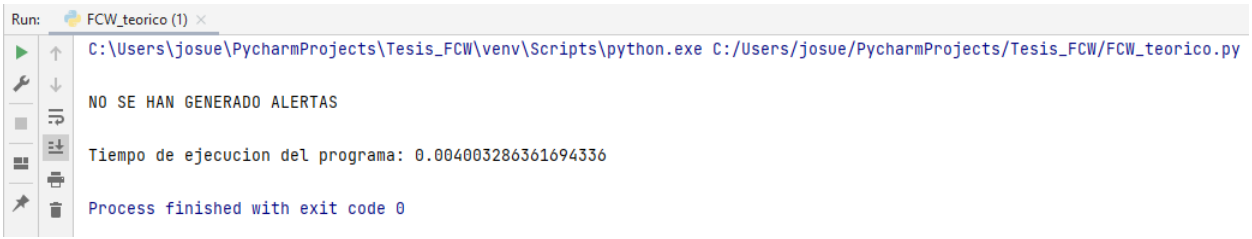



**Ilustración 3.40. Salida de alertas algoritmo FCW con ruido normal para Escenario 5**

## 3.4 Análisis de costos

Ya que el sistema actual está pensado para implementarse en base a dos modelos distintos de Lidar, existirán dos precios para la compra de partes y dispositivos que se detallan en la tabla a continuación. El primer modelo contará con el Lidar TF02-Pro, el cual se recomienda usar en el área urbana, mientras que el segundo modelo contará con el más costoso Lidar TF03, el cual se recomienda usar tanto para ciudad como para autopistas y carreteras de alta velocidad.

**Tabla 3.9. Costos de adquisición de productos, Modelo 1 sistema FCW**

| Producto | Modelo | Precio |
|---|---|---|
| Minicomputador | Raspberry Pi 3B | $35,00 |
| LIDAR | Benewake TF02-Pro 40m | $87,90 |
| GPS | U-Blox Neo-6M | $11,50 |
| Acelerómetro | MPU-6050 | $2,25 |
| Pantalla TFT LCD | ST7735 | $14,50 |
| Fuente Alimentación | Uctronics U6223 | $14,99 |
| Accesorios | Varios | $20,00 |
| | **TOTAL** | $186,14 |

El segundo modelo presenta exactamente los mismos sensores y dispositivos, con la única diferencia presentada en el cambio del sensor Lidar por un modelo de mayor rango.

**Tabla 3.10. Costos de adquisición de productos, Modelo 2 sistema FCW**

| Producto | Modelo | Precio |
|---|---|---|
| LIDAR | Benewake TF03 100m | $260,00 |
| Productos Modelo 1 | | $98,24 |
| | **TOTAL** | $358,24 |

# CAPÍTULO 4

## 4. CONCLUSIONES Y RECOMENDACIONES

Mediante la búsqueda de información, desarrollo y simulación del sistema realizado en este proyecto, se pudo concluir satisfactoriamente los objetivos propuestos, por lo que me dispongo a presentar las siguientes conclusiones.

### 4.1 Conclusiones

- La placa de ordenador Raspberry Pi contiene todo lo necesario para integrar tanto hardware y software en un solo sistema físico que trabaja en conjunto para cumplir con las expectativas de rendimiento y fiabilidad. Este minicomputador contiene el exacto número de puertos sobre los cuales se conectan cada uno de los sensores y módulos, así como todos los protocolos de comunicación necesarios para el funcionamiento y obtención de datos. Asimismo, presenta el suficiente nivel de procesamiento para los procesos y algoritmos empleados, tal que su propio tiempo de reacción representa un valor despreciable para el sistema en sí.

- La programación del algoritmo de prevención de colisiones de tipo alerta de colisión frontal (FCW) presenta distintas ecuaciones teóricas que calculan con total exactitud los tiempos adecuados, según la recomendación del autor de dicho algoritmo, la efectiva generación de alertas de colisión. Aun así, su rendimiento está limitado a la adquisición de datos, que a su vez está limitada por las características y parámetros de los sensores elegidos para el sistema.

