POLITÉCNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES
UNIVERSIDAD POLITÉCNICA DE MADRID

José Gutiérrez Abascal, 2. 28006 Madrid
Tel.: 91 336 3060
info.industriales@upm.es

www.industriales.upm.es

05 TRABAJO FIN DE MASTER

INDUSTRIALES

Yanni Porteiro

TRABAJO FIN DE MASTER

Lane-Changing Decision-Making Algorithm Based on LIDAR Perception for Autonomous Driving

TRABAJO FIN DE MASTER
PARA LA OBTENCIÓN DEL
TÍTULO DE **MASTER EN INGENIERÍA MECÁNICA**

JUNIO **2022**

Yanni Porteiro

DIRECTOR DEL TRABAJO FIN DE MASTER:

Felipe Jiménez

INDUSTRIALES
ETSII | UPM

POLITÉCNICA

Este Trabajo Fin de Máster se ha depositado en la ETSI Industriales de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Ingeniería Mecánica
Título: Lane-Changing Decision-Making Algorithm Based on LIDAR Perception for Autonomous Driving

Junio 2022

Autor(a): Yanni Porteiro Paraponiaris

Tutor(a):
Felipe Jiménez
Departamento de Ingeniería Mecánica
ETSI Industriales
Universidad Politécnica de Madrid

Resumen

Introducción

Uno de los sistemas de mayor demanda energética e impacto ecológico en nuestra sociedad no agrega ningún valor intrínseco. Según el "2020 Key World Energy Statistics Report", realizado por la Agencia Internacional de la Energía, el sector del transporte es responsable del 36% del consumo mundial de energía, siendo el 21% la fracción estimada para los turismos privados [1] (Figura 1). Por lo tanto, la optimización de esta actividad representa una prioridad dentro de las actividades de desarrollo tecnológico. En el campo de los sistemas de transporte inteligente, los vehículos autónomos individuales juegan un papel importante, ya que son potencialmente independientes de la infraestructura y compatibles con los sistemas de tráfico conducidos por humanos.

Estos vehículos se beneficiarían potencialmente comunicación con vehículos circundantes y centros de gestión de transporte. Esto, combinado, facilita la optimización del movimiento del vehículo. También pueden operar en pelotones de alta velocidad, aumentando de forma significativa la eficiencia operativa, y colaborar para evitar congestiones, aumentando la capacidad de tráfico de las carreteras. Es por lo que considerar una red de dichos vehículos tiene el potencial de aumentar considerablemente la eficiencia de todo el sector del transporte [2]. Por otro lado, la mejora de la seguridad en la conducción es la otra gran justificación de la inversión en el sector.

A pesar de los importantes avances actuales en este campo, la conducción autónoma todavía se enfrenta a ciertos retos tecnológicos, que suponen una barrera para desarrollar todo su potencial. Específicamente, la parte del sistema que se ocupa de la percepción del entorno está muy lejos de la precisión y flexibilidad de la percepción humana. Por lo tanto, es importante no solo desarrollar aún más las tecnologías de percepción, sino también elaborar algoritmos de toma de decisiones que puedan aliviar la dependencia de esta precisión. Este trabajo tiene como objetivo proporcionar un algoritmo de toma de decisiones destinado a las operaciones de cambio de carril, compatible con un sistema de percepción que se base únicamente en sensores LIDAR.

Objetivos y Metodología

El trabajo gira en torno al objetivo principal de crear un algoritmo válido para la toma de decisiones que implicque el cambio de carril, en el ámbito de la conducción automatizada. Bajo ciertas suposiciones y simplificaciones, que se describirán junto con el algoritmo, debería poder analizar el entorno y devolver una señal principal, que es la confirmación para iniciar el cambio de carril. Este objetivo principal implica varios objetivos secundarios, enumerados en orden de importancia:

• Creación de un algoritmo que priorice la seguridad: La principal medición en la que se apoyará la seguridad será la distancia de seguridad, que se calculará de forma continua mediante un modelo dinámico.

• Analizar la eficiencia de la decisión de cambio de carril: el algoritmo debe implicar una mejora en la eficiencia del tráfico, en comparación con un vehículo convencional.

• Hacer que el código sea compatible con la percepción LIDAR: Debe estar diseñado para recibir las entradas de, al menos, un sistema de percepción. En aras de la simplicidad, solo se considerará LiDAR en este caso, siendo conocido como un sistema relativamente robusto para métodos reactivos.

• Creación de entornos válidos para la simulación: durante la etapa inicial del proceso de creación del código, se utilizarán datos generados por simulación. Los entornos de simulación deben elegirse de manera que permitan desarrollar un algoritmo válido para escenarios reales.

• Testeo del algoritmo y calibración de parámetros: Como el programa se basará en varios parámetros preestablecidos, deberá probarse repetidamente para que puedan calibrarse de acuerdo con el análisis de las salidas.

El flujo de trabajo ha seguido el modelo Scrum, normalmente utilizado en el desarrollo de software; un método de desarrollo ágil que se basa en un proceso de iteración continua, donde el resultado gana definición después de cada iteración, también llamado "sprint". La Figura 2 muestra la distribución de sprints; un algoritmo de alto nivel (Sprint 1), un código probado en condiciones ideales (Sprint 2), un código que incluye percepción LIDAR (Sprint 3) y un análisis final del resultado (Sprint 4). Este proyecto se centra en el lado teórico del algoritmo, cubriendo los dos primeros sprints representados en la Figura 2.

Implementación

La lógica expuesta en este trabajo se aborda de forma tradicional, es decir, un algoritmo determinista que produce un comportamiento reactivo, inspirado en un modelo de campos potenciales. Este enfoque representa una simplificación considerable de la carga de trabajo para el desarrollo de este, ajustando la carga de trabajo al alcance de un proyecto de fin de máster.

La alternativa al procesamiento de datos reales, hacia la implementación del algoritmo, ha sido generar los datos del vehículo a través de herramientas de simulación. En este sentido, la aplicación “Driving Scenario Designer” de MATLAB está especialmente concebida para este tipo de aplicaciones. La herramienta permite la creación de una carretera con las dimensiones, la curvatura y el número de carriles deseados. Los agentes se dimensionan según se desee y se introducen en el segmento de carretera, dibujando trayectorias independientes y perfiles de velocidad. En primer lugar, la velocidad se establece como constante, de modo que las trayectorias dibujadas se traducen en una tabla de coordenadas que incluye datos para cada intervalo de tiempo. Después de eso, la velocidad discretizada en cada intervalo puede modificarse en la tabla que se muestra en la Figura 28. Una consideración importante es que el “Driving Scenario Designer” de MATLAB no integra ningún modelo de tráfico, lo que significa que las velocidades deben describirse manualmente.

El proceso de toma de decisiones está inspirado en un modelo típico de control robótico de campos potenciales. Sin embargo, una de las diferencias clave radica en el hecho de que los potenciales calculados resultantes no controlan directamente manejo del vehículo. En cambio, sus valores se usan solo para evaluar la seguridad de un posible cambio de carril. El algoritmo de control y planificación de movimiento, a cargo de realizar el cambio de línea en sí, no forma parte del alcance del proyecto.

En su primera iteración, la lógica del algoritmo se expone en la Figura 32, libre de restricciones vinculadas al lenguaje de programación que finalmente se utilizará. No tiene en cuenta todos los posibles potenciales posibles de vehículos circundantes, representados en la Figura 31, sino solo los considerados esenciales, en un segmento de carretera de dos carriles, cuando el vehículo autónomo, permaneciendo en el carril lento, pretende encontrar un espacio seguro en el rápido. Estos son, el potencial del lado izquierdo, los diagonales izquierdos (delanteros y traseros) y las distancias a los vehículos diagonales, en el sentido de la trayectoria, para garantizar la distancia de seguridad durante y después del cambio.

Inicialmente se comprueba si hay un vehículo delante, en caso afirmativo, el algoritmo evalúa si el vehículo de delante (dentro del rango de visión) tiene una velocidad inferior a la velocidad deseada, que podría ser definida por el usuario o fijada como la velocidad máxima permitida de la vía. Sin embargo, en el caso de que la velocidad de un vehículo delantero varíe muy de cerca la velocidad deseada, este no sería un umbral adecuado desde una perspectiva de control, ya que conduciría a operaciones de cambio de carril ineficaces o innecesarias. Por lo tanto, se establece un tiempo de espera.

Una vez que se ingresa al modo de intención de cambio, diferentes potenciales (que no eran relevantes hasta este punto) comienzan a calcularse y evaluarse secuencialmente siguiendo un orden definido. En primer lugar, se comprueba que no circula ningún vehículo por la derecha en el carril de destino, que en este caso es siempre el de la derecha. Se comprueba si este potencial es inferior a un determinado umbral, que correspondería al cálculo del potencial de proveniente de los bordes de la vía de destino. El siguiente a comprobar es el potencial de la diagonal trasero, encargado de evaluar el riesgo de que los vehículos se acerquen por detrás en el carril rápido. El resultado del cálculo depende lógicamente de la velocidad, así como de la distancia. Grandes distancias entre el vehículo que se aproxima y el autónomo devolverían valores reducidos de potencial, y ocurriría lo contrario con la velocidad. En este caso, el umbral OPdlim debe establecerse empíricamente, dependiendo de la agresividad de conducción o el nivel de seguridad deseados. Asimismo, la distancia al vehículo que se aproxima en la diagonal trasera debe considerarse de forma independiente, ya que su posición con respecto al AV debe respetar una distancia de seguridad después de ejecutar el cambio de carril. Como las velocidades observadas son relativas al AV, un vehículo colocado demasiado cerca para un cambio seguro, con una velocidad alrededor a la del usuario, devolvería un potencial cercano a cero y, de lo contrario, confirmaría la viabilidad del cambio, lo cual no es aceptable. El umbral para esta comprobación de distancia puede ser la tradicional distancia de seguridad, que se establece en la literatura como la longitud de un vehículo estacionario. Es muy recomendable, sin embargo, utilizar la denominada distancia de seguridad “dinámica”, que considera la velocidad de los agentes, ya que a mayor velocidad mayor distancia de frenado. El cálculo y la evaluación del potencial de la diagonal delantera (de los vehículos colocados en el carril izquierdo) se realiza después. La relevancia de esta acción es similar a la anterior, pero se le da menos prioridad en la secuencia de comprobaciones, dado que un vehículo que se aproxima rápidamente desde detrás en el carril rápido es considerablemente más probable que un vehículo en el carril rápido posicionado al frente frenando a velocidades por debajo bajo de la del flujo de tráfico en el carril lento. Una vez comprobadas estas tres ubicaciones potenciales de vehículos en la vía rápida, dando valores superiores a los umbrales en el caso de las distancias e inferiores en el caso de los potenciales, el algoritmo devuelve una señal de confirmación del cambio, que es continua en el tiempo.

Respuesta esperada del algoritmo ante diferentes escenarios:

- Si no hay ningún vehículo posicionado frente al AV, la intención de cambio permanecería inactiva y el bucle de detección continuaría funcionando.
- Si se detecta un vehículo delante y su velocidad es superior a la deseada, la intención de cambio no se activaría y el algoritmo seguirá comprobando continuamente la existencia de dicho vehículo y comparando su velocidad, hasta que esta velocidad disminuya lo suficiente.
- Si hay un vehículo al frente con una velocidad que oscila en torno a la del AV, no se alcanzará el límite de tiempo de espera y nuevamente, la lógica seguirá siendo la verificación de existencia de vehículos y la comparación de velocidades.
- Si un vehículo delantero tiene una velocidad muy baja, esto se reflejará en alcanzar el tiempo de espera, y el AV activará la intención de cambio.

- Si no hay vehículos detectados en el carril rápido, devolverá directamente una confirmación de cambio. Esto no se muestra en la Figura 32, ya que no se verifica explícitamente, sino que está implícito en la forma en que se tratan los datos de los vehículos circundantes en la fase de programación.
- Si se detecta un vehículo en el lateral, en el carril de destino (izquierda), el cambio no se confirmará y la lógica volverá a la verificación inicial del vehículo delantero y al bucle de comparación de velocidades.
- Si no hay un vehículo al costado, sino uno que se aproxima por la parte trasera, el potencial de la diagonal trasera se comparará con el umbral previamente calibrado y la distancia en la dirección de la trayectoria será la última verificada antes de la confirmación del cambio.
- Si hay un vehículo delante, en el carril rápido, se comprobará de forma análoga la seguridad del cambio.
- Si se percibe que alguno de los casos anteriores se dan de forma simultánea, se realizarán las comprobaciones correspondientes por orden de prioridad, tal y como se muestra en el esquema, y la confirmación sólo se dará si se confirman las comprobaciones de seguridad de forma individual (que nunca pueden ocurrir con un vehículo de en el lateral, pero puede ocurrir con un vehículo que se acerca por detrás y otro que frena por delante, si espacio es lo suficientemente grande en términos de distancias y velocidades).

Datos de entrada y consideraciones

El algoritmo debe saber de antemano cuáles son las posiciones de todos los agentes respectivamente a las marcas de la vía. Esta suposición es razonable, conceptualmente, ya que se asume en este caso la existencia de mapas electrónicos instalados en el vehículo que contengan información vectorizada de dichas marcas. Del mismo modo, se supone que un sistema GPS y una unidad de medición interna (IMU) proporcionan la posición y la velocidad precisas del vehículo autónomo en relación con las marcas de los carriles. La percepción a través de LiDAR, junto con el reconocimiento de vehículos y los algoritmos de seguimiento implementados previamente, debería proporcionar una variedad de datos de los agentes detectados dentro del rango del sensor, que incluye:

- Las posiciones del centro de los agentes circundantes o, precisamente, el centro del cuboide que los envuelve, resultante del proceso de reconocimiento y seguimiento de vehículos, respectivamente a la ubicación del sensor LiDAR (dx,dy).

- La velocidad de tales cuboides, que se da como un escalar unidimensional, ya que su velocidad se supone alineada con sus trayectorias, en la dirección del flujo de tráfico.

- Dimensiones de los vehículos (w,l), es decir, la dimensión de los cuboides.

Otra información que debería estar disponible es:

- La velocidad deseada del AV, ya sea como entrada del usuario o como velocidad máxima permitida en la carretera.

- El tiempo máximo de espera, que también se asume como definido por el usuario y un valor constante en la toma de decisiones.

- La posición relativa del sensor LiDAR en el vehículo, es decir, la distancia al frente, atrás, lado izquierdo y derecho de la carrocería (Lf, Lb, Wl, Wr), siendo la altura considerada como no relevante en este caso.

Finalmente, ciertos umbrales para las comprobaciones de seguridad, como OPldbw y OPldfw, deben estar disponibles de antemano.

Sistema de referencia:

El origen del sistema de referencia para la posición de los agentes se encuentra en el sensor LiDAR principal, por lo que las mediciones de las posiciones de todos los agentes y las marcas de línea son relativas al AV. Para el cálculo de los potenciales de las marcas de carril y de los agentes, es necesario catalogar los primeros en función de la zona ocupada de la vía, relativa al origen, definiendo el campo de visión (FoV) en áreas delimitadas. Si se asume un camino recto, estas áreas se pueden establecer utilizando un sistema cartesiano, donde las ubicaciones x e y de los cuboides que contienen a los agentes se comparan directamente con estos límites para la tarea de catalogación. El enfoque se aclara en la Figura 33.

Cálculo de campos potenciales:

El potencial lateral se obtiene como el inverso del cuadrado de la distancia entre la posición del LiDAR y el lado más cercano del cuerpo del agente lateral (lado izquierdo en este caso):

$$OPs_{B_i} = 1/\left(B_{i_y} - \left(\frac{w_i}{2}\right)\right)^2$$

Solo se considera el potencial máximo entre los vehículos del lateral, que se contrasta con un umbral que se establece en función de la posición de la línea de borde del carril izquierdo, dando luz verde a la siguiente comprobación si el potencial del agente es inferior.

$$OPs_{B_i} = 1/\left(B_{i_y} - \left(\frac{w_i}{2}\right)\right)^2 < OPs_{lim} = 1/d^2_{leftboarder}$$

A la comprobación lateral le sigue la comprobación de la diagonal trasera, en la que se identifica que los vehículos pertenecen o no al área de la diagonal trasera: si su posición x está detrás de la parte trasera del AV, si su ubicación y está a la izquierda del lado izquierdo del AV y si al menos el lado interior del cuerpo del agente está más cerca del AV que el borde izquierdo del carril rápido:

$$x_{B_i} < -L_b \;;\; y_{B_i} > W_l \;;\; y_{B_i} - \frac{w_i}{2} < d_{left\ boarder}$$

Los agentes seleccionados obtienen sus potenciales diagonales posteriores calculados, como directamente proporcionales a sus velocidades longitudinales relativas e inversamente proporcionales a la distancia diagonal al cuadrado entre los agentes, calculada utilizando los valores de posición x e y de ambos vehículos como se muestra a continuación. Aunque los potenciales tradicionales se calculan en base a velocidades y distancias en la misma dirección, esta desalineación en el algoritmo propuesto se considera una simplificación admisible:

$$OP_{ld_{Bw}} = \frac{V_{B_i} - V_A}{d^2_{ld_{bw}}} = \frac{V_{B_i} - V_A}{\left(\sqrt{\left(x_A - x_{B_i}\right)^2 + \left(y_{B_i} - y_A\right)^2}\right)^2} = \frac{V_{B_i} - V_A}{\left(x_A - x_{B_i}\right)^2 + \left(y_{B_i} - y_A\right)^2}$$

En el caso de la diagonal frontal, se sigue la misma lógica, con la diferencia de los cambios de signo necesarios para ajustarse a las diferentes zonas:

$$OP_{ld_{Bw}} = \frac{V_A - V_{B_i}}{\left(x_{B_i} - x_A\right)^2 + \left(y_{B_i} - y_A\right)^2} \quad ; \quad \min x_{B_i} > 2 \cdot L_f + L_b + \frac{L_i}{2}$$

Habiendo verificado estos potenciales y distancias, para cada instante de tiempo de simulación, el resultado es un valor de 0 (Falso) o 1 (Verdadero) para la viabilidad de un cambio de carril, como se expone en la Figura 36.

Salidas del algoritmo:

Las salidas del algoritmo, después de haber computado los datos de la simulación, se almacenan en diferentes “array” que cubren la duración de la simulación. De esta manera, al disponer de un valor por cada instante de tiempo, estas salidas pueden representarse como señales continuas, con la resolución temporal lograda durante la generación de datos simulados.

Durante el desarrollo y refinamiento del algoritmo, es importante tener visibilidad del proceso interno de este, razón por la cual las variables clave utilizadas en la lógica también se almacenan como señales continuas, como se muestra en la Figura 37 y la Figura 38. En este caso, los resultados expuestos corresponden a las salidas del programa después de haber procesado un set de datos correspondiente a un escenario básico, el cual se presenta en el capítulo "Data generation through simulation”, utilizado únicamente para validar la lógica y el código. Este conjunto de datos contiene las trayectorias de seis vehículos, incluido el AV, siguiendo un tramo recto de vía a diferentes velocidades, con una duración total de 450 cs (4,5 segundos). En el escenario simulado, una sucesión de tres vehículos adelantará continuamente al AV. Para evitar colisiones (se recuerda que el algoritmo no incluye el control del vehículo), el agente delantero está programado para conducir más rápido que el intermedio, y el trasero más lento. Sin embargo, la velocidad deseada del vehículo ego se preestablece más alta que la de los vehículos delanteros, para permitir que se ejecute la segunda fase del algoritmo.

Asimismo, considerando la carga computacional de la generación del conjunto de datos, se ha fijado a 0 el tiempo de espera del vehículo ego, de forma que se habilite la intención de cambio en cuanto se compruebe, para el primer intervalo de tiempo, que la velocidad del vehículo frontal es menor que la deseada. Por lo tanto, como se muestra en la Figura 38 (a), la señal "Chanintention" se devuelve como un valor continuo "True" (1). Con respecto a la verificación del vehículo lateral, devuelve un valor de 1 cuando el vehículo lateral no está presente. En la simulación, el vehículo lateral adelanta al AV al comienzo de la simulación, lo que se puede observar en el cambio de aproximadamente medio segundo en la señal de salida "SIDECHECK" Figura 38 (b). La verificación de la diagonal frontal, en este caso, como era de esperar, siempre permaneció como "True" (Figura 38 (c)), lo que implica que la verificación de seguridad fue exitosa, ya que el agente tiene una velocidad más alta que el AV y por lo tanto el potencial resultante es negativo, de acuerdo con la definición de potencial diagonal frontal dada anteriormente. Asimismo, siempre se cumplió el control de distancia de seguridad.

El cálculo del potencial de la diagonal frontal (Figura 38 (d)), que es la base para obtener la señal anterior, arroja una curva parabólica de valores negativos crecientes, ya que el vehículo en la diagonal frontal tiene una velocidad positiva constante mayor que la del AV y la distancia entre ellos aumenta con el tiempo. La verificación de la diagonal posterior, por el contrario (Figura 39 (b)), devuelve constantemente una respuesta "False". Esto se considera correcto, ya que el vehículo colocado en la diagonal trasera se acerca al AV con una velocidad relativa positiva. En la Figura 39 (d), por lo tanto, se puede observar cómo el valor del potencial aumenta parabólicamente con la reducción cuadrática de la velocidad. Esto sucede independientemente de la verificación de la distancia de seguridad, ya que el umbral para un potencial diagonal posterior positivo se establece preliminarmente en cero.

El potencial lateral es considerablemente representativo (Figura 39 (c)). Muestra valores variables desde el comienzo de la simulación, hasta aproximadamente 70 cs más adelante, cuando la curva se convierte instantáneamente en cero. Esto representa la fracción de la simulación en la que el vehículo lateral está situado dentro del alcance lateral del AV. La curva se vuelve cero cuando el segundo agente alcanza al autónomo.

Estructura del código de MATLAB

El código del algoritmo, incluido en el Apéndice A, comienza inicializando un conjunto de variables. Se utilizan tres variables para la consideración del tiempo de espera. “t0” es un contador de tiempo simple, “t_clock” recupera el valor de temporal del instante evaluado y “t_wait” es un tiempo de espera máximo definido por el usuario. Todos ellos se inicializan como 0 cada vez que se ejecuta el algoritmo. Como se mencionó anteriormente, "t_wait" se establece en cero para acelerar el proceso de toma de decisiones. "Vdes" también está definido por el usuario, representa la velocidad deseada y es inicializada con un valor de 50m/s, superior al agente más rápido. “SIDE CHECK”, “BACK DIAGONAL CHECK” y “FRONT DIAGONAL CHECK” se definen inicialmente como “falsas”. Asimismo, se crean como un conjunto de ceros los arrays que contendrán la información de los vehículos designados en las distintas zonas, extrayendo el número de agentes, de forma preliminar, del archivo que contiene los datos simulados.

Una vez inicializadas las variables necesarias, el algoritmo en sí se programa dentro de un ciclo "for" que ejecuta la lógica a través del conjunto de datos completo, ejecutándose una vez cada instante de tiempo, hasta que se evalúan todo el tiempo de simulación. Dentro de este bucle, un conjunto de bucles "for" de segundo nivel recorren la información de todos los agentes (el archivo de datos está estructurado conteniendo la información de todos los vehículos dentro de cada instante de tiempo). Dentro de estos bucles for, la lógica del algoritmo de alto nivel, expuesta anteriormente, se aplica mediante un conjunto de operaciones "if/else". Toda la información necesaria para aplicar estas operaciones lógicas, como las posiciones del vehículo, las dimensiones, las velocidades, en relación con el vehículo ego, se obtienen al referirse con precisión a la ubicación exacta en el archivo de datos ("allData"), el cual es generado por MATLAB con una estructura específica. El cálculo de los potenciales se almacena directamente agregando columnas de información en este archivo. En la Figura 39 se muestra una versión muy simplificada de esta sección del código.

Resultados

El algoritmo ha sido probado en un conjunto de escenarios diseñados para representar situaciones típicas de cambio de carril. Algunos de ellos se utilizaron para refinar el código y calibrar algunos de los umbrales potenciales y los dos últimos para comprobar el comportamiento del algoritmo ante situaciones específicas, las cuales no estaban previstas

durante su fase de diseño. Los escenarios de conducción varían en términos de cantidad de agentes, trayectorias y velocidades, pero todos consisten en una sección de carretera recta de 1000 metros. Para reducir el tiempo de cálculo, la frecuencia de muestreo se ha reducido de 100 a 10 Hz. Las variables medidas son las señales de salida frente al tiempo de simulación, medidas en centisegundos (cs) y decisegundos (ds), dependiendo de la resolución de la simulación. La lista de tests realizados es:

- Velocidad variable en vía unidireccional, utilizado para establecer el umbral de potencial diagonal trasero, consiste en AV, colocado permanentemente en el carril lento a una velocidad constante de 80 km/h, siguiendo a un vehículo que le precede con la misma velocidad constante. Ambos coches son adelantados por un único vehículo, situado en el carril rápido, igualmente con una velocidad relativa positiva constante. El vehículo que adelanta circula a una velocidad que varía para cada iteración de la prueba desde 120 km/h de velocidad relativa a 90, 60 y 30, como se observa en las figuras 40 a 43. El potencial de la diagonal trasera se fijó inicialmente dentro del orden de magnitud del correspondiente a la media de las velocidades relativas ensayadas y distancia igual a la distancia dinámica de seguridad. El resultado de promediar este potencial para las diferentes velocidades fue 0,075, que luego se fijó en 0,01 para los potenciales diagonales posteriores y 0,02 para los potenciales diagonales frontales. Esta diferencia proviene de la estimación de que un potencial de diagonal frontal positivo representa una situación de mayor riesgo, dada la reducida visibilidad y maniobrabilidad del agente ejecutando la correspondiente frenada. En la Figura 45 se puede observar que, como era de esperar, velocidades relativas más altas producen potenciales más altos. Los potenciales más altos resultan en una prohibición de cambiar de carril cuando hay mayores espacios entre el agente que adelanta y el AV. Sin embargo, la cantidad de tiempo de prohibición de cambio aumenta de forma considerable a medida que la velocidad relativa disminuye.

- Velocidad variable en vía bidireccional, que en este caso se utiliza para establecer el umbral del potencial diagonal frontal, ya que los vehículos que circulan en sentido contrario se consideran como el peor escenario de potencial generado para dicho tramo de carretera. Salvo el hecho de cambiar el sentido de la vía rápida y las velocidades estudiadas, el escenario se diseña de la misma forma que el caso anterior. Las señales obtenidas son similares, con la única distinción de que el potencial diagonal que se activa es el frontal, en lugar del posterior. Los potenciales obtenidos son considerablemente mayores para las mismas velocidades absolutas del vehículo, ya que la velocidad relativa del vehículo que se aproxima es el resultado de sumar la velocidad absoluta de ambos agentes. Esto se traduce, de nuevo, en mayores distancias entre vehículos al inicio de la prohibición de cambio, pero menores tiempos de espera hasta confirmación de cambio.

- Test de intensidad variable, donde se genera una situación de seguimiento de coches en el carril rápido a valores de intensidad extraídos de un modelo de tráfico lineal de Greenshields. El resultado esperado de la prueba es identificar el nivel de intensidad en el que comienzan a ocurrir las confirmaciones de cambio de carril, definiendo los límites funcionales del algoritmo. Para favorecer la simplicidad, el vehículo autónomo se coloca a velocidad cero. Se utilizó nuevamente la misma sección recta de la carretera y los parámetros de la carretera se calcularon asumiendo una capacidad máxima de 160Veh/h por carril. Aplicando el modelo Greenshields:

$$K_c = 160\frac{veh}{km} \quad ; \quad V_l = 100\frac{km}{h}$$

$$Intensidad\ máxima = i_m = \frac{V_l * K_c}{4} = 4000\frac{veh}{h * lane}$$

$$Densidad\ de\ tráfico\ a\ intensidad\ máxima = \frac{K_c}{2} = 80\ Veh/km$$

$$Velocidad\ a\ intensidad\ máxima = V_m = \frac{i_m}{K_m} = 50\ km/h$$

La curva intensidad-densidad que se muestra en la Figura 51 representa las condiciones de tráfico del escenario considerado, por carril, respondiendo a la ecuación:

$$i = V_l\left(K - \frac{K^2}{K_c}\right)$$

Dentro de la curva, se probaron 5 condiciones diferentes, correspondientes a tres niveles de intensidad: Intensidad máxima, 3000 veh/h y 40 veh/km, 3000 veh/h y 120 veh/km, 2000 veh/h y 24 veh/km y 2000 veh/h y 136 veh/km (Figura 52). Las limitaciones funcionales del algoritmo, que está directamente relacionado con los umbrales de potenciales empleados, se evalúan observando la cantidad de confirmaciones de cambio de carril que se dan. En el cuarto caso, la señal de confirmación de cambio se activa brevemente dentro de los espacios entre vehículos en el carril rápido. Este límite se establece en 24 veh/km, correspondiente a una intensidad de tráfico de approx. 2000 veh/h y una velocidad media de circulación de 85 km/h. Sin embargo, debe tenerse en cuenta que esta es la densidad de tráfico en la que los cambios de carril comienzan a ser plausibles cuando el AV está detenido. A medida que aumenta la velocidad de este, hasta alcanzar las condiciones de velocidad del carril izquierdo (considerando el carril izquierdo como el carril rápido), el potencial diagonal posterior calculado se reduce linealmente y, por lo tanto, los espacios para las confirmaciones de cambio de carril también aumentan linealmente en el tiempo, teniendo como límite mínimo la distancia de seguridad dinámica una vez que se iguale la velocidad de ambos carriles.

- Adelantamiento trasero, teniendo el AV posicionado como arriba, un solo agente se acerca desde atrás a mayor velocidad, moviéndose hacia el carril rápido, adelantando al este y reincorporándose al carril lento. Las trayectorias de los vehículos no se cruzan en ningún punto. La trayectoria del agente que adelanta se define y ajusta manualmente mediante una función de suavizado incluida en la herramienta de MATLAB. Esta sencilla prueba tiene como objetivo revisar el código cuando un agente se desplaza por los diferentes tramos de la vía, generando todos los potenciales considerados (Hasta el momento, solo se registraron potenciales individuales). El potencial diagonal trasero y los potenciales laterales se activan en este caso, variando según la forma de la trayectoria del vehículo que adelanta. La señal de confirmación de cambio de carril se comporta como se esperaba. El potencial diagonal frontal no se activa, ya que el código no lee potenciales negativos (lo cual no es necesario), resultado de velocidades relativas negativas (vehículos que se alejan del AV).

- Congestión de tráfico, donde ambos carriles están cerca de la capacidad máxima de la vía de 160 veh/km. El escenario es similar al de intensidad variable, pero en este caso ambos carriles contienen numerosos agentes y la velocidad del AV no es nula. La intención de la prueba es revisar los resultados de confirmación de cambio cuando el carril ocupado por el AV es ligeramente más lento y rápido que el adyacente, respectivamente. Los escenarios se pueden observar en las figuras 57 y 58. El carril rápido (vehículos naranjas) se ha simulado con una densidad de tráfico de 140 veh/km (velocidad de flujo de 12,5 km/h) y el carril lento una densidad de tráfico de 155 veh/km (velocidad de flujo de 3,12 km/h). El resultado principal de la prueba es observar que, cuando el vehículo ego (azul) se ubica en el carril rápido, se comienzan a obtener potenciales de la diagonal frontal, ya que la velocidad relativa de los vehículos en la diagonal frontal se vuelve positiva. Debido a que el umbral de cambio de carril para la diagonal delantera es considerablemente más grande que el trasero, se reducen drásticamente las posibilidades de que se confirme un cambio de carril en tal situación.

Conclusiones y trabajo futuro

Este trabajo ha concluido con la conceptualización, programación y testeo de un algoritmo determinista de toma de decisiones para el cambio de carril, basado en un modelo reactivo de campos potenciales sensible al movimiento de los agentes circundantes. Se ha considerado que los algoritmos tradicionales y deterministas aún superan a los modelos estocásticos de aprendizaje automático en la solidez de la toma de decisiones. Por lo tanto, el trabajo presentado ha priorizado la seguridad en los cambios de carril, sacrificando la versatilidad de los enfoques basados en inteligencia artificial. Dentro de la lógica utilizada, se ha establecido una cadena de confirmaciones de seguridad, analizando el movimiento de todos los agentes existentes dentro del campo de visión, cada centésima de segundo, para asegurar la viabilidad del cambio de carril. Estas confirmaciones se definen, asimismo, según un modelo que considera no solo la distancia a los agentes circundantes, sino también la velocidad relativa y sus ubicaciones. Los potenciales aumentan cuadráticamente con la velocidad del agente que se aproxima, aumentando su sensibilidad a los usuarios de conducción agresiva y situaciones de cambio de carril con grandes diferencias de velocidad entre carriles.

Como se ha descrito anteriormente, se ha establecido el límite funcional actual del código en la obtención de luz verde de cambio de carril para densidades de tráfico en el carril objetivo de 136 vehículos por kilómetro, circulando a una velocidad máxima de 15 km/h. En estas circunstancias, un agente adelantaría al AV cada 2,2 segundos. Considerando un tiempo de reacción humano, desde la percepción hasta la actuación de los controles del vehículo, de dos segundos, los cambios de carril no serían viables en estas condiciones, salvo en un vehículo autónomo, donde las decisiones se toman cada 1cs.

La percepción LiDAR se ha considerado desde la fase conceptual. El resultado es un algoritmo que asegura la compatibilidad con los datos obtenidos de estos sensores. Generan nubes de puntos que luego se procesan en tiempo real, mediante un software ya establecido, para eliminar datos irrelevantes e identificar y rastrear vehículos, agrupándolos en cuboides. Este software devuelve la velocidad, las dimensiones y la orientación de dichos vehículos. El código se ha escrito recibiendo estas entradas establecidas por la industria, que fueron generadas por una herramienta de MATLAB que también considera sensores LiDAR como la principal fuente de percepción, entregando dichas entradas en el mismo formato. Finalmente, también se ha demostrado que las decisiones que se toman se pueden tomar dentro de un campo de visión de 80 metros, típico de los sensores LiDAR estándar utilizados en el sector de la conducción autónoma.

Finalmente, se creó un conjunto de escenarios de conducción, que representan situaciones realistas para el problema de cambio de carril dentro del entorno considerado para el proyecto. La fase de pruebas permitió el refinamiento del código y el establecimiento de los potenciales para la toma de decisiones. Sin embargo, hay un tanto considerable que se ha delegado para futuros trabajos. Una vez que el algoritmo ha sido capaz de obtener resultados exitosos con datos ideales; el siguiente paso hacia una implementación completa sería generar nubes de puntos correspondientes a los escenarios creados y reconocer y rastrear agentes (vehículos). Esta generación de nubes de puntos corresponderá a la colocación ficticia de sensores LIDAR teóricos en el ego-vehículo simulado, y el algoritmo se ejecutará con la nube de puntos como única entrada, junto con los umbrales de potenciales ya calibrados. Para ello, la herramienta "Driving Scenario Designer" permite generar nubes de puntos virtuales de los diferentes escenarios, una vez especificadas las posiciones y especificaciones de los sensores virtuales.

CÓDIGOS UNESCO:

- 331101 TECNOLOGÍA DE LA AUTOMATIZACIÓN
- 331702 AUTOMÓVILES
- 331710 INGENIERÍA DEL TRÁFICO

PALABRAS CLAVE: Conducción Autónoma, Toma de decisiones, cambio de carril, percepción LiDAR, algoritmo determinista, campos potenciales, modelo reactivo.

Abstract

Currently, transportation systems represent a major cause of non-value-added energy waste, together with the time of its users, which in most cases utilize them daily. It becomes then evident that automating such activity entails considerable advantages at a global scale. During recent years, such automation has started to become possible, mainly thanks to the advances in computer science.

Within automated transportation systems, the focus is primarily on autonomous driving, due to its flexibility for performing in mixed traffic, together with human-driven vehicles. However, this also implies a considerable technical challenge, linked to the inherent variability of human driving. Ensuring safety in this case, without unnecessarily interrupting driving, becomes a priority.

Activities such as lane-keeping and adaptative cruise control have already successfully been achieved by most manufacturers. However, full driving automation cannot happen until handling of complex situations and scenarios get perfectioned, such as urban crossroads or roundabouts. These scenarios are yet far to be solved, being the main limitation vehicle perception.

Nonetheless, a task of relative complexity that is half-way of becoming resolved is lane-changing in simple roads. This work is focused in such area. Specifically, with literature differentiating between perception, location, control and decision-making, the latter is here treated.

A traditional, deterministic decision-making algorithm is proposed, based on a typical potentials reactive model. As opposed to current trend in modern decision-making, based on machine learning techniques for agents' behaviour prediction, deterministic-reactive approaches prioritize simplicity and decision-making robustness. The potentials are speed and distance dependant, calculated relatively to the ego vehicle.

The algorithm receives as an input the positions, speed, and dimensions of all the surrounding agents inside the field of view of a theoretical sensor, giving as an output a viability confirmation for the lane change. The data of all the surrounding vehicles is evaluated every timestamp, locating them in predefined road segments around the AV (front, back, side, back diagonal, and front diagonal). Considering their location, potentials are calculated accordingly, and certain deterministic rules are applied, giving a lane change confirmation if pre-established safety thresholds aren't reached.

Position of the lane markings are assumed to be known, in a straight section of a two-lane highway, thanks to the use of an electronic map, containing vectorized marks, and a GPS and internal measurement unit (IMU) for location. The logic has been conceived considering a purely LiDAR-based perception, towards simplifying a potential future implementation. A formal analysis of LiDAR perception applied to this case, by evaluating the quality of the obtained point clouds, recognizing, and tracking vehicles, and rerunning the code to compare results, has been left for further work.

The result is a MATLAB code that can handle simulation data from MathWorks' Driving Scenario Designer tool, used to generate a set of relevant lane-changing situations. Simple relative speed tests were initially carried out, to set initial values of potential thresholds. With the ego vehicle positioned on the slow lane, having the intention of changing sides, overtaking vehicles were generated in the fast lane at speeds ranging from 30 to 120 km/h. An initial threshold for back-diagonally approaching vehicles was established. Similarly, the direction of the fast lane was switched, with vehicles approaching opposite to the ego vehicle and relative speeds increasing considerably. A different potential limit was therefore established for agents

approaching from the front diagonal, increasing the distances at which lane-changing starts to be negated.