- La programación del código para la adquisición de datos presentó ciertos desafíos al momento de procesar señales de sensores con ruido, en específico el sensor de distancia Lidar, que fueron efectivamente combatidas mediante un número de pruebas concretas. Sobre estas

pruebas se logró encontrar la manera óptima de convertir estos datos a variables cinemáticas, tal que se logró maximizar lo mayor posible el tiempo de respuesta del sistema, al mismo tiempo que dichas variables obtenidas sean lo suficientemente precisas para generar alertas efectivamente y sin falsos positivos en ninguno de los escenarios de conducción.

- Los escenarios de simulación creados para la simulación del sistema demostraron la efectividad de este de acuerdo a diversos casos de colisiones posibles que se podrían dar a cabo en la vida real. Asimismo, se demostró que en todos los escenarios donde había una distancia segura entre ambos vehículos, o la velocidad medida se encontraba dentro del rango de operación, la colisión se pudo prevenir. En ciertos escenarios donde la distancia entre vehículos era muy corta, o la velocidad era superior al rango de operación, la colisión no se pudo evitar. De todos modos, sí se observó una disminución significativa de velocidad que implica menor gravedad del accidente. Aun así, lo anterior no le quita mérito a la efectividad del sistema, ya este mismo también alerta al conductor al sobrepasar la distancia no segura, con lo que sobre éste cae la responsabilidad.

## 4.2 Recomendaciones

- A pesar de que el sistema fue probado de acuerdo a los parámetros encontrados en la ficha de datos de cada uno de los sensores, es importante adicionar una serie pruebas intensivas para cada uno, debido a que el comportamiento real de un dispositivo puede diferir debido a múltiples factores externos fuera del control del usuario.

- El sistema físico ha sido diseñado tal que depende en gran cantidad de su correcto posicionamiento en el panel frontal del vehículo donde se lo piensa instalar, de modo que se requeriría incluir el diseño físico de una carcasa para todos los dispositivos del sistema, que sea estable, se adhiera

fuertemente a la superficie y que pueda ser nivelada con mucha precisión, tal que permanezca paralela con el suelo.

- Es importante en la fase de implementación, realizar pruebas primero en un simulador real de conducción, tal que se pueda calibrar con mucha más precisión los instantes en que se deba de generar una alerta. Esto es debido a que a pesar de que una alerta generada por el sistema sea válida, no se puede saber con exactitud si el conductor la entienda como necesaria en conducción normal, puesto a que si este se encuentra a plena concentración, su tiempo de reacción será significativamente menor al de una persona distraída. Por ese motivo, se requiere aportes de personas de prueba para mejorar todavía más este sistema.

## 4.3 Limitaciones

- Aunque este sistema fue dimensionado desde su inicio solo para funcionar con vehículos pequeños y medianos, en la industria automovilística existen vehículos deportivos, los cuales presentan una altura muy por debajo de lo estándar, y muy probablemente no puedan ser detectados por el Lidar al estar este colocado en el panel frontal de un vehículo de mayor tamaño. Sin embargo, esto no fue considerado debido a la casi escasa cantidad de estos tipos de vehículos en el mercado local.

- En curvas el sensor de distancia no podrá detectar al vehículo en frente, ya que este naturalmente se encontrará en un ángulo mayor al del campo de visión del Lidar. También, en pendientes, el valor de aceleración del vehículo ego no será considerada, ya que el valor de gravedad estaría afectando considerablemente la medición del eje paralelo a la calle sobre el cual se adquieren los datos.

- El sistema presenta en todo momento un ligero retraso máximo de 250ms, el cual podría duplicarse en casos de cambios bruscos de distancia, es decir, en cambios de carril del vehículo ego o de algún vehículo delantero.

# BIBLIOGRAFÍA

J. Jansson, “Collision avoidance theory with application to automotive collision mitigation”, Tesis de Maestría, Dept. Ing. Elect., Universidad de Linköping, Linköping, Suecia, 2005.

L. Danielsson, “Tracking and radar sensor modeling or automotive safety systems”, Tesis de Doctorado, Dept. Signals and Systems, Chalmers University of Technology, Göteborg, Suecia, 2010.