Specific scenarios were also implemented. Traffic jam situations proved that the vehicle is unlikely to obtain a changing confirmation, due to reduced inter-vehicle distances, unless courtesy situations occur. When the speed of the ego vehicle is higher than the destination lane, lane-change confirmations are considerably more unlikely to happen, due to the difference in safety thresholds. A side-effect that in this case is desirable. Likewise, a back overtaking scenario was used to prove the correct vehicle location through the different road segmentations.

Finally, a variable intensity traffic scenario, following a Greenshields's linear model was used, for a lane of 160 veh/km*lane of maximum capacity and 100 KPH of maximum speed. Five different conditions within that model were tested, setting a functional limit for the algorithm of 2000 veh/h; value that improves linearly as the ego-vehicle increases speed, originally stopped.

UNESCO CODES:

- 331101 AUTOMATION TECHNOLOGY
- 331702 AUTOMOVILES
- 331710 TRAFFIC ENGINEERING

KEYWORDS: Autonomous Driving, Decision-making, lane changing, LiDAR perception, deterministic algorithm, potential fields, reactive model.

Acknowledgements

This Master thesis represents the conclusion of a career phase as a student. It is therefore especially important to dedicate this work to my family, for a constant unconditional support, hoping to be able to start returning that favour in the coming years.

A special mention to Juan Carlos, one of the most special people I've met. A friend that I can consider as family and hope will accompany me for a lifetime, for making my time in Madrid extremely pleasant.

Finally, thanks to my thesis mentor, Felipe Jimenez, for giving me a chance within the field of autonomous driving, despite my lack of experience, and for his patience and understanding of personal circumstances.

Table of contents

1 Introduction 20
2 Objectives & Methodology 22
2.1 Project objectives 22
2.2 Project methodology 23
3 Theoretical background 27
3.1 Summary of traffic theory and intelligent transport systems 27
3.1.1 Introduction to traffic theory 27
3.1.2 Macroscopic characterization of the traffic flow 27
3.1.3 Microscopic traffic models 29
3.1.3.1 Free-flow models 30
3.1.3.2 Car-following models 30
3.1.4 Intelligent Transport Systems. Definition and typologies. 32
3.2 Introduction to autonomous vehicles 33
3.2.1 Positioning systems in AV 34
3.2.1.1 Internal vehicle positioning 34
3.2.1.2 Vehicle localization 35
3.2.2 Perception systems in AV 36
3.2.2.1 Vision-based 36
3.2.2.2 LIDAR-based 37
Basic concepts 39
Measuring techniques 39
Imaging systems 40
Technology challenges 40
3.2.3 Decision-making in AV 41
3.2.4 Control systems in AV 44
3.3 State of the art of autonomous lane-changing algorithms 46
3.3.1 Behaviour prediction in lane changing 47
3.3.2 The use of RL in lane changing 49
3.3.3 The use of Game Theory in lane changing 51
3.3.4 Cooperative lane changing 55
3.3.5 Potential fields theory application in AV 58
3.3.5.1 Lateral Planning Potentials 59
3.3.5.2 Longitudinal Planning 60
3.3.6 Overview and conclusion 61
4 Implementation 62
4.1 Data generation through simulation 62

4.1.1 Functioning of the “Driving Scenario Designer” 62
4.1.2 Generated data sets and considerations 64
4.1.2.1 Initial data set 64
4.2 Decision-making algorithm 65
4.2.1 Fundamentals and approach 65
4.2.1.1 Considered potential fields 65
4.2.1.2 High-level algorithm principles 66
4.2.1.3 Expected overall behaviour 68
4.2.2 Algorithm inputs and considerations 70
4.2.3 Reference system and nomenclature 71
4.2.4 Algorithm drive-through 72
4.2.4.1 Car-following section 72
4.2.4.2 Change-intention section 73
4.2.5 Expected algorithm outputs 78
4.3 Code structure 80
4.3.1 MATLAB code for driving scenario data generation 80
4.3.2 MATLAB code for decision-making algorithm implementation 81
5 Results 83
5.1 Variable relative speed 83
5.2 Reverse variable relative speed 85
5.3 Variable traffic intensity 87
5.4 Back overtaking 90
5.5 Traffic congestion situation 91
6 Conclusions and Future Work 93
7 Planning and budget 95
8 References 98
Appendix A. Decision-making algorithm code 101
Appendix B. Simulation scenario generation code 107

Table of Figures

Figure 1. Largest end uses of energy by sector [1]. 20
Figure 2. Thesis work content diagram. 23
Figure 3. Graphic representation of traffic flow linear model. 29
Figure 4. Graphic representation of traffic flow logarithmic model. 29
Figure 5. A psycho-physical car-following model [10]. 31
Figure 6. General hardware of an autonomous vehicle [13]. 34
Figure 7. Example of semantic segmentation [17]. 37
Figure 8. Example of object detection [18]. 37
Figure 9. Autonomous driving sensor set-up [30]. 39
Figure 10. LiDAR measurement techniques [30]. 40
Figure 11. Situation where predictive decision-making of the ego vehicle (blue) would enhance AV operation [22]. 42
Figure 12. Example of single-hidden-layer NN structure [25]. 43
Figure 13. Phases of the Monte Carlo Search Tree algorithm [26]. 44
Figure 14. Basic vehicle notation (left). Forces on tire: Normal, Fn, lateral, Fc and break, Fb (right) [27]. 45
Figure 15. General approach of Weida Wang et. al. work [39]. 48
Figure 16. Graphic description of states [39]. 48
Figure 17. Working principle of the reward system within the RL definition [38]. 50
Figure 18. Graphic representation of the safe free space definition [38]. 50
Figure 19. Decision-making algorithm representation, based on a deep Q Network and formal safety verification [38]. 51
Figure 20. Typical game-theoretical structure for two players and two actions [40]. 52
Figure 21. Structure of the network responsible for payoff parameter obtention [40]. 53
Figure 22. Game-theory-based decision-making structure [40]. 54
Figure 23. Graphic representation of simulation results for traditional Q-learing vs Nash Q-learing algorithms [40]. 55
Figure 24. Cooperative lane changing environment depiction [42]. 55
Figure 25. Collaborative lane-changing decision-making strategy [42]. 56
Figure 26. Representation of artificial potential fields [29]. 58
Figure 27. Bird's-eye view of a lane-changing field experiment [43]. 61
Figure 28. Agent data generation through MATLAB simulation. 63
Figure 29. Theoretical sensor setup on ego vehicle towards point cloud generation through simulation in MATLAB. 64
Figure 30. Image of created two-lane scenario with ego vehicle(blue) and five agents (orange). 65
Figure 31. Representation of potentials and agents' distribution in a line-changing simplified environment. 66
Figure 32. High level depiction of the lane-change algorithm. 69
Figure 33. Chosen reference system for agents and lane marking location definition. 71
Figure 34. Definition of agents (Bi) and ego vehicle (A) dimension and location metrics. 71
Figure 35. Programmatic version of the lane changing algorithm. Car following module. 74
Figure 36. Programmatic version of the lane changing algorithm. Car following module. 77
Figure 37. First set of algorithm outputs for the development dataset. 79
Figure 38 Second set of algorithm outputs for the development dataset. 80
Figure 39. High level code structure. 82
Figure 40. Testing driving scenario for relative speed of 120 KPH. 84
Figure 41. Testing driving scenario for relative speed of 90 KPH. 84
Figure 42. Testing driving scenario for relative speed of 60 KPH. 84
Figure 43. Testing driving scenario for relative speed of 30 KPH. 84

Figure 44. Output signals obtained from a relative speed test of the algorithm. 85
Figure 45. Comparative of the signals obtained for the tested relative speed scenarios. 85
Figure 46. Testing driving scenario for reverse relative speed of 230 KPH. 86
Figure 47. Testing driving scenario for reverse relative speed of 200 KPH. 86
Figure 48. Testing driving scenario for reverse relative speed of 170 KPH. 86
Figure 49. Output signals obtained from a reverse relative speed test of the algorithm. 86
Figure 50. Comparative of the signals obtained for the tested reverse relative speed scenarios. 87
Figure 51. Traffic intensity calculation following Greenshields's lineal model. Testing at Imax, 3000 and 2000 veh/h. 88
Figure 52. Tested driving scenarios for different intensity values. 88
Figure 53. Signals obtained for the tested variable intensity scenario conditions. 89
Figure 54. Testing driving scenario for a back overtaking situation. 90
Figure 55. Obtained signals for the back overtaking tested scenario. 91
Figure 56. Ego-centric view of traffic jam simulation scenario. 91
Figure 57. Testing driving scenario for a traffic congestion situation. 92
Figure 58. Testing driving scenario for traffic congestion situation with faster right lane. 92
Figure 59. Output signals obtained for both tested traffic congestion scenario. 92
Figure 60. Project time plan. 96

1 Introduction

The seek for technological progress has always been inherently bounded to human nature, with the aim of enhancing our capabilities beyond the basic needs of the species. However, pushing to the extreme the efficiency of our systems is a relatively new demand that society is facing, given the accelerated growth of our population in an environment where resources are limited.

Ironically, one of our highest energetically demanding and ecological impacting systems does not add any intrinsic value. According to the 2020 Key World Energy Statistics Report, carried out by the International Energy Agency, the transport sector is responsible for 36% of the global energy consumption, being 21% the fraction estimated for passenger cars [1] (Figure 1).

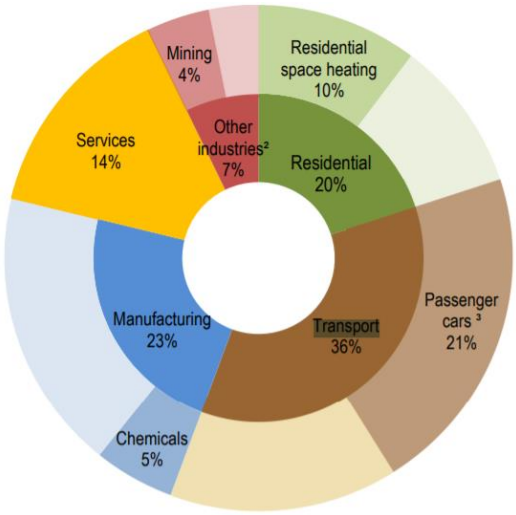

Figure 1. Largest end uses of energy by sector [1].

Given the previous information, and assuming transport of goods and people is inevitably needed for the functioning of our industry and society, streamlining this activity represents a priority within technological development. The two biggest fields of innovation based on achieving such improvement, from an efficiency perspective, are independently focused on alternative energy sources to fossil fuels for vehicle propulsion (e.g., EVs, fuel cells, etc.) and reducing human interaction in the act of transport itself (intelligent transport systems and autonomous driving). Among these two complementary approaches, this document is treating the second one.

Within the field of intelligent transport systems, which approach is achieving an interconnected transport infrastructure (including the corresponding fleet of vehicles) in a way that minimizes human manoeuvres (being these a major source of efficiency losses), individual autonomous vehicles play an important role, as they are potentially independent from such infrastructure and compatible with human-driven traffic systems. If perhaps not being this one the ultimate solution from an energetical efficiency perspective, it certainly involves advantages such as no need for mayor initial investments in infrastructure accommodation and ease of transition with regards to the existing technology (manual vehicles). Therefore, again, the following pages will be narrowed down to the topic of autonomous vehicles only, disregarding a possible connection to an intelligent infrastructure.

From a general perspective, human-driven vehicles elimination idea involves several advantages, including human labor reduction, ability to perform parallel tasks while being driven to the destination, stop spending the time of the driver in parking slot search, saving parking space as the driver does not have to exit the cabin, etc. However, as depicted before, one of the main outcomes of the introduction of automated driving is the ability to frame an extremely efficient transport system. This improvement relies on the use of several redundant sensors for making the correct decisions in a fraction of the time a human would (with the corresponding implications in safety enhancement), which further results in a more efficient travel [2].

These vehicles would potentially benefit from efficient onboard algorithms, communication with vehicles around and transportation management centers. This, combined, eases the optimization of vehicle movement. They can also operate in high-speed platoons, significantly increasing the operational efficiency and collaborate to avoid congestions, increasing road's traffic capacity. This implies that the overall effect of a single AV might be reduced. However, considering a network of such vehicles has the potential of considerably increasing the efficiency of the whole transportation sector [2].

On the other hand, despite being safety one of the main concerns about the rise of autonomous driving technology, the improvement of driving safety is the other mayor justification for such investment. To shed some light on this aspect: 93% of the people interviewed in the United Stated showed, in the Swedish Psychology journal "Acta Psychologica", that they would rate-themselves as above-average drivers [3]. However, each year 1.35 million people die as a result of a traffic accident globally [4] and, as probably expected, The NHTSA has identified human error as the origin of 94% of road accidents [5]. Therefore, eliminating human interaction during human transport could drastically reduce the amount of traffic accidents, together with the subsequent deaths and injuries.

Despite the considerable late progress in the field, autonomous driving is still facing certain technological challenges, which are representing a barrier for it to develop to its full potential. Specifically, the part of the system that deals with the perception of the environment is far away from the precision and flexibility of human sensing. Therefore, is important not only to further develop AV perception technologies, but to elaborate decision-making algorithms that can alleviate the dependence on perception precision.

Typically, being based on the NHTSA's policy, the level of automation of AVs is graded in a scale from 0 to 4. Within that range, Level 0 represents no-automation. The driver is in complete and sole control of the vehicle. Level 1 implies automation relegated to only low-level control functions. Level 2 involves the automation of at least two primary control functions, relieving the pilot from the act of driving. Being able to cede full control to the system is considered Lever 3, under stable environmental conditions, requiring transition back to the driver when those conditions overcome the vehicle's capacity. Finally, in Level 4, the driver becomes a passenger and provides the destination or navigation input, but it is not expected to be available for control during the trip, including both occupied and unoccupied vehicles [6].

Ultimately, Level 4 of automation will inevitably have to deal with considerably dynamic, safety-compromising scenarios, relying in advanced perception system and proficient decision-making. These environments have been identified as: Dealing with pedestrians, managing preference of access in crossroads and roundabouts, entering, and exiting highways, overtaking vehicles involving lane-changing, etc.

Therefore, being the state of the art focused in fully defining the above, this thesis aims to provide a decision-making algorithm destined to the lane changing operations, towards vehicle orientation, compatible with a perception system that relies solely on LIDAR sensors.

2 Objectives & Methodology

As mentioned above, this work revolves around the main objective of creating an algorithm valid for the decision-making that lane changing involves, in the field of automated driving. Under certain assumptions and simplifications, that will be described together with the algorithm, it should be able to analyse the environment and return one main signal, which is, the confirmation to initiate the change of lanes.

2.1 Project objectives

This main goal involves several secondary objectives, listed in order of importance:

a. Creating an algorithm that prioritizes safety: Being safety one of the most important outcomes expected from automated transport, the decision-making should be based on allowing a lane-change only under certain safety conditions. The main metric safety will rely on must be safety distance, which will be calculated continuously, using a dynamic model. This model should not only consider the distance to the considered vehicle, but also its relative velocity to the autonomous vehicle, hereinafter regarded as the "ego-vehicle".

b. Analysing Lane-change decision efficiency: After safe changes are guaranteed, the implementation of the algorithm must involve an improvement in efficiency, in comparison with a human-driven vehicle. After such analysis, it is also possible to calculate how would the increase/decrease of decision-making efficiency affect the traffic flow from a macroscopic perspective.

c. Making the code compatible with LIDAR perception: It must be designed to receive the inputs from, at least, LIDAR perception. As it will be described, usually, automated driving implements data-fusion to process information from several perception systems working in parallel, such as LIDAR or image recognition. For the sake of simplicity, only LIDAR will be considered in this case, being known as a reliable system for reactive methods.

d. Creating valid environments for simulation: During the early stage of the code creation process, environment data from simulations will be used. This implies high flexibility in terms of datasets size scalability and diversity. Also, it will allow to implement certain simplifications, needed to be able to keep the focus on the main objective of the thesis. These simplifications will be described in the following sections. However, simulation environments must be chosen so that they allow developing an algorithm valid for real scenarios.

e. Algorithm testing and metrics calibrating: As the functioning of the program will be based on several pre-set metrics, such as waiting time and safety thresholds, it will have to be repeatedly tested so that they can be calibrated according to the analysis of the outputs. Likewise, the logic of the algorithm itself must be validated through testing and results analysis.

Within this section, the main objectives of this work have been described, being the secondary ones a break-down of the main one. The individual consecution of each objective,

as well as a discussion of the level of completion of the main one, will be included in this text under the chapter: *Conclusions and Discussion*.

2.2 Project methodology

The workflow has followed the Scrum model, typically used in software development, as it has been proven to be suitable for the design of such complex systems. Scrum is an agile development method that relies on a continuous iteration process, where the outcome gains definition after each iteration, also called "sprint". Figure 2 shows sprint distribution, where each of them, according to the Scrum model, ends with an evaluable version of the outcome, these are, a high-level algorithm (Sprint 1), a tested code under ideal conditions (Sprint 2), a code that includes LIDAR perception (Sprint 3) and a final analysis of the outcome (Sprint 4).

This project is centered in the theoretical side of the algorithm, covering the first two sprints depicted in the figure below. However, the complete project is now conceived, indicating the route for future development. The structure of the work is organized as shown in the following image:

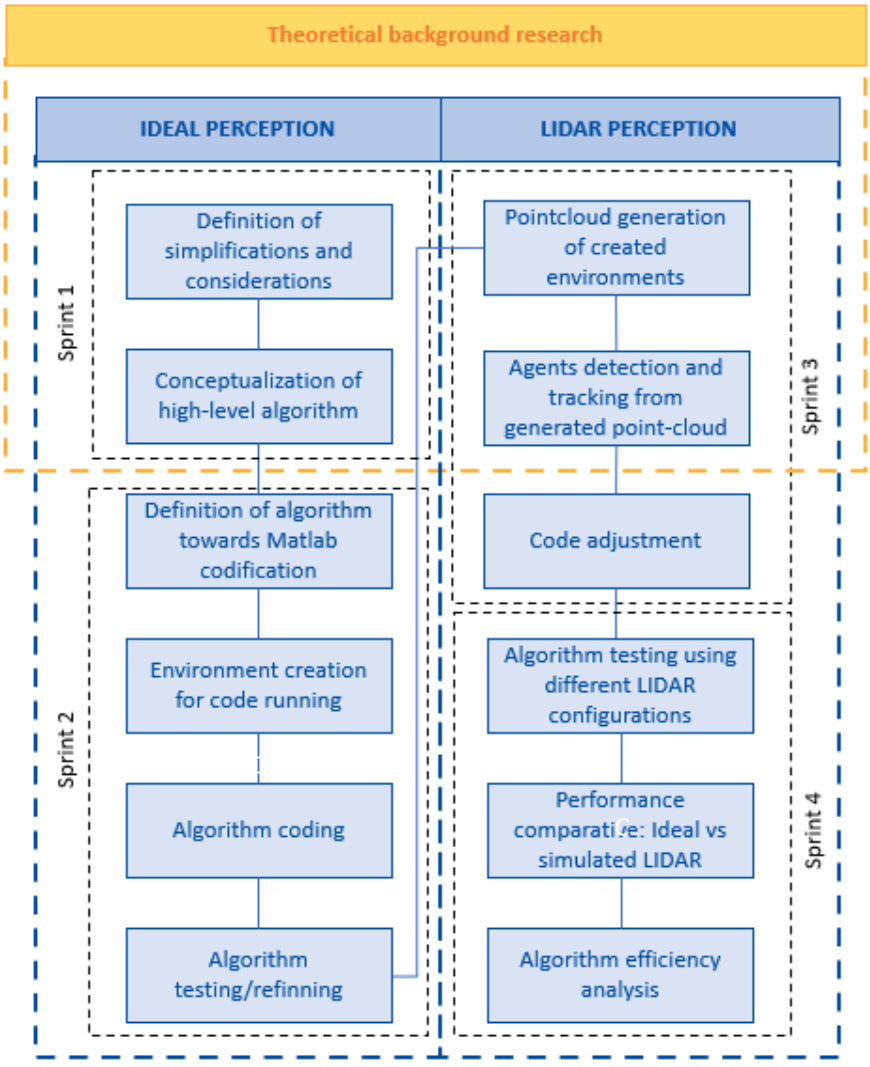

Figure 2. Thesis work content diagram.

The work content is mainly divided in two modules. The first one (left) corresponds to the initial part of the algorithm development, where ideal perception is supposed, for the sake of simplicity, until the basic aspects of the code are validated. In the second one (right) LIDAR perception is included as the way of generating the input for the algorithm, being the main input the position of the agents on each timeframe of the simulation.

Work content where theoretical research played an important role corresponds to the initial phase of each of the modules, meaning that the research was not only carried out at the beginning of the project. The outcome of the research will be described under the theoretical background chapter.

Finally, as the continuous line shows, these two modules have been approached sequentially, starting from the algorithm creation under an ideal perception environment, continuing by addressing LIDAR implementation. Each of the tasks are, in chronological order:

IDEAL PERCEPTION SCENARIO

1. Definition of simplifications and considerations
   Once the theoretical background and the state of are understood, through a scientific literature research, the project begins by framing the problem to solve. In this case, a considerable number of simplifications and assumptions had to be considered from an early stage, given the implications on the structure of the algorithm. As it will be described, an autonomous driving system relies on several sub-systems, such as location, perception, control, etc. Therefore, as the focus will be on the decision-making, non-essential sub-systems will not be considered, and the information shared by the essential ones will be assumed and simplified. These assumptions and simplifications will be described further in the corresponding section.

2. Conceptualization of high-level algorithm
   After the problem is framed in terms of what the high-level output of the algorithm will be, which was defined through the definition of the objectives of the thesis, and so have the inputs needed in the prior step, a high-level version of the algorithm is constructed. This will serve as a basic layout for a later code-level definition, containing the main modules the program will include. Likewise, at this stage, the mathematical model's decision-making is going to be based on are defined. As it will be described, the algorithm will rely on an abstraction of the potential fields model.

3. Definition of algorithm towards MATLAB codification
   Once a high-level system has been defined, as well as the mathematical functions behind key decisions, a low-level algorithm is drawn, with a considerable increase in the level of detail. This will serve in the same manner as a template for the writing of the script in MATLAB. In this stage, the algorithm is halfway between the decision-making primary concepts and leading with format of the data. In this way, it will keep the same basic structure of the prior algorithm, but it will be broken down into sub-modules, based on programmatical logic, foreseeing inputs and outputs as the code would do. For such task, the format of the data the code will use as an input has to be defined from this stage, which is why it is carried out in parallel with point 4.

4. Environment creation for code running
   At this point, the creation of the environment represents the replacement of the essential sub-systems, as mentioned above, so that the algorithm can perform. The two main outcomes of this stage are, firstly, generating data to an environment that is

simplified, but realistic enough to allow a future algorithm implementation without the need of mayor changes and, secondly, establishing the format of the data that is going to be handled in, through a first iteration of the environment creation, so that the low-level algorithm can be written. Likewise, environment creation will take place a second time after algorithm coding, to generate a set of data which will serve as the baseline for the algorithm testing and refining.

5. Algorithm coding

Taking the low-level algorithm as a template and having extracted the first data-frame from the environment creation initial iteration, the code of the decision-making program is written in the shape of a MATLAB script. As it will be described later in more detail, the focus of the coding is on writing functions corresponding to the different blocks of the algorithm's diagram and accessing in an ordered and efficient manner the specific data from the environment data-frame so that calculations can be performed, and the relevant information can be shared between functions. Likewise, during the coding stage, the way of understandably and concisely showing the decision-making results is ideated and programmed.

6. Algorithm testing/refining

As an outcome of this section of the project, different data-frames are created, temporarily moving back to stage 4, resembling a suitable set of scenarios to test the script against. During the testing process, all the debugging needed will be carried out but also several pre-set variables the algorithm is using must be calibrated, which mainly are decision-making thresholds, user defined settings, etc.

LIDAR PERCEPTION SCENARIO

7. Point cloud generation of created environments

One of the key steps of any software development is its validation process. Especially when dealing for software for autonomous driving, it is important to contrast the functioning of the algorithm with real data. In this case, for the sake of simplicity, the use of real data will be avoided, as the real data needed would have to be very specific (this will later be addressed with more detail). Therefore, instead, the approach will be to generate the point cloud that corresponds to the 3D simulated environment and the location and characteristics of LIDAR sensors located in the ego vehicle for the simulation. Even though this represents a considerable simplification, one is allowed to easier compare results with and without perceived LIDAR point cloud as an input, of the same environment, so that the limitations that the LIDAR perception introduces in the functioning of the algorithm can be directly observed.

8. Detection and tracking of agents from generated point-cloud

Once the point cloud has been generated, the algorithm will not be able to work with the rough data, but with a data struct containing position, dimension, and velocity of all the agents perceived, on each timestamp of the simulation. Therefore, it will be needed to implement vehicle recognition and tracking. For this purpose, the LIDAR Labeler tool of MATLAB will be used, which uses a pre-trained neuronal network to track the points of the point-cloud corresponding to moving vehicles once the identification of those is done manually. The output of the execution of the program is a file containing the prior-mentioned information about the cuboids the tracked vehicles have been fit in by the tracking algorithm.

9. Code adjustment

   This step is needed as the format of the data struct belonging to the simulations from the environment creator tool is different from the one extracted from the LIDAR labeler one. As the code of the algorithm heavily relies on a systematic extraction of data from this struct, a considerable part of the code will have to be modified.

10. Algorithm testing using different LIDAR configurations

    Once the code has been modified accordingly to obtain the same results as the ones obtained with the prior version. It is simple to consider several configurations of LIDAR distribution. The settings of the environment simulation are changed for each LIDAR disposition (number, orientation, number of layers, range, etc.) and the simulation is re-run to extract the new data struct containing agents positioning together with the corresponding point-could, then the LIDAR labeler is used to extract again the data of the agents and the quality of this data is compared with the "ideal" original one. Although the process is relatively time consuming, the outcome will be identifying which is the ideal disposition of LIDAR sensors in the vehicle.

11. Performance comparative: Ideal vs simulated LIDAR

    After the optimum LIDAR disposition is achieved, both versions of the code will be run so that results can be systematically compared. This comparison will assume that the code that does not include LIDAR perception will achieve instant ideal results. Therefore, a measure of the lag on the different lane change confirmation signals, once the second code is run, will be used to judge the theoretical limitations of LIDAR perception implementation.

12. Algorithm efficiency analysis

    To conclude this work, the analysis of the work delivered will revolve around an estimation of the efficiency improvement that its use involves, compared to manual driving. As it will be described, the approach will be of implementing a deterministic, reactive model that will be the basis of the decisions made. Therefore, a simple, straight forward way of performance comparison will be the time it takes for the ego-vehicle to provide with a change confirmation vs a human, estimating the effect of human reaction time and ego-vehicle perception limitations, assuming that the change confirmation in both cases is always correct.

3 Theoretical background

Under this section, the theoretical background needed to contextualize the work done on this thesis will be exposed. The aim has been to synthesize the text to keep the focus only on the essential information for the understanding of the decision-making algorithm, as the largest part of the thesis will revolve around the algorithm creation process and result analysis.

3.1 Summary of traffic theory and intelligent transport systems

Firstly, a summary is necessary to set the minimum theoretical framework needed to be able to analyse the posterior efficiency of the algorithm, followed by the concept of intelligent transport systems, typologies, and status of the technology.

Introduction to traffic theory

The aim of traffic flow theory is to mathematically describe the interaction between drivers, vehicles, and infrastructure, being the infrastructure the immobile actor, which characteristics influence the behaviour of the mobile ones, the vehicles [7].

However, traffic flow represents a highly complex phenomenon, which has been mathematically modelled from a broad variety of approaches. The movement itself of each individual vehicle responds to several physical phenomena, from the biological and behavioural implications of the perception and actuation of the driver to the electromechanical systems that define the dynamic of the vehicle. If it is also considered the simultaneous presence of numerous drives and vehicles, influencing each other in the same scenario, it becomes evident that a deterministic approach towards the study of the traffic flow has reduced utility. Instead, despite the behavioural discrepancies between individuals, stochastic models have been proven to successfully define the dynamics of traffic flow, especially when being analysed from a macroscopic perspective. In fact, being the sample of vehicles considered big enough, their behaviour of the specimens as a hole becomes surprisingly close to the one of any fluid, being defined by metrics that are direct analogies from the variables that constitute the basis of the theory of fluid mechanics.

However, such analogies only entail a fraction of the traffic flow theory, which gathers several different analytical methods linked to different sets of traffic flow characteristics, such as road capacity, flow-density relation, headway distribution, etc. Examples of analytical methods linked to them are, on the other hand, shockwave theory and microscopic simulation models [8].

As an outcome, traffic theory has become relatively accurate in predicting when traffic jams and queuing are occurring, given a certain demand level, as it can also precisely estimate how long these queues will be and how they would propagate in space and time, including the moment of decongestion [8].

Macroscopic characterization of the traffic flow

From a general perspective, the macroscopic study of the traffic involves the establishment of the relations between the variables that describe the behaviour of traffic flow as a separate entity from the accumulation of individual vehicles that conform such flow, although there is obviously a link between both perspectives. Such variables are, mainly:

- Traffic intensity (i):
  Defined as the number of vehicles that go through a determined section of the road per times unit, is normally set as a probabilistic function of the likelihood for a certain number of vehicles, which is modelled as a Poisson distribution for light traffic and (<= 200 veh/h) and as a binominal function for greater values, shown below respectively.

$$Poissons\ distribution: P(i) = \frac{e^{-qt}(qt)^i}{i!} \qquad i = 0,1,2 \dots$$
$$being, q = average\ number\ of\ vehicles\ passing\ through\ the\ section$$

$$Binominal\ distribution: P(i) = \binom{m}{i} * P^i * (1-P)^{m-i} \qquad i = 0,1,2 \dots m$$

- Traffic velocity (v):
  Which represents an average value of the individual velocities of the vehicles, either applied to a local point of the road during a certain timespan, a spatial segment of it or a certain itinerary. It generally corresponds to a normal distribution.

$$f(v) = \frac{1}{\sigma * \sqrt{2\pi}} * e^{-\frac{(V-\mu)^2}{2\sigma^2}}$$

  Being average temporal speed and average spatial speed defined as, respectively:

$$ATS = \frac{\sum \frac{d}{t_i}}{n} \qquad ; \qquad ASS = \frac{d}{\frac{\sum t_i}{n}}$$

- Traffic density (K):
  Meaning the number of vehicles per spatial unit that occupy a certain segment of the road in a determined timeframe. Being traffic density the metric that is more directly linked to traffic jams. It is known that maximum density of a standard road is approximately 160 vehicles per hour.

The relation between these three variables is, if expressed by their mean values, well known to be:

$$i\left(\frac{vehicles}{h}\right) = V\left(\frac{km}{h}\right) * K\left(\frac{vehicles}{km}\right)$$

However, it is when establishing the relation of two of these variables, or when the consideration of mean values is not enough, when there is room for different mathematical macroscopic models. These models make use of a few pre-established values:

- Road speed limit: V_l
- Maximum intensity road speed: V_m
- Critical traffic density that causes a traffic blockage: K_c

The broadest spread models are:

- The Linear model (Greenshields, 1935) [9]:

$$V = V_l\left(1 - \frac{k}{k_c}\right)$$

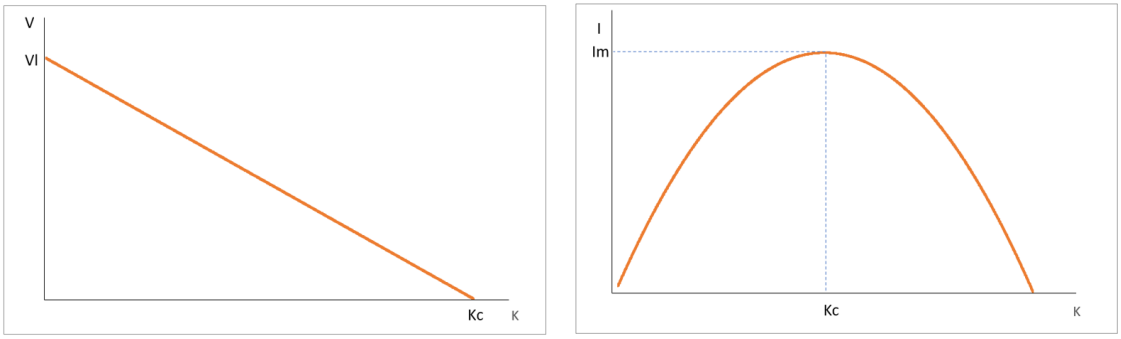

Figure 3. Graphic representation of traffic flow linear model.

- The logarithmic model (Greenberg, 1959) [9]:

$$V = V_m * Ln\left(\frac{K_c}{K}\right)$$

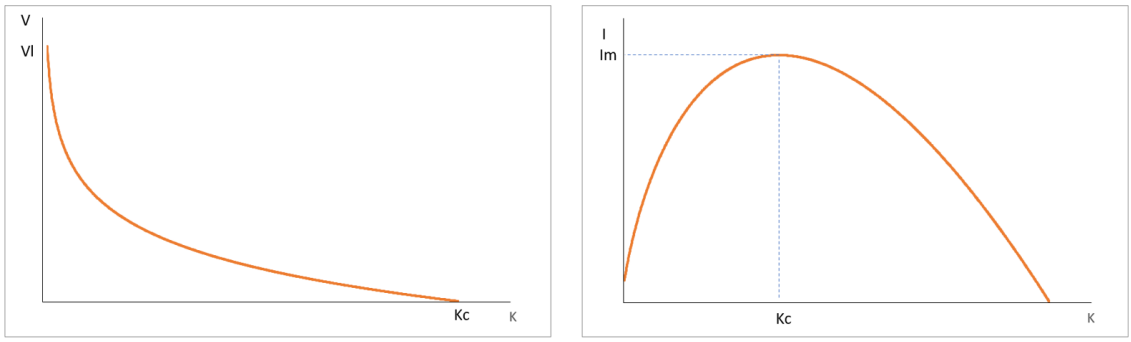

Figure 4. Graphic representation of traffic flow logarithmic model.

As an outcome, considerably important theories, applied into the study and control of traffic, such as shockwave theory, which describes the dynamics of traffic jams, are founded in the previous models. However, to not deviate from the focus of the thesis, only these basic macroscopic models will be exposed.

Microscopic traffic models

When the interest is on being able to model the dynamic behaviour of a specific vehicle, which is influenced by the infrastructure geometry and the interactions with the rest of the road users, a macroscopic study will not provide such information, but microscopic models instead. These models are especially relevant as they allow to generate simulated environments that serve as a base for developing several autonomous driving software features.

There are numerous microscopic models that are not only differentiated in the approach, but also the traffic situation that they pretend to model. Some examples of the broadest spread microscopic models are:

3.1.1.1 Free-flow models

When vehicles freely drive through the road with minimum interaction with other road users, normally due to the reduced amount of them in the surroundings, free-flow, or free-speed models are the ones that better adapt to the behaviour of human drivers in these kinds of scenarios. In these cases, the driver's desired speed, speed limit of the road and its geometry are the main variables that influence the movement of the vehicle [8].

Most free-flow models, therefore, are relatively simple and use as main variable desired speed as a function of road speed limit. The control of the vehicle then gets reduced primarily to the lateral course keeping within one lane. However, it must be considered that not only the desired speed changes from driver to driver, but also within one driver given a certain timeframe. In any case, most models simplify this phenomenon and assume that desired speed has a driver-specific constant value [8].

Nonetheless, as soon as traffic conditions slightly deteriorate, the driver will no longer be able to freely select the desired speed, as he will be constrained by the decisions of the rest of the users. Therefore, car-following models play normally a more significant role in the study of traffic and simulation traffic generation, which is why, hereinafter, those will be the models that will be explained in more detail and applied to the practical phase of the thesis, given the case [8].

MITSIM, a simulation environment that incorporates a free-flow model for one of the regimes it contemplates, presents the following shape [10]:

$$a_n = \begin{cases} a_n^+ & V_n < V_n^{desired} \\ 0 & V_n = V_n^{desired} \\ a_n^- & V_n > V_n^{desired} \end{cases}$$

Where a_n^+ represents the maximum acceleration rate a specific vehicle can have and a_n^- the normal deceleration rate, measured both in m/s2 [10].

3.1.1.2 Car-following models

These models rely on a considerably different approach. Driver modelled behaviour primarily depends on the one of the pre3cedent vehicle of the same lane. It is when if driving at the desired speed of the ego-driver become the reason of a collision with the precedent vehicle, when a car-following model is normally applied. Most models use as the fundamental variable to control the acceleration of the following car, but some of them consider its speed instead. Initial models, like the car-following model developed by Gipps in 1986, could be described as simplistic as they considered only one regime of speeds/accelerations. However, most recent simulation environments apply models that change depending on the regime, integrating fre-flow, car-following and even emergency deceleration functions [10].

Depending on the base logic the models are made upon, these can be divided in:

- Stimulus-response or Gazis-Herman-Rothery models (most utilized ones so far), which are stimulus-response functions based. GHR models depend mainly on follower's speed, difference of speed with the leader and space between them. Likewise, as in most of the cases, GHR models make use of different parameters to control the proportionalities between variables [10].

  The following is an example of a stimulus-response based model:

$$a_n(t) = \propto * V_n^{\beta}(t) * \frac{V_{n-1}(t-T) - V_n(t-T)}{(X_{n-1}(t-T) - X_n(t-T))^{\gamma}} \quad [10]$$

Where a represents acceleration, V velocity, x position, t-T time difference and the rest are the above-mentioned calibration parameters.

- Safety distance or collision avoidance models, which rely on the assumption that a safety distance is always kept between the two vehicles. this safety distance however, in most elaborated collision avoidance models, is not constant, but the result of a manipulation of Newton's equations of motion [10].

  One simple safety distance model that can serve as a reference is the one developed by Pipes in 1953, when the first car-following models were being developed [8]:

$$s_i = S(v_i) = S_0 + T_r v_i \quad [8]$$

  Where S_0 is the effective length of a stopped vehicle including an additional distance in front, T_r is a parameter corresponding to the reaction time of the follower, s_i is the distance between follower and leader and v_i is the speed of the follower.