Mobility and transport - European Commission. (2016, Octubre 17), “Collision avoidance systems”. [Online]. Disponible en: https://ec.europa.eu/transport/road_safety/specialist/knowledge/esave/esafety_measures_unknown_safety_effects/collision_avoidance_systems_en

F. Derks, (2018, Mayo 23), “How do collision avoidance systems work?”, OrCAD. [Online]. Disponible en: https://www.orcad.com/cn/node/6581

El Universo, (2021, marzo 16). “Cuáles son las marcas y modelos de vehículos más vendidos en lo que va de 2021 en Ecuador”. [Online]. Disponible en: https://www.eluniverso.com/entretenimiento/motores/cuales-son-las-marcas-y-modelos-de-vehiculos-mas-vendidos-en-lo-que-va-de-2021-en-ecuador-nota/

NHTSA, “Automotive Collision Avoidance Systems (ACAS) Program”, U.S. Department of Transportation, Springfield, VA, DOT HS 809 080, Agosto 2000

N. Venkateswaran, W. J. Hans, and N. Padmapriya, “Deep learning based robust forward collision warning system with range prediction,” Multimedia Tools and Applications, vol. 80, no. 14, pp. 20849–20867, 2021

N. Augustsson, M. Wilhelmsson, “Tracking and radar sensor modeling or automotive safety systems”, Tesis de Doctorado, Dept. Signals and Systems, Chalmers University of Technology, Göteborg, Suecia, 2010.

Y. J. Zhang, F. Du, J. Wang, L. S. Ke, M. Wang, Y. Hu, M. Yu, G. H. Li, and A. Y. Zhan, “A safety collision avoidance algorithm based on comprehensive characteristics,” Complexity, vol. 2020, pp. 1–13, 2020.

Hyden C., 1996. Traffic conflicts technique: state of the art. In: Topp, H.H. (Ed.), Traffic safety work with video processing. University Kaiserslautern, Transportation Department, 1996, Green Series No. 37, pp. 3–14

K. Lee and H. Peng, “Evaluation of automotive forward collision warning and collision avoidance algorithms,” Vehicle System Dynamics, vol. 43, no. 10, pp. 735–751, 2005.

A. Burgett, A. Carter, G. Preziotti, “An algorithm for rear-end collision avoidance warning systems”, National Highway Transportation Safety Administration & The John Hopkins University Applied Physics Laboratory, 2000, pp. 310.

M. Skolnik. Introduction To Radar Systems. McGraw-Hill, 1983. ISBN 0-07-066572-9.

Y. Fang, X. Chen, "Design and Simulation of UART Serial Communication Module Based on VHDL," 2011 3rd International Workshop on Intelligent Systems and Applications, 2011, pp. 1-4, doi: 10.1109/ISA.2011.5873448.

Subero, A., 2017. USART, SPI, and I2C: Serial Communication Protocols. Programming PIC Microcontrollers with XC8, pp.209-276.

Sharma, B., (2021, February 10). What is LIDAR Technology and how does it work? Geospatial World. Disponible en: https://www.geospatialworld.net/blogs/what-is-lidar-technology-and-how-does-it-work/

# APÉNDICES

# APÉNDICE A

## Código de la obtención de datos de simulación Matlab

```
%% SIMULACIONES SISTEMA FCW

%drivingScenarioDesigner('C:\Users\josue\Documents\MATLAB R2021a\Esce-
nario.mat')


%% INICIAR SIMULACION
load('E4_SIM')
% Nombre de la variable struct de simulacion: sim1

%Para obtener el tamaño de líneas de datos
n1 = size(struct2table(sim1),1);
%Para obtener un array de los datos de tiempo
tiempo = zeros(n1,1);
for i=1:n1
    t = sim1(i).ObjectDetections{1, 1}.Time;
    tiempo(i,1)=t;
end
tf = tiempo(n1,1);

%% Array de los datos de distancia
distancia = zeros(n1,1);
distancia_ruido1 = zeros(n1,1);
distancia_ruido2 = zeros(n1,1);

for i=1:n1
    x = sim1(i).ObjectDetections{1,1}.Measurement(1);
    distancia(i,1)=x-1.9;
end
%Array de los datos de distancia con ruido por error de exactitud
for i=1:n1
    x = sim1(i).ObjectDetections{1,1}.Measurement(1);
    rn = rand(1);
    rn2 = rand(1);
    % 50% para sumar error
    if rn2>0.5
        %Probabilidad de 68.2% de repetibilidad 1σ
        if rn>0.318
            xr1 = x + 0.005;
            xr2 = x + 0.03;
        %Probabilidad de 27.2% de repetibilidad 2σ
        elseif rn>0.046
            xr1 = x + 0.015;
            xr2 = x + 0.04;
        %Probabilidad de 4.6% de repetibilidad 3σ
        else
            xr1 = x + 0.025;
            xr2 = x + 0.05;
        end
```

```
    % 50% para restar error
    else
        %Probabilidad de 68.2% de repetibilidad 1σ
        if rn>0.318
            xr1 = x - 0.005;
            xr2 = x - 0.03;
        %Probabilidad de 27.2% de repetibilidad 2σ
        elseif rn>0.046
            xr1 = x - 0.015;
            xr2 = x - 0.04;
        %Probabilidad de 4.6% de repetibilidad 3σ
        else
            xr1 = x - 0.025;
            xr2 = x - 0.05;
        end
    end

    distancia(i,1)=x;
    distancia_ruido1(i,1)=xr1;
    distancia_ruido2(i,1)=xr2;
end

%% GRAFICA DISTANCIA VS TIEMPO
figure(1)
subplot(2,1,2)
plot(tiempo(:,1),distancia(:,1))
title("Distancia vs Tiempo");
xlabel("Tiempo");
ylabel("Distancia");
xlim([0 tf]);
hold on
plot(tiempo(:,1),distancia_ruido1(:,1))
plot(tiempo(:,1),distancia_ruido2(:,1))
hold off

%% Para obtener velocidades reales (v, v_rel)
velocidad = zeros(n1,1);
velocidad_relativa = zeros(n1,1);
for i=1:n1
    %Velocidad vehiculo ego
    v1_struct = sim1(i).ActorPoses;
    v1_cell = struct2cell(v1_struct);
    v1_cell2 = v1_cell(3,1);
    v1_array = v1_cell2{1,1};
    v1 = v1_array(1,1);
    velocidad(i,1)= v1;

    v_rel = sim1(i).ObjectDetections{1,1}.Measurement(4);
    velocidad_relativa(i,1)=v_rel;
end

%% Para obtener la aceleración lineal
a = diff(velocidad)/0.01;

%% Para obtener la aceleración relativa
ar = diff(velocidad_relativa)/0.01;

%% Posición de vehiculo 1,2 vs tiempo
pos1 = zeros(n1,1);
pos2 = zeros(n1,1);
for i=1:n1
```

```
%% Posición de vehiculo 1,2 vs tiempo
pos1 = zeros(n1,1);
pos2 = zeros(n1,1);
for i=1:n1
    %Posicion vehiculo anfitrion
    pos1_struct = sim1(i).ActorPoses;
    pos1_cell = struct2cell(pos1_struct);
    pos1_cell2 = pos1_cell(2,1);
    pos1_array = pos1_cell2{1,1};
    p1 = pos1_array(1,1);
    %Compensada la diferencia de coordenadas terreno vs vehiculo
    pos1(i,1)= p1+2.7;

    %Posicion vehiculo delantero
    pos2_struct = sim1(i).ActorPoses;
    pos2_cell = struct2cell(pos2_struct);
    pos2_cell2 = pos2_cell(2,2);
    pos2_array = pos2_cell2{1,1};
    p2 = pos2_array(1,1);
    %Compensada la diferencia de coordenadas terreno vs vehiculo
    pos2(i,1)= p2-2;
end

% Grafica de posiciones vs tiempo
% subplot(2,1,1)
plot(tiempo(:,1),velocidad_relativa(:,1),tiempo(:,1),pos2(:,1),'LineWi
dth',2);
LineX = {'Color','black','LineWidth',2};
xline(4.31, LineX{:});
title("Posición de vehículos vs Tiempo");
ylabel("Posición");
legend({'Vehículo ego','Vehículo delantero'},'Location','southwest')

%% Export to txt file
out_r = distancia(2:n1,:);
out_r = round(out_r,2);

out_v = velocidad(2:n1,:);
out_v = round(out_v,2);

out_vr = velocidad_relativa(2:n1,:);
out_vr = round(out_vr,2);


out_a = round(a,2);

out_ar = round(ar,2);

out_r1 = distancia_ruido1(2:n1,:);
out_r1 = round(out_r1,2);

out_r2 = distancia_ruido2(2:n1,:);
out_r2 = round(out_r2,2);

output = [out_r, out_v, out_vr, out_a, out_ar, out_r1, out_r2];

%% Creacion de archivo de texto

writematrix(output,'E4.txt')
```