- Psycho-physical models, presented as a new approach in 1963, are also defined as action point models. Whereas traditional GHR models assumed that there was also a response of the follower regardless the distance to the leader (which is normally corrected by the consideration of different driving regimes, as stated above) and driver's reaction was immediate, psycho-physical models use thresholds for driver's behaviour changes. In this way, a driver will only be able to react to a change of the motion of the leader once a certain threshold on the variable that describes such phenomenon has been reached. The idea is clarified in the following graph [10]:

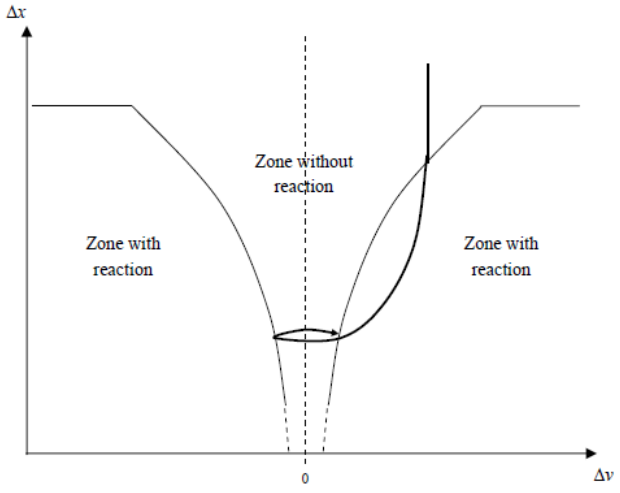

Figure 5. A psycho-physical car-following model [10].

- Fuzzy logic, on the other hand, has been trying to better represent the implications of human decision-making processes into the model by trying to quantify them through logic rules. In this case, drivers are not simply assumed to know the exact values of the parameters that drive their behaviour, such as speed or distance. Alternatively, for example, the value of a variable would be categorised between very

low, low, moderate, high, or very high, leading to different behaviours. These sets may in many cases even overlap between each other, having then a probabilistic density function in the overlapping segments that would determine the probability of the driver acquiring the behaviour corresponding to one set or another. This approach seems the better way of approximating to human behaviour. However, little attempts have been made into calibrating such models [10].

Intelligent Transport Systems. Definition and typologies.

Under this heading, a general overview of Intelligent Transport Systems (ITS hereinafter) will be described with the aim of providing the reader with the reference of the former field of industry this work belongs to, having acknowledged that the content of the section itself is not directly related with the practical part of the text.

The primarily function of ITS is improving the safety, performance, and efficiency of transportation, by making use of information and communication technologies. The literature normally describes the three main agents this communication is established among as vehicle, infrastructure, and user (referencing both drivers and passengers). With the purpose of establishing such communication, the system is divided into sub-systems responsible for collecting the relevant information that enables a description of the traffic scenario, processing and integrating such information and transmitting it efficiently to the end users of the road, respectively. As a result, ITS allows applications such as route planning, dynamic traffic management, reporting traffic events such as traffic jams or accidents (which are ultimately also considerably reduced), etc. Specific examples of the tools ITS integrate to provide such functionalities automatic traffic management systems, public transport information services, traveller information systems, fleet management and location systems, emergency management, electronic payment, and cooperative vehicular systems [11].

Some of the most relevant technologies and applications within ITS are described below:

- Wireless Vehicular Communication:

A well-established section of ITS is cooperative vehicle communication through different systems that enable it. This process is sub-divided in communication vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I) and infrastructure-to-vehicle (I2V) [11].

Connected vehicles flow of information is normally real-time based, improving operation safety, fuel consumption reduction and mobility efficiency, while being applied on driverless vehicles (autonomous driving) and/or non-autonomous. The communication, although is not only constrained to it, is normally established through GPS and Internet-provided resources [12].

Likewise, there is a distinction between independent or autonomous systems and integrated, single-platform-sharing ones. Public busses wireless communication with traffic lights to establish passing priority in an intersection is an example of a currently applied independent system, whereas novel efforts in the field are focused in two-way communication with open protocols to allow multiple services and applications in parallel, distributed in several mobile nodes (vehicles) that share information with the infrastructure. Hardware such as antennas mounted not only in the infrastructure nodes, but also on the roofs of the vehicles, have a crucial role in the performance of these kind of systems, where vehicles interaction with electromagnetic fields implies one the main technical challenges, together with channel characterization, and the impact of human exposure to such fields [11].

- Sensor-provided Networks and Surveillance:

New sensing technology is likewise being provided to cover the demands of ITS in terms of data acquisition, that traditional methods result to be inefficient in terms of performance, cost and maintenance. These technologies are normally catalogued as Sensor Networks and have potential applications in both highways and urban areas, collecting data with the purpose of planning and managing operations on the road together with delivering information to the road users [11].

The data transfer can either be performed using an ad-hoc system or making use of the infrastructure, being also possible to have a hybrid set-up. Two typologies of sensors are likewise distinguished, which are on-road and on-vehicle sensors. Communication set-ups and different types of sensors end up having numerous types of installations, being the wireless version of these systems where industry is currently focused on, given the operational advantages that they provide in comparison with wired networks [11].

Currently, a considerable number of urban areas employ loop detectors, which are sensors integrated in the material of the road for vehicle counting and speed measuring towards traffic surveillance. Likewise, other systems use closed circuit television, radar, laser, and image processing techniques. On the other hand, on the vehicle side of the system toll tags or GPS are also used for counting and speed measuring purposes [12].

- Data Processing Techniques:

In last place, one of the main technological challenges ITS needs to overcome is dealing with the drastic increase on data processing needed in comparison with the traditional methods mentioned. Specially, this situation is attributed to the need of real-time data which acquisition must be managed, transferred, interpreted, aggregated, and analysed prior to real-time decision making. This data increase is lead likewise by the fact that all the agents mentioned of the ITS system act as data generators and data sources, simultaneously, using novel types of sensors that are also individually capable of acquiring high volumes of data to be processed, together with the general increase of use of the infrastructure by the end users. Therefore, a considerable effort is being put into new data processing and mining techniques, where optimization algorithms play an important role [11].

Within the ITS, the content of this work fits under the field of autonomous driving, being a technology that, even though often treated as a separate topic, entails a shared framework with wireless vehicular communication. Moreover, some of the assumptions regarding acquired data for the sake of the decision-making algorithm simplification, rely on existence of intelligent transport systems up to some degree.

3.2 Introduction to autonomous vehicles

Typically, an autonomous vehicle, given a state of the art that heavily involves continuous change based on latest research, involves the procurement of a commercial vehicle that serves as a platform for development of autonomous driving. It is instrumented with sophisticated sensors that serve as a perception system for a computer program, as human drivers use sight and hearing to shape the driving scenario. These sensors are connected to a cluster of computer systems networked to shared data, which on real time elaborate an output that is

passed to actuators of the steering, brake, and throttle, which drive the vehicle. The intelligence of autonomous driving is hidden in the software of such computation system, being responsible for the mapping of the sensory percept towards the actuation of the actuators. Given the large number of sensors generating a considerable amount of data, mapping techniques become extremely complex, which likewise involve complex navigation manoeuvres [13].

From a software perspective, the overall architecture is similar to the problem of mobile robot navigation, which follows a cycle of sense, plan, and act. As mentioned, the data of the environment received by multiple sensors is processed and used to plan a move. After the decision is made and sent to the control system, the vehicle changes its position and therefore the environment around detected by the sensors based on which the plan is revised, and a new action is taken. The whole system might not operate under a single layer of hierarchy, but several ones, having the decisions made at multiple layers. Each layer relies on the information from different sets of sensors. Top layers respond for long term decisions linked normally to route planning, working at lower decision-making frequencies, whereas lower layers operate at high frequency and oversee simpler decision-making such as monitoring the control error for generating control signals [13].

A scheme of how the different systems interact in an autonomous vehicle is shown:

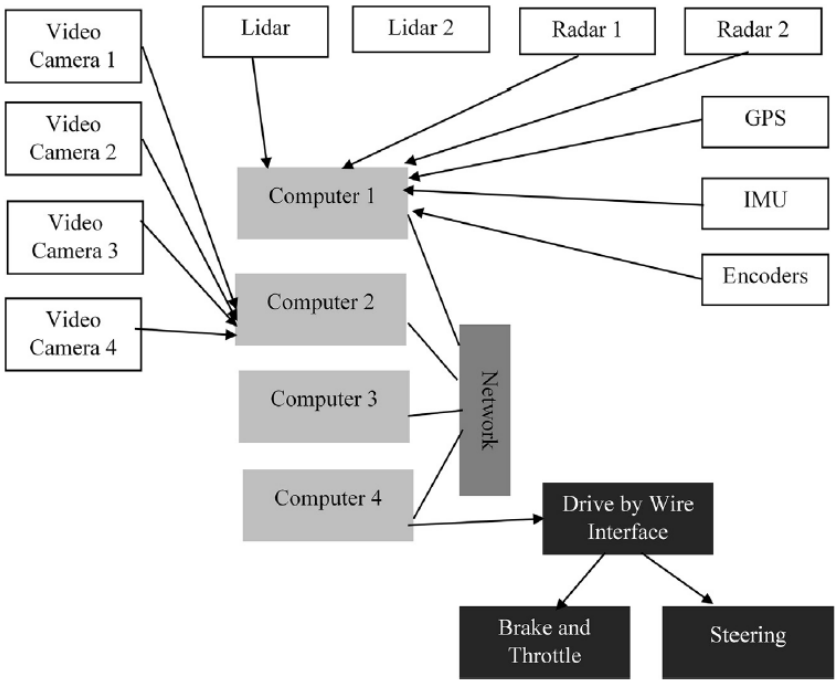

Figure 6. General hardware of an autonomous vehicle [13].

Positioning systems in AV

One of the essential tasks of autonomous driving is real-time setting of the reference point that serves as a base for the perception, decision making and actuation. Location and speed are the main variables and dealing with the errors inherent to different methodologies acting in parallel is not a trivial topic.

3.2.1.1 Internal vehicle positioning

Sensors in an AV are not only used for perceiving the environment, but also to tack its own location estimating the based on the latest position measurement and the error generated. The system that deals with such measurement is called IMU (Internal Measurement Unit), which

typically consists of an accelerometer, a gyroscope, and a magnetometer. While the accelerometer perceives the acceleration in the three axis of its own coordinate frame, the gyroscope measures in the same way the angular orientation and magnetometers are also used to measure changes in orientation changes of the vehicle. Likewise, encoders can potentially be fitted on the wheels so that rotation speed can be calculated. This motion measurement is used as a redundant check of the position, orientation, and speed of the AV. However, all these sensing methods accumulate a certain amount of error besides the on inherent to the sensor itself, as neither of them accounts for slippage when the vehicle looses traction, which is why different internal and external sensing approaches are used together to overcome these limitations [13].

3.2.1.2 Vehicle localization

Generally, GPS (Global Positioning Systems) are used as an external source for vehicle positioning in the road in terms of the global coordinate frame (longitude and latitude). The location is achieved by communicating with satellites that constantly broadcast their location being synchronized by an atomic clock. However, current GPS technology implies a considerable error, especially when reception is low or the vehicle is surrounded by tall buildings, which is why its information is usually delegated to higher-order decision-making [13].

As an alternative to the lack of robustness of GPS as a standalone location system, electronic maps and IMUs can be used, being subject of latest research. Moreover, in AD, accuracy demands are beyond what traditional electronic navigation maps can offer. Therefore, High-Definition Maps (HDM) are used instead as an important component for AD systems, as they provide a-priori information that include lane lines and road signs. They also provide a large amount of semantically rich data to guide an AV at a finer scale, ensuring an “over-the-horizon” perception capability towards decision-making and control [14].

Different levels of automation require different typologies of HDMs. Level 3 (conditional autopilot) requires the most basic kind, a static HDM; level 4 (highly AD), a Dynamic HDMs and level 5 (Full AD) a so-called Smart HDM. In general, HDM solutions go from high-cost approaches, such as LIDAR perception combined with vision sensors, GPS and IMU in a mobile road measurement system for acquisition, to low-cost ones that involve artificial intelligence to extract information of HD aerial photographs [14].

An ordinary electronic navigation map has an accuracy between 5 and 10m, being able to describe only location and shape of the road. An HDM, however, can depict location, shape and attributes of lane lines, road signs and junction indications. The industry of AD defines that the accuracy of an HDM to be functional should be between 1m and 20cm [14].

In case of not disposing of an electronic map of the road, a map of the environment is still needed so that the decision-making algorithms can perform. Real-time mapping algorithms are then used based on the current position of the vehicle, where all the obstacles and regions of interest are plotted and tracked. Vision sensors are used then to recognize position estimates with respect to landmarks and neighbourhood objects. However, this information might as well not be reliable as sensors are noisy and land-marking are subjective to not always be within sight or not reliable. Therefore, mapping requires localization and vice versa, conforming a problem known as Simultaneous Localization and Mapping (SLAM) [13].

In the SLAM method, both perception and localization are tasks performed in real time. This technique relies on continuously monitoring the environment to easily adapts to it. Although HDM provides a robust solution for static objects as it is not susceptible to on-board sensors range limitations, SLAM is able to cope with moving objects and quickly changing scenarios [15].

Perception systems in AV

Perception systems entail the hardware and corresponding software related to the sensing activity of the autonomous vehicle. The data received will be processed towards decision-making, which output will be handed to the control unit of the vehicle, consisting of the actuators needed to manoeuvre the vehicle.

Although the information processed does not come only from perception systems, as it is also acquired through the IMU, infrastructure and other vehicles (as previously stated), typically, the represent the most important source of information low-level navigation relies on. The technology that is currently handling perception is divided into:

3.2.1.3 Vision-based

Vision systems, simplified, consist of image recognition through cameras. The popularity of this setup relies on its similarities with human perception, which is mostly based on the sense of sight, when driving. Therefore, most road signalling, designed for human vision, is highly compatible with image recognition techniques [16].

However, computer vision should not be considered as a robust standalone system, but a type of perception that should be complemented with other sensing technologies, for the sake of safety and perception robustness, applying data fusion techniques to be able to merge real-time information. Moreover, computer vision has become a solid solution for autonomous driving perception once machine learning techniques, specifically deep learning, emerged to solve some aspects of image recognition [16].

Traditionally, vision perception relies in carrying out two main tasks, commonly identified as object detection and semantic segmentation. Semantic segmentation is normally used to divide the real-time image in sections, based on certain characteristics of the image at a pixel level. It is especially useful when wanting to focus on certain regions of interest and discard the rest, when approaching elements of the image that are continuous and are not suitable to be treated as punctual objects, such as the road, sidewalks, or the sky. Through semantic segmentation, certain rules can be applied so that the vehicle always remains on what is recognised as the road or to discard the data belonging to the sky, to enhance computational efficiency. Likewise, semantic segmentation can also be applied at object level to distinguish between different typologies of obstacles. For example, within the detection of vehicles, semantic segmentation could be applied to differentiate between a car, a truck, or a bus, involving different corresponding behaviours of the AV [16].

More concisely, the final objective of semantic segmentation is assigning a class to each individual pixel of a real-time changing image. As processing each pixel individually would be excessively time consuming, a pre-processing is performed in groups of pixels called super-pixels. These groups of pixels are assigned based on low level features, such as proximity, colour, or texture. Then, a trained classifier, like the ones mentioned for object detection, is used for semantically dividing these super-pixels. Then, spatial coherence techniques are implemented for reclassifying these groups once considered the classification obtained for their neighbours. Finally, each pixel is simply assigned the class of the super-pixel they belong to [16].

On the other hand, object detection focuses on identifying whether an image contains an instance of a specific class or not, and subsequently delimit its dimensions in a simple shape, such as a rectangle acting as a bounding box, so that it can be easily tracked, instead of using the full silhouette of the object as semantic segmentation would. Within object detection, in order to be able to identify a type of object, for instance, a pedestrian, which can adopt numerous different poses and dimensions, the following three main steps have traditionally been followed: Generation of candidates, mostly based only on the size of the object (i.e. canonical size of a pedestrian), classification of candidates and refinement (applying machine

learning techniques which make selection decisions on a stochastic basis where the optimum threshold is obtained using training & validation datasets). After the object is detected, the effort is focused on tracking it [16].

A graphic example of the difference between object detection and semantic segmentation is shown below:

Figure 7. Example of semantic segmentation [17].

Figure 8. Example of object detection [18].

3.2.1.4 LIDAR-based

Lidar, as one of the most important sensing technologies in autonomous driving, together with image recognition, stands for Light Detection and Ranging. One of the main differences with cameras is, however, its performance being independent from illumination conditions. Likewise, cameras are limited in their vision range as they are directional, unlike lidar, able to do 360 scans of the environment. Their performance is, however, very determined by weather conditions and the cleanliness of the sensor screen. In general, a robust system relies, in the fusion of both sensing technologies. Radar technology can also be used to obtain similar information, but lidar provides a considerably higher spatial resolution as the laser beam is more focused and able to scan a larger number of layers in the vertical direction. The measuring element is an infrared laser beam that is used to determine the distance between the sensor location and an object positioned within its range, which nowadays is reaching 300 m in the automotive industry, where objects can still be reliably detected. However, it needs to be mentioned that the quality of the data does get reduced with the distance, due to the spatial divergence between laser beams. Typically, lidar light wavelength range is positioned around 900nm, between 750 and 1000 nm, although higher wavelengths are now being used to perform better under bad weather conditions [19], [16].

The internal mechanism of a simple lidar, consisting of a rotating swivel, allows steering the laser beam to scan the field of view. The laser beam is not continuous, but a pulse of light that is reflected in the surrounding objects and received back by the sensor, which is able to accordingly generate a three-dimension point cloud. This cloud consists of many points distributed in a three-axis coordinate system, organized in layers. Each point, apart from providing information of its distance to the sensor, can also indicate the reflection intensity value, which is, the reflectivity and therefore the colour of the surface of the object [16], [19].

More specifically, the measurement method of lidar sensors to be able to generate a three dimensional point cloud is known as ToF (Time of Flight) method, where the main metric is the time it takes to the laser pulse to arrive back to the sensor, after being emitted and reflected on an object, relying in the constant propagation speed of light [16]. It is also able to register the return signal strength, which is a sign of the reflectivity of the laser contacting surface, road pavement marking can be recognised and transformed into electronic maps through vectorization algorithms [21].

Lidars can also be equipped with rotating hexagonal mirrors that enable avoiding having to mechanically rotate the laser beam itself, so that a 360 deg view can be obtained. These kinds of sensors are not able to directly measure velocity of the objects, unlike novel technologies such as coherent lidars, used for applications such as wind turbulence speed measurement [16] [19] [20].Despite its robustness and accuracy in object detection, which greatly contribute to autonomous driving safety in terms of surroundings recognition and decision-making , the biggest limitation of this technology is its considerable cost, which is moving in the direction of being reduced, as per the trend of the industry, together with the reduction of its dimensions [19], [16]. Likewise, their working principle provides them with active illumination, making them highly effective at night. Also, due to having detection ranges that are valid up to a few hundred meters, they offer a solution that fills a range gap between cameras and radars. LiDAR market has lately grown significantly, mainly due to its demand on the automotive sector. Since autonomous driving is becoming a reality, LiDAR became one of the most prominent solutions towards achieving complete awareness of the vehicle surroundings [30].

The following figure displays a typical sensor setup in a vehicle. Radar (1) for cross-traffic alerts, blind-spot assistance and (2) adaptative cruise control; LiDAR (3) is used to generate in real-time a high-confidence, 3D representation of the environment in the shape of a point cloud; and cameras that help in features such as (4) object detection and classification and (5) collision avoidance [30].

For the sake of simplicity, this work considers a LiDAR-only perception system, being the reason why understanding its possibilities and limitations is of great importance. Although LiDAR perception has already been introduced amongst autonomous driving perception approaches, this chapter will deep dive into some key concepts, together with current technical challenges, to provide the reader with an understanding of the main technology this work is built upon.

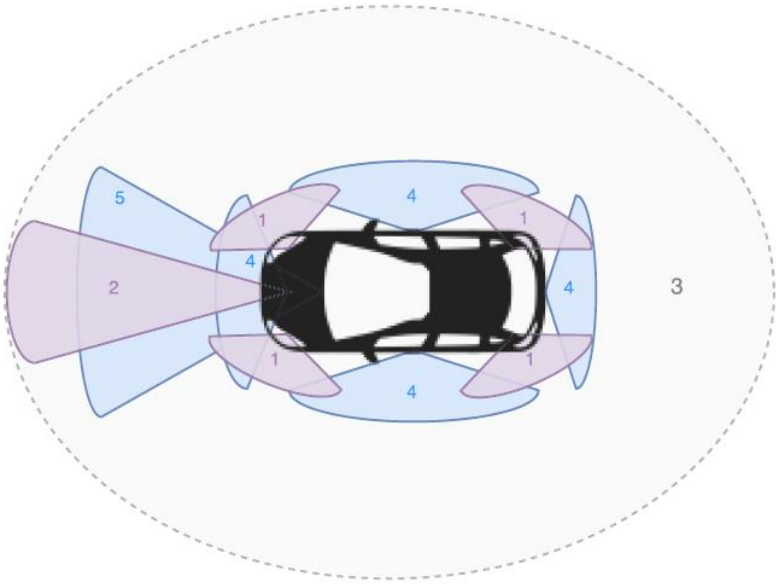

Figure 9. Autonomous driving sensor set-up [30].

Basic concepts

Essentially, the basic equation that describes the functioning of a LiDAR sensor dis the following:

$$R = \frac{1}{2} c\tau$$

It defines how the distance to a surrounding object (R) is obtained from the round-trip delay (time of flight, τ) of an emitted light signal, of speed c, by modulating its phase, frequency, and amplitude. This process has a result the construction of a point cloud that represents the vehicle surroundings in 3D. Regardless of the measurement technique for this ToF, some key metrics of the Lidar performance are below [30] [31].

Detection range refers to the maximum distance for an object to be detected by the sensor, which is measured under controlled conditions on materials with 80% of reflectivity, being the result mainly limited by the transmitted power of the device. This **transmitted power**, is likewise limited by human eye safety regulations, being also defined as the laser's maximum exposure. This value is calculated from certain characteristics of the laser such as wavelength, beam diameter, exposure duration, pulse width and repetition rate [30] [32].

Field of view (FoV) denotes the angle within the emission of laser beams is limited by the construction of the device itself. Some sensors can provide 360 degrees of the surrounding, horizontally by using rotating components. However, LiDARs vertical FoVs are always limited. Regarding the quality of the measurement, **precision** is defined as the repeatability of the laser measurement, **accuracy** refers to the deviation of a single measurement of the real, unknown value, and **resolution**, which is based on the smallest angular separation between two points that can be captures by the laser, defining the density of the point cloud [30].

Measuring techniques

Primarily, LiDAR measuring techniques get split between pulsed or continuous wave, having implications in robustness but also complexity and therefore cost. **Pulsed ToF** relies on accurate

timers to measure the round-trip time interval between an emitted and received laser pulse, this time is digitalized through time-to-digital converters, and they provide a measure of the distance from the sensor to the reflecting point. **Amplitude Modulated Continuous Wave** operates similarly but substituting sharp pulses by a signal modulated in intensity, where the distance is obtained by integrating sections of the signal. The method is in this way able to detect the difference in phase between emitted and received sinusoids by evaluating the result of the integrated windows of both. The following image clarifies the three approaches here depicted [30].

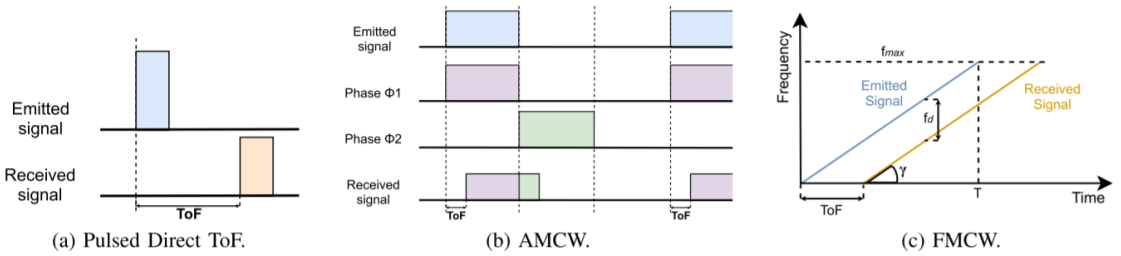

Figure 10. LiDAR measurement techniques [30].

Lastly, **Frequency Modulated Continuous Wave** LiDAR systems focus on the properties of the light to measure distances rather than modulating its intensity. In this case the frequency is linearly modulated synchronised with an up-chirp signal. Therefore, maximum measurable distance is limited by the signal period (ToF < T). The distance to the measured pint is in this case directly proportional to the frequency difference between the emitted and received signals, which can be measured constantly [30] [31].

Imaging systems

To generate the surrounding-representing point cloud, LiDAR sensors count with a variety of imaging techniques. As mentioned under autonomous driving introduction, these methods rely either on a rotating mechanism to steer the laser beam, or a solid-state system that do not require mechanical movement. Likewise, solid-state beam steering sensors reduce in size the required mechanism of original rotating systems. **Rotor-Based Mechanical LiDAR** have been used within automotive self-driving for a considerable time, providing 360 degrees FoV and a variety of vertical angle ranges. This FoV is obtained through a mechanical rotation of the scanning section of the sensor. **Scanning Solid-State** ones are limited to a restricted FoV, as they lack a rotating mechanism, which contributes to cost reduction. This FoV can however be widened by installing several of these sensors around the vehicle. In this case, FoV ranges from 25 to 150 degrees. Finally, **Full Solid-State** sensors completely lack mechanical rotating parts. They are also named Flash sensors as their working principle is like digital cameras, illuminating the environment with a flashlight, measuring with photodetectors arrays the back-scattered light. [30].

Technology challenges

Further than the different constructive solutions mentioned above, the use of LiDAR sensors implies several other aspects the industry is still aiming to resolve, prior to a robust implementation of a sensing solution for autonomous driving. These mainly have to do with data post-processing towards appropriate object recognition. One of the main struggles of LiDAR sensors is achieving an effective **calibration**. Intrinsic calibration aims to resolve

uncertainties proceeding from the tolerances of their internal components and their manufacturing process itself, by adjusting the measurement to planar targets of accurately known dimensions, in a controlled environment. On the other hand, extrinsic calibrations are also needed and refer to the installation in the actual vehicle, combining different sensor outputs within different setups [30].

Another problematic current technology is aiming to cope with is the **mutual interference** of LiDAR laser beams, especially when augmenting the number of vehicles that rely on LiDAR in the same area. Laser emissions generated by a vehicle will likely be received by another vehicle's receptors, generating noise in the resulting point cloud [30]. Likewise, real-time point cloud generation, handling, storage, and transmission continues being one of the main limitations with regards to the need of **data compression**, as the generated output of the complete representation of the surroundings can reach several gigabits per second [30]. As priorly mentioned, adverse weather conditions also pose a considerable problem over LiDAR sensing. Rain, fog, or snow can severely affect the quality of the obtained point cloud. In this case, several **weather denoising** techniques have been proposed to mitigate noise generation. [30].

Decision-making in AV

Autonomous driving requires, apart from accurate and redundant perception of the environment, robust decision-making algorithms capable of handling complex, dynamic situations. An example being an urban area with an intersection and multiple vehicles of different sizes, pedestrians and traffic lights. Mainly, the technical challenges that decision-making algorithms need to overcome are, firstly, coping with the noise of the information provided by the perception systems, which can be substantial especially when performing under inconvenient environmental conditions such as bad weather or poor illumination; being able to predict the behaviour of the motion of its surroundings to be able to establish efficient trajectories, which is something the human driver is normally able to intuitively discern; and being able to ensure the avoidance of collisions, meeting the vehicle kinematic and dynamic constraints and following established traffic rules. The following Figure represents a typical situation in which needs to decide which action to take under the uncertainty of the immediate trajectory of a surrounding vehicle [22].

Depending on the scope, the decision-making task of an autonomous vehicle is normally divided into strategic, tactical, and operational decision-making, which can also be identified as navigation, guidance, and stabilization. The higher level, strategic, copes with long term decision-making such as route optimization (e.g., depending on traffic level). Tactical level represents an in-between situation where decisions are discrete, such as deciding whether to change lanes. Operational decisions are, finally, more linked to the control of the vehicle under certain situations, such as emergency stops, avoiding obstacles, etc. Normally, AVs implement the three typologies, within which numerous algorithms for different tasks are programmed into the vehicle control units [23].

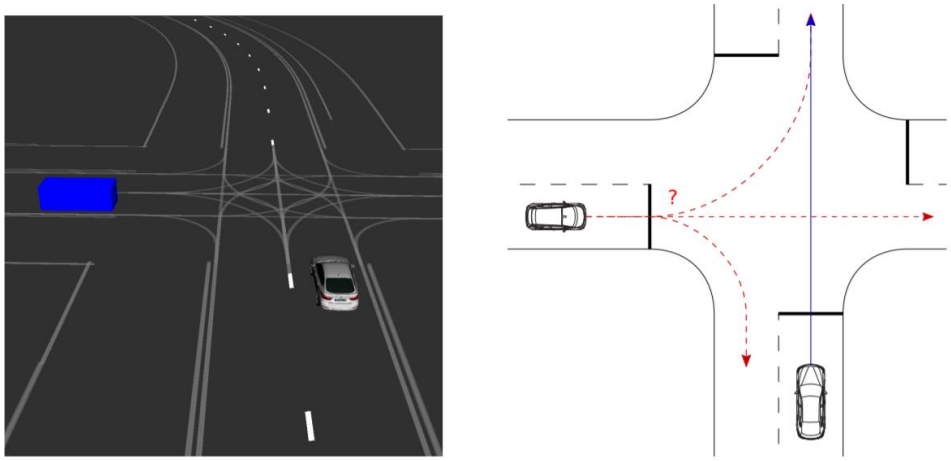

Figure 11. Situation where predictive decision-making of the ego vehicle (blue) would enhance AV operation [22].

Likewise, depending on the methodology behind the construction of the algorithm, the field presents two main streams. On one hand, traditional decision-making is based on a deterministic structure, where most decisions follow a sequence of logic gates, having as an input a set of situations that are assumed to be known, or having done a considerable number of assumptions and simplifications. These algorithms normally make the vehicle behave in a reactive way, having the advantage of its simplicity and therefore robustness, under no too complex situations. However, in more dynamic and complex environments, such as the priorly-mentioned intersection, reactive approaches are not as effective as predictive. In such cases, a vehicle driven by reactive models can easily end up in a situation where all possible actions become unacceptable, and therefore either freezing in a complete stop or following highly evasive routes with are often not optimal and potentially dangerous [24].

Predictive decision-making, on the other hand, is normally constructed on a stochastic structure, where decisions are made based on probabilistic estimation of the next movements of surrounding objects. Although some of these algorithms only output the decisions made, normally, also a measure of the uncertainty of the agent making the decision should be provided [23]. The decisions are normally triggered using fuzzy thresholds and the probability values behind the predictions are based on pre-trained machine learning techniques. This second approach is, by a considerable margin, representing the main direction in current autonomous driving decision-making development, due to latest advances in artificial intelligence that are related to increased computational power of today's hardware. AI-based systems highly enhance decision-making flexibility in complex environments, solving one of the major challenges of AD, having as a disadvantage a performance that is highly impacted by the quality and quantity of the algorithm training datasets, which has a direct impact on their robustness.

Not only current decision-making relies on machine learning techniques, but also other systems like vehicle perception, where semantic segmentation and object tracking is normally performed using pre-trained neuronal networks. Very briefly, some of the most used methods within machine-learning in decision-making are Deep Reinforcement Learning and Tree Search algorithms [24]:

- Deep Reinforcement Learning combined reinforcement learning (usually stated as RL) and deep learning. RL is inspired by behavioural psychology. The main idea behind it is an artificial agent that may learn by interacting with its environment similarly to a biological one, learning to make decisions and optimize objectives by trial and error through a system of cumulative rewards. The reinforcement itself is defined as the task

of learning how the agents ought to take sequences of actions in an environment to maximize these rewards. RL has in the last years become increasingly popular due to its proven success in addressing challenging sequential decision-making problems. Likewise, deep learning mainly consists of a neuronal network (NN) characterized by a succession of multiple processing layer. Each layer represents a non-linear transformation by a parametric function and the sequence of these transformations leads to learning different levels of abstraction. The transformation layers between input and output are defined as "hidden", and in the simplest case, at every iteration, the algorithm changes its internal parameters to fit the desired function. The following Figure shows a simple scheme of a single-hidden-layer network. Within one given NN, an arbitrarily large number of hidden layers is possible, being the current trend to have an ever-growing number of them, with more than 100 in some supervised tasks. The combination of both, defined as deep RL, results useful in problems that are not sequential but high dimensional state-space. Simple RL approaches presented design issues in the choice of features. Deep RL has been successful in complicated tasks with lower prior knowledge thanks to its ability of learning different levels of abstraction from the data. For example, a deep RL agent can successfully learn from visual inputs such as images, composed of thousands of pixels, opening the possibility of mimicking human problem solving capabilities, which was difficult to conceive some years ago [25].

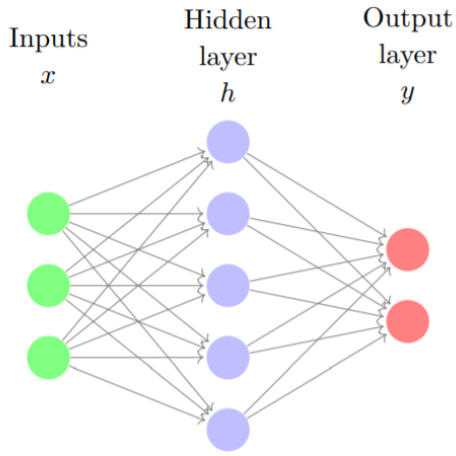

Figure 12. Example of single-hidden-layer NN structure [25].

Some popular deep RL algorithms used in AV decision-making are Policy Gradients, based on using a parametric representation of the logic behind the rewarding system; Deep Q-Network, which approximates the Q matrix of values computed with a NN; Actor-Critic, which is a temporal-difference method with a separate memory structure to explicitly represent the reward policy independently from the value function; Proximal Policy Optimization, that improves the stability off the actor training by limiting the policy update at each training step; Trust Region Policy Optimization, where the policy gradient computes to the steepest ascent direction for the rewards and updates the policy in that direction; and Imagination-Augmented Agent, having the idea of allowing the agent to imagine future trajectories and incorporating them into the decision process [24].

- Tree Search Algorithms are often the solution for planning problems, by simulating ahead into the future, evaluating future states and backing-up those evaluations to the root of the search tree. Among TSA algorithms, Monte Carlo Tree Search is one of the most powerful and widely used, consisting of several phases. First, it simulates the trajectories into the future starting from the root state. It then evaluates the performance of the "leaf" states, either using a random rollout or an evaluation function such as a value network. Finally, it backs-up these evaluations to update the internal values along the trajectory by, for example, averaging over evaluations. By incorporating a NN, the model gets renamed as MCTSnet, adding the simulation-based search into a the NN, expanding, evaluating and backing-up a vector embedding. The parameters of the NN are again trained end to end using a gradient-based optimization. The key idea is to represent the internal state of the search, at each node, by a memory vector. The computation of the network moves forward from the "root" state, like a simulation of the original method, implementing a simulation policy based on the memory vector to select the trajectory to travel [24]. The following figure represents the basic structure of the Tree Search method:

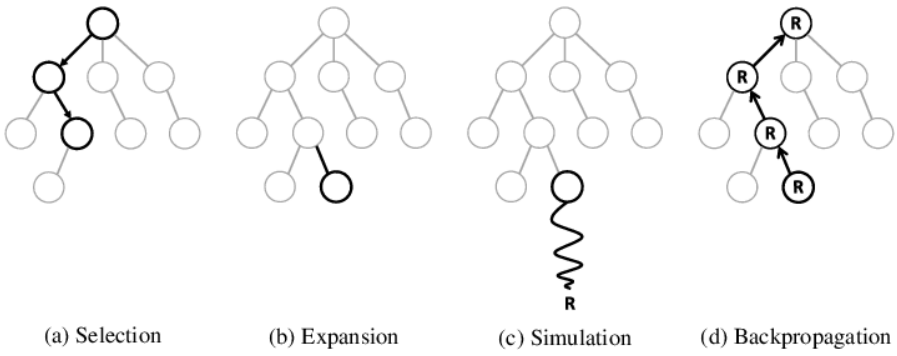

Figure 13. Phases of the Monte Carlo Search Tree algorithm [26].

The decision-making algorithm for lane changing described in this work, which will be later exposed, is deterministic and involves a reactive vehicle behaviour. It was conceived that way to, firstly, fit under the scope of a master thesis, as an AI-based algorithm would require a considerable amount of work before it is refined and trained enough to show consistent results, and, secondly, as the environment it was designed for does not involve, in first instance, a high level of complexity, as it will be later depicted, so a deterministic algorithm should perform correctly for the defined task.

Control systems in AV

Given that vehicle positioning, surrounding perception and subsequent path planning (high and operational level) are solved, the remaining task for autonomous driving is keeping the vehicle within the programmed path at the programmed speed profile. This path becomes the main input for a control system, known as a drive-by-wire interface, together with a dynamic model of the vehicle, which changes depending on the vehicle morphology. Withing the control task, it is normally split into lateral and longitudinal control, having as an output actuation signals on steering wheel, brake, and throttle. In this section, vehicle stability systems, like ABS or ESP, which are part of the vehicle control field, will not be treated as they are not considered intrinsic to the autonomous driving task.

- **Vehicle modelling:** Designing a vehicle controller requires a mathematical model of the vehicle's behaviour. Two main approaches exist towards the generation of the model: dynamics and kinematics. Normally, a dynamic model provides a highly accurate picture of the vehicle behaviour, being the controllers designed with this approach sufficiently robust with those dynamics. The system is typically dynamically simulated using a three-dimensional body (vehicle) with 6 degrees of freedom (DoF), being these DoFs the number of movements of the vehicle. These are three translational variables (x,y,z) and three rotational (pitch, yaw, roll). Besides that, there are several factors to be included in the model, such as vehicle's weight, centre of gravity, cornering stiffness, wheel slippage, etc. The vehicle dynamics usually respond to theoretical differential equations and the parameters within those equations. The next figure (left) shows the notation used in a simplified physical system corresponding to a four-wheel car [27]:

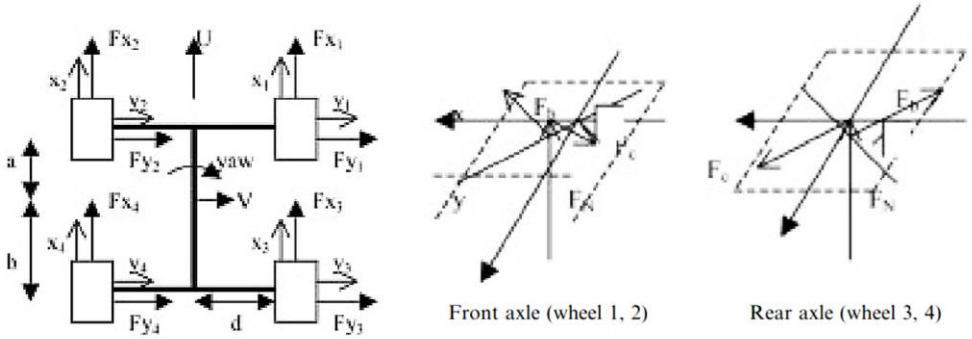

Figure 14. Basic vehicle notation (left). Forces on tire: Normal, Fn, lateral, Fc and break, Fb (right) [27].