# APÉNDICE B

## Código de Algoritmo FCW para la importación de datos de simulación y obtención de valores prácticos

```
import numpy as np
import time
from matplotlib import pyplot as plt
t1 = time.time()

# IMPORTACI?"N DE VARIABLES DESDE MATLAB
var = []
with open("E4.txt") as textFile:
    for line in textFile:
        values = [item.strip() for item in line.split(',')]
        var.append(values)
var = np.array(var)

rango_real_array = var[:, 0]
rango_real_array = [float(item) for item in rango_real_array]

rango_array = var[:, 5]
rango_array = [float(item) for item in rango_array]

vh_array = var[:, 1]
vh_array = [float(item) for item in vh_array]

vr_array = var[:, 2]
vr_array = [float(item) for item in vr_array]

ah_array = var[:, 3]
ah_array = [float(item) for item in ah_array]

ar_array = var[:, 4]
ar_array = [float(item) for item in ar_array]

i1 = len(rango_array)

# PARA EVITAR ERRORES AL INICIO DEL PROGRAMA SE ENCERAN CIERTAS VARIABLES
vr = 0
ah = 0
ar = 0
al = 0
vh = vh_array[0]

# VARIABLES PARA ALGORITMO FCW
alert_high = 0
alert_med = 0
alert_low = 0
gps = True
rain = False
ah_max = -0.6*9.81
intensidad = 1500
```

```
# VARIABLES PARA OBTENCION DE VR y AR
r_lista = []
vr_lista = []

# Listas para graficar variables
rango_graph = [0.0]*i1
v_graph = [0.0]*i1
vr_graph = [0.0]*i1
a_graph = [0.0]*i1
ar_graph = [0.0]*i1
tiempo_array = []

# Listas para graficar alertas
d_tresh_graph = []
d_high_graph = []
d_med_graph = []
d_low_graph = []
tls_graph = []

alert_high_graph = []
alert_med_graph = []
alert_low_graph = []

# VARIABLES PARA LIMITAR SOLO UNA IMPRESION DE RANGO POR ALERTA
printed_h = 0
printed_m = 0
printed_l = 0

# VARIABLES PARA EL BUCLE PRINCIPAL
i = 0
j = 0

# BUCLE PRINCIPAL
while i < i1:
    tiempo = i * 0.01
    tiempo_array.append(tiempo)
    # Distancia 100 Hz
    rango = rango_array[i]
    ah = ah_array[i]
    # Velocidad 1 Hz
    if j == 101:
        vh = vh_array[i]
        j = 0

    # if intensidad > 500:
    #     a,b,dt1,dt2 = 25,1,0.25,0.25
    # else:
    #     a,b,dt1,dt2 = 30,2,0.3,0.6
    # CONVERTIR RANGO A VELOCIDAD RELATIVA
    r_lista.append(rango)
    r_len = len(r_lista)
    if r_len == 26:
        r1 = r_lista[0]
        r2 = r_lista[25]
        vr = (r2 - r1) / 0.25  # Se convierte en 10Hz,0.1s
        vr_lista.append(vr)  # Ingresar el valor vr a su lista
        r_lista.clear()  # Borrar los contenidos de la lista de distancia
        r_lista.append(r2)  # Ultimo valor a la lista nueva
```

```
    # CONVERTIR VELOCIDAD RELATIVA A ACELERACION RELATIVA
    vr_len = len(vr_lista)
    if vr_len == 2:
        vr1 = vr_lista[0]
        vr2 = vr_lista[1]
        ar0 = (vr2 - vr1) / 0.25
        if -1.3 < ar0 < 1.3:
            ar = 0
        else:
            ar = ar0
        vr_lista.clear()
        vr_lista.append(vr2)

    # ALGORITMO NHTSA-JHU/APL
    vl = vr + vh
    al = ah + ar

    if -0.5 <= al <= 0.5:
        tls = 0
    else:
        tls = -vl / al

    tls_graph.append(tls)

    # Distancia treshold
    d_tresh = 2 + vh*0.1
    d_tresh_graph.append(d_tresh)

    # Distancia miss cuando TLS>TR