Likewise, as a basis of the vehicle model, a mathematical model of the interface to the road is needed, which is the tire model. In this case, the parameters of the model are measured in a physical tire, making the model mayorly empirical, making it possible to accurately determine lateral as well as longitudinal forces, together with the wheel aligning moment. The lateral force if a function of the slip angle, the normal force, and the camber angle. The longitudinal one is, on the other hand, the opposite of lateral force, in the model established by Pacejka [27].

- **Vehicle controller:** The methodology of the controller can be typically divided in several ways: by control input method (two-wheel steering, four-wheel steering, or direct yaw control); by controller design method (optimal, neuronal network, input scaling) or by controller implementation structure (feed-forward, feedback, or a combination of both). Within the several controllers included in an autonomous vehicle, the main ones are the lateral controller and the longitudinal controller. A feedback controller presents some advantages over a purely feed-forward control, being the most important one the fact that a feedback controller maintains stable vehicle behaviour during changes in driving conditions. Within feedback controllers, there is proportional feedback and state feedback. The proportional feedback controller is designed by taking the output signal from the control subject, multiplying it with a suitable constant and using it as an input signal for the steering wheel angle, for example.

  A lateral controller or steering controller is the main controller of the system, as the main input is normally based on the output of the vehicle path creation process. It is designed to follow the desired path, without having influence on its determination, which is responsibility of a higher-level planner that prioritizes avoiding collisions or

arriving at the ultimate destination. The controller, in this case, only requires information of the vehicle location relatively to the path. The error between the path and the vehicle is calculated and a microcomputer that sets the correcting steering wheel angle. These models can, however, easily become highly complex as body motion in interaction with the wheel suspension must be considered during cornering, transient steering, and directional stability, being the behaviour of tires non-linear [27].

A longitudinal controller does not depend on the look-ahead distance or angle difference between road tangent and vehicle orientation, as a lateral controller does. Instead, it just depends on the longitudinal dynamics of the vehicle, when driving straight ahead or with small lateral acceleration values, assuming equal rolling resistance coefficient for all wheels. Normally, the longitudinal controller has as an input the desired acceleration profile along the path, achieved by acting on throttle and break [27].

3.3 State of the art of autonomous lane-changing algorithms

One of the fundamental leaps from level-1 to level-2 and above AVs is lane-changing function. Lane-changing research has attracted much attention in recent years [33].The act of changing lanes is already highly complex during manual car operation. Recognizing the time window to overtake a vehicle, at the right speed, in dense traffic environments, has considerable safety implications, especially in high-speed lanes such as a highway. Therefore, is has become one of the focuses of autonomous driving study. Even though several automotive brands have recently started to offer lane-changing assistance to the driver or even full lane change automation, the matter is still being developed towards delivering more robust solutions.

Within the topic of lane changing, one can differentiate efforts being carried out in three main aspects. One of them is the understanding of human lane change behaviour to a point that it can be mathematically modelled. This has become a reality partially due to the increase of computing power that leads to the effective implementation of AI, obtaining model that are able to describe microscopic lane-changing behaviour. These models, together with car-following ones, are the basis of simulation tools that are used for the development and validation of autonomous driving systems. Likewise, they allow certain real-time prediction of surrounding vehicle in real scenarios, increasing robustness and efficiency of the AV decision-making. In [34] and [35], a review of different approaches towards this modelling can be seen. These, however, will not be exposed here, as it lays outside the scope of this work.

Likewise, besides the aspect of lane-changing here studied, lane-changing trajectory planning is the other main area. Path planning in this case is closer to the motion control of the vehicle when changing lanes, rather than to the decision making. The lane-changing planning is however not a trivial topic and has been studied in multiple publications, proposing several solutions. In some cases, such as the proposal exposed in [36], the planning activity does involve some intrinsic decision-making for dynamic trajectory correction, as the initial one is calculated before the ego vehicle attempts to overtake, and environment conditions are likely to change through the lane-changing process.

Lane-changing decision making is the other major field, being the focus of this thesis. Specifically, high level decision-making continues to be an important challenge due to the already mentioned complexity of dynamic environments surrounding the ego vehicle. As a way of coping with this complexity, the above-mentioned path planning and manoeuvre execution has in several cases been disassociated from the high-level decision-making, so that they can be treated independently. In this case, the decision-making layer plans the high-level approach, such as deciding whether to perform an overtake or stay in the lane. This decision-making involves setting up a structure of rules and after deducing the optimal solution in every

situation. However, traditional approaches have the disadvantage of not being able to cope with situations that where not conceived during the development of the rule-based system. If it was the case, new rules would have to be added to the algorithm. If this situation occurs repeatedly, there is a considerable risk of ending up with a decision making process of extreme complexity, during its operation, and reduced comprehensibility when being developed or modified [37], [38].

One of the fundamental elements within high-level lane changing algorithms is the selection of an adequate safety distance with the vehicle in front, this is normally calculated dynamically, considering vehicles speeds and typology, AV dynamic capabilities and taking as a baseline human average reaction time. Likewise, to be able to ensure lane-changing safety, latest algorithm aim to predict lane changing actions of other vehicles. A lane change from a rear vehicle into the fast lane at the same moment as the AV, as well as a vehicle already situated in the fast lane overtaking the AV and positioning directly in front, could severely compromise the obstacle-free area. In this scenario, sudden braking or even collision might occur, being a situation reactive approaches might not be robust against [39].

To be able to perform such prediction, enabling AV's safe interaction with human-driven vehicles, the biggest effort is currently done in characterizing human driving-related decisions when interacting with other road users. The models arising are either data-driven or motivational. Both approaches present certain flaws; while data-driven ones normally lack explainability, motivational ones fail in generating testable predictions [37].

Besides human-driven vehicles lane changing predictions, in general, increased computing power and available data have enabled machine learning techniques also in the field of lane changing decision-making. Although there are different methods being applied, by a big margin, reinforcement learning (RL) has been extensively explored and already provided promising results. RL-based algorithms are currently overtaking traditional rule-based ones in terms of collision-free, efficient lane-changing results, while requiring less human [38].

Game-theory is an alternative approach being studied. It aims to overcome the fact that AI techniques still treat the vehicle as a reactive entity, even though using prediction of obstacle movement, neglecting the decision-making process of surrounding vehicles. Game theory is a rather general model, it represents a field of mathematics that studies optimal decision-making of multi-agent systems. Within a game theory mathematical framework, the best possible response of the agents, given a certain state, is obtained, when having either cooperative or conflicting objectives [40].

These subjects are depicted below by exposing related works within lane-changing decision-making:

Behaviour prediction in lane changing

Elaborating on the above-mentioned approach. Weida Wang et. al. [39] propose a decision-making strategy that is based on an intelligent lane-changing behaviour prediction. Their work starts by using a fuzzy interference system (FIS) to integrate different driving environments with the cognitive process of the drivers. The fuzzy logic is intended to emulate human cognition and lane-changing feasibility is extracted from the environment characteristics. After that, a neuronal network described as long, short-term memory (LSTM) is programmed and given as inputs the lane-changing feasibility computation and the vehicle trajectory. This NN then elaborates agent's lane-changing predictions and, finally, based on those predictions, a decision-making strategy is designed, in this case, for path planning of the AV [39].

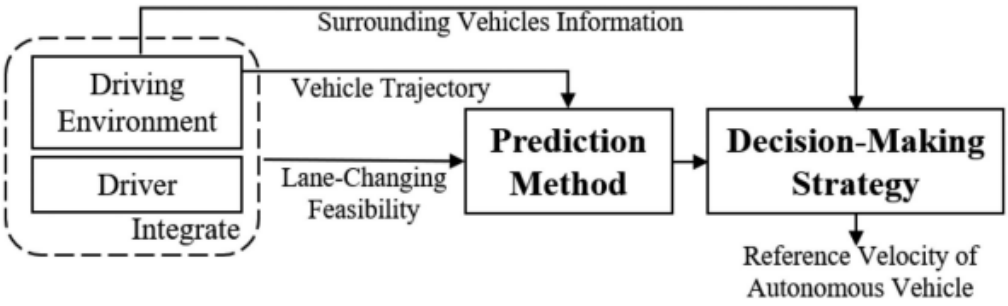

Figure 15. General approach of Weida Wang et. al. work [39].

Focusing on the decision-making part of the work, the algorithm is not intended to perform lane-changes of the AV, but to ensure safety distance ahead of the vehicle and comfort of the passengers by adjusting AV's speed (calculation of reference speed) based on surrounding vehicles behaviour prediction. The algorithm functions as a "Finite state machine", by making decisions depending on the definition of four different discrete states that describe the driving situation: **Cruising,** If the distance between the AV and the preceding vehicle (1) is bigger than the cruising distance. **Following,** If the preceding vehicle is within the following distance range (2), which is defined as the difference between cruising distance and safety distance. **Real-time avoidance (RTA),** if a vehicle is detected to enter the safety distance of the AV (3). **Avoidance-in-advance (AIA),** If the prediction result of the NN shows that a vehicle will enter the safety distance (4). The four states are shown in the following figure [39].

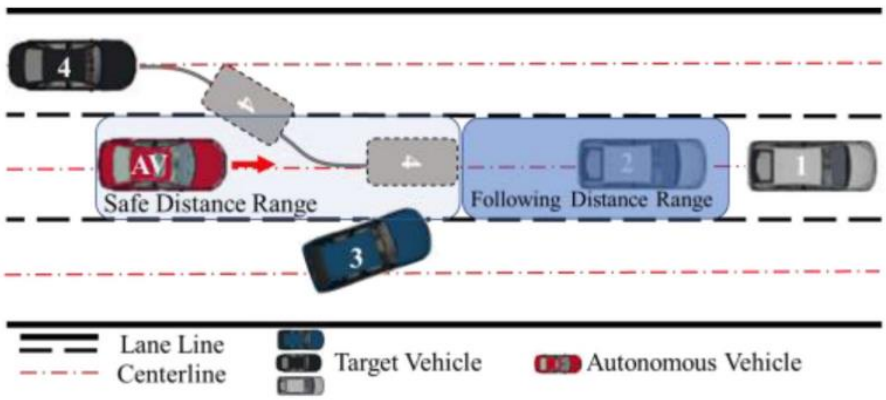

Figure 16. Graphic description of states [39].

After these states are defined, as reference speed for the ego vehicle to ensure safety distance is calculated. In Cruising state, the speed is simply given by the desired cruising velocity. In following state, the speed of the AV will adjust to the speed of the vehicle ahead, avoiding entering the reference following distance. This distance is dynamically calculated, considering the magnitude of the speed [39].

The innovative part is the calculation of the reference speed in RTA and AIA states. In RTA, where obstacles are not predicted but suddenly detected within the safety distance, the AV will apply the emergency brake with the maximum deceleration, being the reference velocity described as [39]:

$$v_{AV,ref}^{t} = v_{0,RTA} - \int_{t_0}^{t} a_{dmax}(t)dt \quad [39]$$

Where $v_{0,RTA}$ is the initial velocity, t_0 the time when the state becomes RTA and a_{dmax} is the maximum deceleration allowed by wheel grip without locking. In AIA state, where the vehicle entering the safety distance is known (predicted), this safety distance is adjusted in advance. In this case, the motion of the AV is planned with uniform deceleration having as a target the reference velocity that in this case is calculated as [39]:

$$v_{AV,ref}^{t} = v_{TV}^{t} - \frac{2(D_F - D_t)}{t_p - (t - t_{0,AIA})} \quad [39]$$

Where $t_{0,AIA}$ is the time the vehicle becomes AIA state, v_{TV}^{t} is the velocity of the target vehicle at t, D_F is the reference following distance and D_t nis the distance between the AV and the vehicle ahead at t [39].

As a result, the prediction NN was trained and tested using data prevenient from the NGSIM dataset for a number of simulations. The accuracy reached on the predictions reached 92,40%. The algorithm that integrates these velocity profiles was tested using a hardware-in-the-loop system (HIL), where the strategy was compared to human driver data. The maintenance of the safe distance was there validated, and results showed that the prediction of intrusion enabled a velocity and acceleration change more optimized than in human driving cases, theoretically. As a drawback, the system is still to be tested after implementation in an actual vehicle controller [39].

The use of RL in lane changing

A good example of the application of RL in high-level decision making for lane changing is the work of Branka Michevska et. al. [38]. In this case, a reinforcement learning-based approach is chosen, combined with formal safety verification, to neglect possible unsafe actions. This method aims to cope with the lack of safety and computation speed of other machine learning techniques. In this case deep reinforcement learning agent gets trained to drive as close as possible to a desired speed by executing reasonable lane changes (from a high-level perspective) in a simulated road with an arbitrary number of lanes. Fast learning rates are achieved by using a minimal state representation of the environment, to reduce the dimensionality of the state space (13 continuous features) and a Deep Q-Network for the reinforcement learning (deep Q-Network was one of the first methods that successfully brought the perception power of a convolutional NN to the RL problem) [41], [38].

The RL problem is this case is defined as a Markov Decision Process. A set of S states is defined for the environment, together with a set of actions, A, a probabilistic transition function, p, between states, describing the system behaviour, and an immediate reward function, where R is the set of rewards. The aim, as usual in RL, is to find an optimal policy that maximizes the expected cumulative rewards for each state. At each time step t, the RL agent receives information about the state of the environment St and selects an action A(St), receiving a reward a timestep after, Rt [38]. The idea is depicted below:

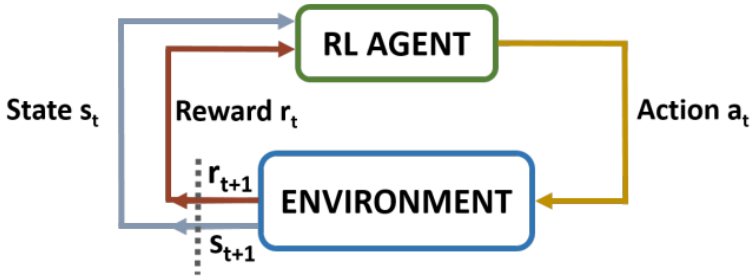

Figure 17. Working principle of the reward system within the RL definition [38].

The defined states in this case are, considering an AV that only observes vehicle directly in front, directly in the rear and on both adjacent lanes, are a resultant vector composed of the above-mentioned 13 variables. These variables are the absolute velocity of the AV, the relative distance to leading or following surrounding vehicles and their relative velocities [38].

$$s = [dr_{ll}, vr_{ll}, dr_{ol}, vr_{ol}, dr_{rl}, vr_{rl}, v_{RL}, dr_{lf}, vr_{lf}, dr_{of}, vr_{of}, dr_{rf}, vr_{rf}] \quad [38]$$

The defined actions, taken depending on the values of s, are (a1) perform a lane change to the left, (a2) keep the lane and (a3) perform a lane change to the right, as the decisions are intended to be kept at a high level. The action is checked by a posterior safety system. If confirmed, the change action is taken as a discrete movement that takes an estimation of 3.5 seconds. In this case, the author affirms that discrete actions tend to perform better in AV than end-to-end systems, which use low-level controls to execute the lane changes. The reward function is designed to maximize the velocity of the vehicle, where $v_{des,t}$ is the desired speed of the RL agent and $v_{RL,t}$ its absolute speed [38].

$$r_t = -|v_{RL,t} - v_{des,t}| \quad [38]$$

The safety verification, as machine learning techniques cannot ensure safety, is formulated in a formal way, taking as a baseline the traditional definition of safety distance, by deriving it to account for the dynamics of the agent and the other traffic participant. This definition provides the calculation of the AV safe free space *Ft,* so that an action can only be taken if the AV is within the limits of *Ft.* The concept is shown in the image below [38].

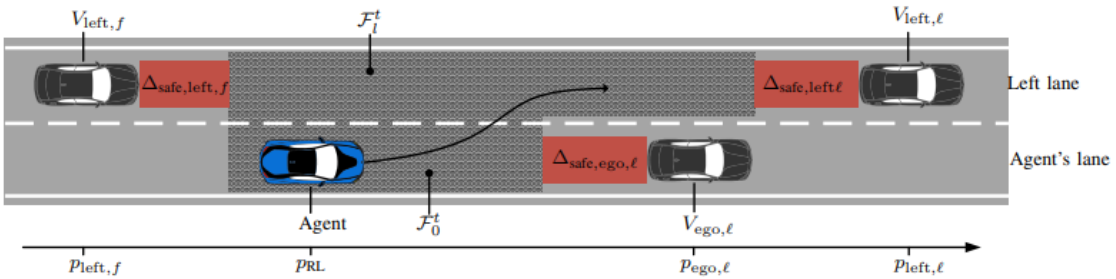

Figure 18. Graphic representation of the safe free space definition [38].

Once the optimized policy is achieved through a data collection and learning phase, it can be employed in the decision-making process, which goes from observing a new state to choosing a safe action. The input of s composed of the 13 variables is input into a 2 hidden layers Q-network that outputs the estimates for Q(s,left), Q(s,keep), and Q(s, right). The action with a higher Q value is forwarded for safety verification and executed if considered safe. If not, the second higher valued action is passed to the safety filter. Ultimately, the AV stays on the lane as Q(s,keep) is always a safe action [38]. A graphic representation of the decision-making process is shown below:

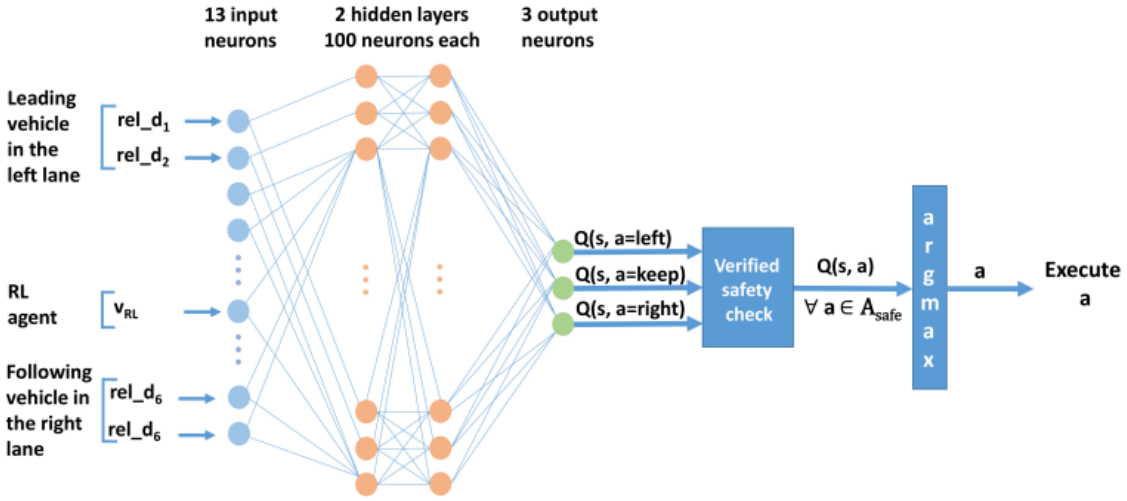

Figure 19. Decision-making algorithm representation, based on a deep Q Network and formal safety verification [38].