    def solve_d_miss_1(am):
        ths = 1.5 - (vh + ah * 1.5)/am
        delta_r1 = (vr*1.5)+0.5*(al-ah)*(1.5**2)
        delta_r2 = (vr+1.5*(al-ah))*(tls-1.5) + 0.5*(al-am)*((tls-1.5)**2)
        delta_r3 = (vr+1.5*(al-ah)+(al-am)*(tls-1.5))*(ths-tls)+0.5*(-
am)*((ths-tls)**2)
        d_miss_1 = rango + delta_r1 + delta_r2 + delta_r3
        return d_miss_1

    def solve_d_miss_2(am):
        tm = 1.5 + (vr+1.5*(al-ah))/(am+al)
        delta_r1 = (vr*1.5)+0.5*(al-ah)*(1.5**2)
        delta_r2 = (vr+1.5*(al-ah))*(tm-1.5) + 0.5*(al-am)*((tm-1.5)**2)
        d_miss_2 = rango + delta_r1 + delta_r2
        return d_miss_2


    if tls >= 1.5:
        d_miss_high = solve_d_miss_1(ah_max)
        d_miss_med = solve_d_miss_1(ah_max*0.75)
        d_miss_low = solve_d_miss_1(ah_max*0.5)
    else:
        d_miss_high = solve_d_miss_2(ah_max)
        d_miss_med = solve_d_miss_2(ah_max*0.75)
        d_miss_low = solve_d_miss_2(ah_max*0.5)

    d_high_graph.append(d_miss_high)
    d_med_graph.append(d_miss_med)
```

```
        d_low_graph.append(d_miss_low)

        # ALERTA MAXIMA
        if d_tresh > d_miss_high:
            alert_high = 1
        else:
            alert_high = 0

        # ALERTA MEDIA
        if d_tresh > d_miss_med:
            alert_med = 1
        else:
            alert_med = 0

        # ALERTA BAJA
        if d_tresh > d_miss_low:
            alert_low = 1
        else:
            alert_low = 0

        alert_high_graph.append(alert_high)
        alert_med_graph.append(alert_med)
        alert_low_graph.append(alert_low)

        if alert_high == 1 and printed_h == 0:
            print("Alerta Máxima, ", "Rango:", rango, "Vr:", round(vr, 2),
                  "Ar", round(ar, 2), "Tiempo:", round(tiempo, 2))
            print("DU:", d_tresh, "DE:", round(d_miss_high, 2))
            printed_h = 1
            finish = 1

        if alert_med == 1 and printed_m == 0:
            print("Alerta Media, ", "Rango:", rango, "Vr:", round(vr,2),
                  "Ar", round(ar,2), "Tiempo:", round(tiempo,2))
            print("DU:", d_tresh, "DE:", round(d_miss_med, 2))
            printed_m = 1

        if alert_low == 1 and printed_l == 0:
            print("Alerta Baja, ", "Rango:", rango, "Vr:", round(vr, 2),
                  "Ar", round(ar, 2), "Tiempo:", round(tiempo, 2))
            print("DU:", d_tresh, "DE:", round(d_miss_low, 2))
            printed_l = 1

        rango_graph[i] = rango
        v_graph[i] = vh
        vr_graph[i] = vr
        a_graph[i] = ah
        ar_graph[i] = ar

        i = i+1
        j = j+1

    t2 = time.time()
    total = t2-t1
    print("")
    #print("Tiempo de ejecucion del programa:", total)
    # FINAL DEL PROGRAMA
```

```
# Obtencion de magnitud de error en velocidad y aceleracion relativa
error_r = [abs(a - b) for a,b in zip(rango_real_array,rango_array)]
error_vr = [abs(a - b) for a,b in zip(vr_array,vr_graph)]
error_ar = [abs(a - b) for a,b in zip(ar_array,ar_graph)]
max_error_vr = max(error_vr[1:i1-10])
max_error_ar = max(error_ar[1:i1-10])
max_error_r = max(error_r[1:i1-10])
# print('Error Vr:', round(max_error_vr,2),'Error Ar:',
round(max_error_ar,2))

# GRAFICAS DE DATOS Y VARIABLES
plt.style.use('default')

plt.figure(1, facecolor='#F1F1F1')

plt.subplot(221)
plt.title('Velocidad Relativa vs Tiempo', fontweight="bold")
plt.plot(tiempo_array, vr_array, label='Sin ruido')
plt.plot(tiempo_array, vr_graph, 'r', label='Con ruido estándar')
plt.ylabel('Velocidad Relativa [m/s]')
plt.grid()
plt.legend()
plt.xlim(0, None)

plt.subplot(222)
plt.title('Aceleración Relativa vs Tiempo', fontweight="bold")
plt.plot(tiempo_array, ar_array)
plt.plot(tiempo_array, ar_graph, 'r')
plt.ylabel('Aceleración Relativa [m/s^2]')
plt.grid()
plt.xlim(0, None)

plt.subplot(223)
plt.title('Error Absoluto Velocidad Relativa', fontweight="bold")
plt.plot(tiempo_array, error_vr, 'k')
plt.xlabel('Tiempo [s]')
plt.ylabel('Error Vr [m/s]')
plt.grid()
plt.xlim(0, None)

plt.subplot(224)
plt.title('Error Absoluto Aceleración Relativa', fontweight="bold")
plt.plot(tiempo_array, error_ar, 'k')
plt.xlabel('Tiempo [s]')
plt.ylabel('Error Ar [m/s^2]')
plt.grid()
plt.xlim(0, None)

plt.figure(2, facecolor='#F1F1F1')

plt.subplot(231)
plt.title('Distancia Elusión de Alerta Leve', fontweight="bold")
plt.plot(tiempo_array, d_tresh_graph, 'k', label='Distancia de Umbral')
plt.plot(tiempo_array, d_low_graph, 'g', label='Distancia de Elusión')
plt.ylabel('Distancia [m]')
plt.xlabel('Tiempo [s]')
plt.legend()
plt.grid()
```

```
plt.xlim(0, None)
plt.ylim(-30, 30)

plt.subplot(232)
plt.title('Distancia Elusión de Alerta Media', fontweight="bold")
plt.plot(tiempo_array, d_tresh_graph, 'k', label='Distancia de Umbral')
plt.plot(tiempo_array, d_med_graph, 'g', label='Distancia de Elusión')
plt.ylabel('Distancia [m]')
plt.xlabel('Tiempo [s]')
plt.legend()
plt.grid()
plt.xlim(0, None)
plt.ylim(-30, 30)

plt.subplot(233)
plt.title('Distancia Elusión de Alerta Máxima', fontweight="bold")
plt.plot(tiempo_array, d_tresh_graph, 'k', label='Distancia de Umbral')
plt.plot(tiempo_array, d_high_graph, 'g', label='Distancia de Elusión')
plt.ylabel('Distancia [m]')
plt.xlabel('Tiempo [s]')
plt.legend()
plt.grid()
plt.xlim(0, None)
plt.ylim(-30, 30)

plt.subplot(234)
plt.title('Generación de Alerta Leve', fontweight="bold")
# real_alert_low = [0]*154 + [1]*298
# plt.plot(tiempo_array, real_alert_low)
plt.plot(tiempo_array, alert_low_graph, 'r')
plt.xlabel('Tiempo [s]')
plt.ylabel('Alerta activa = 1')
plt.grid()
plt.xlim(0, None)

plt.subplot(235)
plt.title('Generación de Alerta Media', fontweight="bold")
# real_alert_med = [0]*155 + [1]*297
# plt.plot(tiempo_array, real_alert_med)
plt.plot(tiempo_array, alert_med_graph, 'r')
plt.xlabel('Tiempo [s]')
plt.ylabel('Alerta activa = 1')
plt.grid()
plt.xlim(0, None)

plt.subplot(236)
plt.title('Generación de Alerta Máxima', fontweight="bold")
# real_alert = [0]*155 + [1]*297
# plt.plot(tiempo_array, real_alert)
plt.plot(tiempo_array, alert_high_graph, 'r')
plt.xlabel('Tiempo [s]')
plt.ylabel('Alerta activa = 1')
plt.grid()
plt.xlim(0, None)

print("")
print(i1)
plt.show()
```

```
print('Error Vr:', round(max_error_vr,2),'Error Ar:', round(max_error_ar,2),
max_error_r)
```

# APÉNDICE C

## Código Matlab para la comprobación de colisión post-alerta

```
%%%% COMPROBACION DE DISTANCIA MINIMA POST-ALERTA %%%%
clc
clear
% USO DE DATOS RECOPILADOS DE LA SIMULACI?"N EN PYTHON

%% INGRESO DE DATOS
% Estos se cambian manualmente acorde a los escenarios
tr = 1.5;
r = 27.75;
v1 = 19.44;
v2 = 17.65;
a1 = 0;
a2 = -5.89;
am = -0.6*9.81;
syms t

%% TRAYECTORIA VEHICULO EGO (Host Vehicle)

% Intervalo de tiempo sin reacción del conductor
x1a = (v1*t);

% Intervalo de tiempo con frenado máximo
d1 = v1*(tr); %distancia recorrida en tr seg

xf = ((v1.^2)/(2*(-am)))+ d1;
a = 0.5*am;
b = v1;
c = d1-xf+0.1;
tf1 = (-b+sqrt((b.^2)-4*a*c))/(2*a);
tf2 = (-b-sqrt((b.^2)-4*a*c))/(2*a);

if tf1 > 0
    tf = tf1;
elseif tf2 > 0
    tf = tf2;
end

x1b = d1 + v1*(t-tr) + 0.5*(am)*((t-tr).^2); % Función desplazada tr seg

% Se crea una función por partes
x1 = piecewise(t<tr, x1a, tr < t < tf+tr, x1b, t>tf+tr, xf);


%% TRAYECTORIA VEHICULO DELANTERO (Lead vehicle)
x2a = r + v2*t + 0.5*a2*(t.^2);
x2 = piecewise(t<=2.955, x2a, t>2.955, 54.19);


%% DISTANCIA MÍNIMA ALCANZADA
x3 = x2 - x1;
```

```
%% Velocidad vehículos
vh1 = @(t)v1 + a1*t;
vh2 = vh1(tr) + am*(t-tr);
vh = piecewise(t<=tr,vh1,t>tr, vh2);
vl = v2 + a2*t;


%% GRAFICAS
% Grafica de posición
figure(1)
subplot(3,1,1);
fplot(x1,'LineWidth',2)
xlim([0 tr+tf+2])
title('Posición Vehículo ego & Vehículo Delantero')
hold on
fplot(x2,'LineWidth',2)
legend('Vehículo Ego','Vehículo Delantero','Location','southeast')
hold off
LineX = {'Color','black','LineWidth',1};
ylabel("[m]");

grid on

% Grafica de velocidad
subplot(3,1,2);
fplot(vh,'LineWidth',2)
xlim([0 tr+tf+2])
ylim([0 v1])
hold on
grid on
fplot(vl,'LineWidth',2)
hold off
ylabel("[m/s]");
title('Velocidad Vehiculo ego & Vehiculo Delantero')
legend('Vehículo Ego','Vehículo Delantero','Location','southwest')

% Grafica de distancia mínima
subplot(3,1,3);
fplot(x3,'LineWidth',2)
xlim([0 tr+tf+2])
ylim([0 20])
ylabel("[m]");
xlabel("Tiempo");
title('Distancia entre vehículo ego y delantero')
grid on
```