For validation purposes, 10 simulated traffic scenarios were created. Each of them having a duration of 500.5 seconds, forcing a number of 143 decisions, one each 3.5, as priorly defined. The RL agent was compared with a rule-based agent used for benchmarking and the RL-based agent outperformed it besides the complexity of the deterministic alternative. During the simulation phase, no collisions were caused by the RL agent, while achieving velocities close to the desired by the agent [38].

However, despite its simplicity and promising results, the decision-making algorithm developed in this work is lacking, as usual in this kind of works, validation on the field. Even though the network training and the simulation scenarios are based on real data, its application in a real scenario might differ from the obtained results as, (1) traffic behaviour would be influenced by the presence of the AV and, (2) a discretization of a change performed on a constant time of 3.5s would be challenged and needing considerable algorithm adjustment [38].

The use of Game Theory in lane changing

The paper elaborated in [40] represents a good example of the use of Game Theory in the field. It addresses the problem of decision making for lane changing by formulating multiple games in normal form for pairs of agents. The outcome is the generation of the optimal action decision for the AV, without having under consideration the global optimality of all agents. In this case, likewise, the author makes use of the aid of deep-reinforcement learning, which, integrated with the game-theoretical formulation, is regarded as Nash Q-learning algorithm [40].

In a general normal-form game, each of a set of N players selects an action to carry out. Depending on the actions executed, a payoff is assigned to each player. Formally, a game in

normal form is expressed as a tuple (N,A,J), there N is the set of players, A the set of actions available to the players and J is the payoff function that establishes the payoff mechanism. The differentiating element with a simple reward method, as RL, is that the objective of a player in the game is to obtain the highest possible payoff by selecting appropriate actions, while keeping under consideration that all the other players will also try to maximize their gains. These games are normally expressed as a table, as it is shown below. In this case A and B are the two players, having each two different actions to perform. U or D for player A and L or R for player B. The table shows the payoffs aij and bij that players A and B, respectively, can receive for a particular combination of actions. If wanted to increase the number of possible actions, the table would grow in number of rows or columns. An increase of players, on the other hand, would mean the table becoming a multi-dimensional array. Nash equilibrium is said to be achieved in the game if all players play their best response against each other, being this the most important solution concept in game theory. It represents a stable solution as, when reached, no player can unilaterally change its decision without negatively affecting their own payoff. In normal-form games, it has been proven that there always exists a Nash solution given by mixed strategies, where the optimal behaviour of an agent is to select the action with a given probability [40].

		B	
		L	R
A	U	a_{11}, b_{11}	a_{12}, b_{12}
	D	a_{21}, b_{21}	a_{22}, b_{22}

Figure 20. Typical game-theoretical structure for two players and two actions [40].

When applied to the lane-changing problem for highway autonomous driving, it can be formulated as a normal-form game. In this case, the ego vehicle travels on a multilane highway shared with other agents, noted as left-back (LB), right-rear (LR), left-front, etc. The state variables in this traffic scenario are defined by the distance and the speed of each agent respectively to the ego vehicle, in the form of a vector that describes each of them. Vehicle goals are to keep a safe distance from all other vehicles and to maintain a desired speed. The ego vehicle (EV), to be able to achieve the goals, must have into account that the rest of the vehicle will also try to optimize their own performance. In a case of 5 vehicles surrounding the ego one, in a three-lane highway, disregarding the one directly in the back, the three vehicle positions in front of the EV are not considered as relevant as their unlikely to react to the needs of the vehicles in the back to change lanes. Therefore, the set of players is defined as N = [E, LB, RB]. In this case, the EV ha only three different actions, which are keeping the lane, changing to the left and changing to the right. The vehicle on the left has the possible actions of keeping the speed (which has the same effect on the EV that accelerating) or reducing the speed. The vehicle on the right would have the same actions as the one on the left [40].

Regarding payoff function in the highway problem, although the game should be designed so that an optimal solution for the eV is reached for every given state, in a highway-driving game states change so often and suddenly that decisions might become unsafe a time period after, due to irrational moves of surrounding vehicles. The EV in this case is allowed to play the game repeatedly every time period so that an action can be reconsidered according to latest surrounding data. This implies that the decision-making game process must be computationally inexpensive, so that it can be quickly executed within the time step. An output of the controlled

system is the time to collision (TTC), between the ego vehicle and any other vehicle, being dependent of the state definition of the agents [40]:

$$T_c^{RF} = \frac{d_{RF}}{v_e - v_{RF}} \quad ; \quad T_c^{LB} = \frac{d_{LB}}{v_{LB} - v_E}$$

Additionally, T_F^E represents the time it takes the EV to complete a lane change. This variable implicitly accounts for the lateral distance the EV must travel to safely change the lane. This way, it can consider both longitudinal and lateral distances before deciding. T_{OE}^{LB} and T_{OE}^{RB} represent the time to overtake the EV by the LB and RB vehicles. ITTC, finally, stands for imminent time to collision, defined by the value TTC would have if travelling at desired speed (T_{IC}^{i} would be the variable representing this). The method of determining the expressions is through the establishment of a set of rules that describe the expected behaviour of each vehicle at a given state, so that the game payoffs can drive the agents towards these kinds of decisions. As an outcome, the expressions of the payoff values are [40]:

$$a_{12} = \theta_1(T_{IC}^{MF} - \theta_2 T_C^{LF}) + \theta_3(\theta_4 T_F^E - T_C^{LB})$$
$$a_{21} = a_{22} = \theta_5 - \theta_6 T_{IC}^{MF}$$
$$b_{21} = T_{OE}^{LB} - \theta_7$$
$$b_{12} = b_{22} = T_C^{LB} - \theta_8$$

Where θ are constants that's which values are obtained through artificial intelligence techniques. The NN output layer compares the payoff to solve the game and determine the bests actions for each vehicle. The weights of the links between the hidden layer and the output layer are static, so that the same operations are always performed. The first layer of the network computes the payoffs a_{ij} and b_{ij} and the output layer compares these values to determine the Nash solution of the game. The structure of the network is shown below [40]:

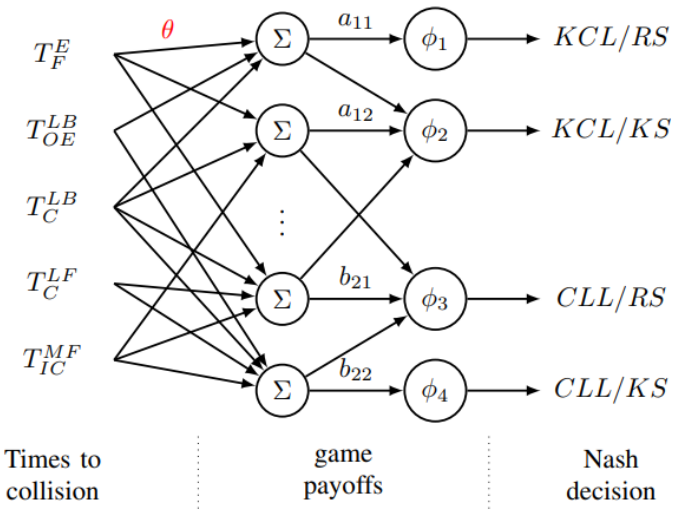

Figure 21. Structure of the network responsible for payoff parameter obtention [40].

After a training phase allows obtaining these coefficients and therefore calculate the payoff values, the proposed decision-making procedure is shown below. The game formulation plays the role of an actor, indicating the EV the optimal decision for each output vector y. during this learning phase the actor is updated with a "critic" function that uses the state information of the vehicles to compute the corresponding rewards. This critic corresponds to both the θ parameters and a Nash Q-learning algorithm that is not exposed here [40]:

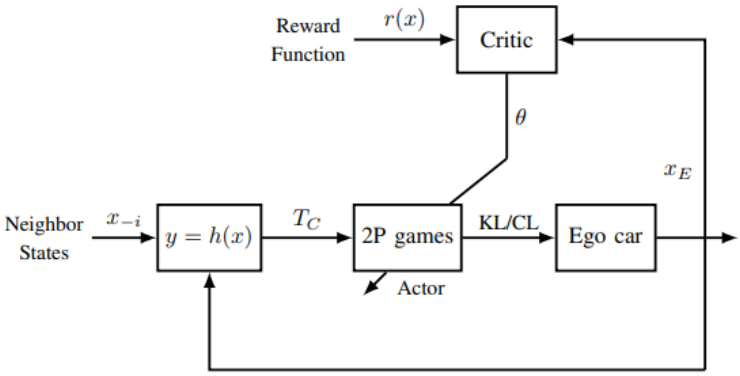

Figure 22. Game-theory-based decision-making structure [40].

The applicability of the proposed method in lane changing is tested through simulation. The vehicle dynamics are obtained using a bicycle dynamic model, where X and Y are the longitudinal and lateral positions. The EV travels in a highway section surrounded by other vehicles, controlling the longitudinal velocity and the lateral position of the EV. A desired velocity of 20 m/s remains constant, while the reference velocity varies according to the traffic circumstances. This reference velocity is the target velocity for the controller. If there is no slower vehicle in front, the reference velocity would be the desired one, if there were, it would become the front vehicle's velocity. The reference for the EV's lateral position is provided by the solution of the game-theoretic algorithm above. The safe distance the EV desires to maintain is set in 40m. Different lane-changing circumstances are simulated. A deep Q-learning algorithm, without a game formulation, is compared to the performance of the Nash Q-learning one developed in this work, using the same simulation settings. In a specific situation, the EV is forced to react to a double change of speed of the LB vehicle. It will maintain a certain speed, then slow down, and then speed up again. IT is expected that the EV initiates the change when LB lowers down speed, going back to the original track when capturing that LB is increasing speed again, eliminating the possibility of a safe change. In the case of applying game theory, the payoff table at that instant forces the EV to continue in the original track, instead of instantly switching lanes, responding to the Nash solution obtained, by considering the goals of LB [40]:

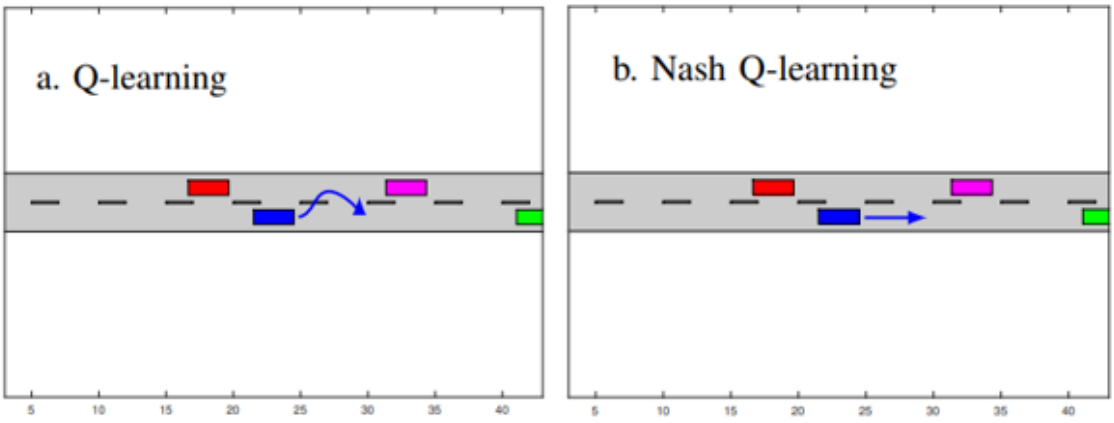

Figure 23. Graphic representation of simulation results for traditional Q-learing vs Nash Q-learing algorithms [40].

This work implements a dynamic behaviour model to the decision-making process, as a positive point compared to the prior example, where vehicle motion was heavily discretized. Likewise, it shows an interesting model to have under consideration the interests of the surrounding agents directly built in the EV decisions [40]. However, scenarios where the algorithm is simulated are heavily simplified. Apart from real traffic implementation, complementing current simulation scenario with more complex traffic conditions and action availability for the players is needed.

Cooperative lane changing

Another approach towards the lane-changing problem is the assumption of a net of interconnected Avs, as one of the intelligent transportation system models priorly exposed. Within this topic, [42] proposes a decentralized cooperative lane-changing decision-making framework for connected autonomous vehicles. The work is divided into the elaboration of three different modules: State prediction, candidate selection and coordination, where each vehicle makes lane-changing decisions independently, in a cooperative way. Each vehicle is assumed to be a collaborative automated vehicle (CAV). The environment considered is a typical three lane highway extending to the theoretical vehicle communication range, meaning that the study is still focused on the decision-making process of a determined ego vehicle, positioned in the centre of its range. The following figure depicts the idea [42]:

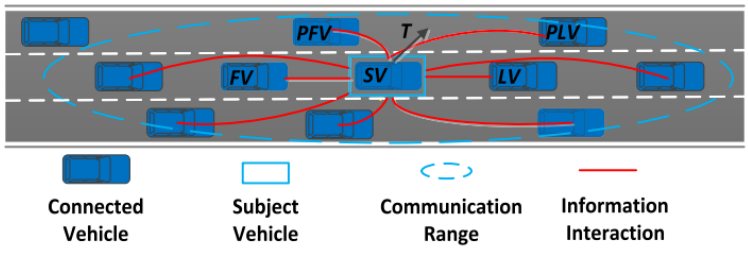

Figure 24. Cooperative lane changing environment depiction [42].

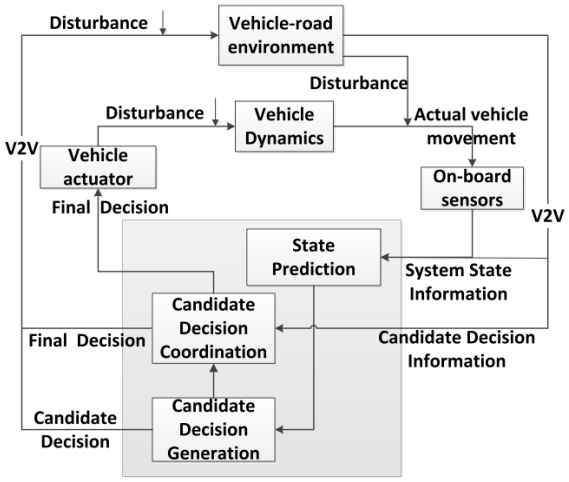

Figure 25. Collaborative lane-changing decision-making strategy [42].

In this case, the lane the ego vehicle (SV) pretends to join is denoted as T. As in other cases, the action of changing the lane is considered in this work as a discrete event, which in this study is also instantaneous. Regarding motion, only longitudinal acceleration and velocity are analysed. The proposed decentralized decision-making framework composed of the three mentioned modules is shown below. V2V communication enables passing information of current system state [42].

The state prediction module employs a car-following model to predict the following state of related vehicles. For such prediction, a modified form of the "full velocity difference" model is applied, developed in a related work, so that it becomes suitable for a connected vehicle environment. This model is the following [42]:

$$a_n = -\frac{1}{\tau}v_n + V\left(\sum_{i=0}^{m} r_i \Delta x_{n+i}\right) + W\left(\sum_{i=0}^{m} u_i \Delta v_{n+i}\right)$$

$$V\left(\sum_{i=0}^{m} r_i \Delta x_{n+i}\right) = \frac{1}{\tau}v_{\max}\left[\tanh\left(s\sum_{i=0}^{m} r_i \Delta x_{n+i}\right) + \tanh\left(s\left(\sum_{i=0}^{m} r_i \Delta x_{n+i} - h_c\right)\right)\right]$$

$$W\left(\sum_{i=0}^{m} u_i \Delta v_{n+i}\right) = \eta\sum_{i=0}^{m} u_i \Delta v_{n+i}$$

Where a_n is the acceleration of the vehicle n, of velocity v_n. Δx_{n+i} and Δv_{n+i} are its velocity and headway, relative to the leading vehicle. r_i and u_i are proximity coefficients defining the intensity of the interaction. τ is the inverse of a relaxation factor towards the legal velocity, s is a smoothing coefficient, η is the sensitivity to the relative velocity and h_c is the safety headway limit.

The candidate generation module takes as an input the state prediction results and generates an optimum decision by cataloguing desirable and undesirable situations. This ends up in the decision of "lane changing" or "lane keeping" for the candidates. The module must capture information from multiple surrounding agents. Candidate generation model is the shown below, formulated referencing the subject vehicle sv:

$$U(sv,C,T) = (\mathrm{a_{c_{sv}}} - \mathrm{a_{sv}}) + \mathrm{p}\left(\min(\{(\mathrm{a_{c_t}} - \mathrm{a_t})|\mathrm{t} \in \mathrm{N_T}\})\right) + \mathrm{q}(\max(\{(\mathrm{a_{c_c}} - \mathrm{a_c})|\mathrm{n} \in \mathrm{N_c}\})), \mathrm{T} \in \{\mathrm{L},\mathrm{R}\}$$

$$\mathrm{N_s} = \{\mathrm{j} \in \mathrm{V_S} : 0 \leq ||-\mathrm{X_j}|| \leq \mathrm{L}\}, \qquad \mathrm{S} = \mathrm{C\ or\ T}$$

$$\mathrm{TS} = \arg\max_{T\in\{\mathrm{L},\mathrm{R}\}} \mathrm{U}(\mathrm{sv},\mathrm{C},\mathrm{T}), \qquad \text{Subject to } \mathrm{U}(\mathrm{sv},\mathrm{C},\mathrm{T}) > \Delta\mathrm{a_{th}}$$

$$\mathrm{a_{c_{sv}}}, \qquad \mathrm{a_{c_t}} \geq -\mathrm{a_{safe}}$$

Where C and T are the current and target lane, L and R left and right target lane, $U(sv,C,T)$ the overall advantage of the sv for the lane changing decision in a cooperative situation, from C to T. N_c the set of vehicles in current lane and N_T the considered vehicles for the target lane, within the communication range. a is the current acceleration of a vehicle, while a_c represents the corresponding acceleration of that vehicle if the subject vehicle effectively changes lanes. Subscripts c and t refer to a following vehicle belonging to either N_c or N_T. X is the current position of a vehicle, $\mathrm{V_S}$ is the set of vehicles in the lane S at a given moment. L is the magnitude of the communication range. Regarding the calculation of the advantage $U(sv,C,T)$, the first term is the calculation if the sv changes to the target lane. The second term includes the politeness factor p and considers the influence of following vehicles likewise changing lanes to T. This term aims to reduce unnecessary or aggressive lane changes. The third term includes the benefit factor q, which quantifies the increase of velocity of following vehicles if the sv leaves the lane, encouraging behaviours with positive effects on traffic efficiency. If the latest both inequalities are satisfied, T with the greatest value of $U(sv,C,T)$ is selected as the target lane candidate TS and the decision is lane changing. In the opposite case, the decision will be lane-keeping. $\Delta\mathrm{a_{th}}$ is the threshold for the advantage value between lane changing and lane keeping [42].

Once the candidate decision is generated, it is broadcasted to the connected interacting vehicles immediately. This process is taken care by the coordination module. When confirmed, the final decision is transmitted again to the surrounding vehicles and its automatically executed by the low-level motion control system. As the vehicle moves, the system state changes, and the optimal decision variable is recalculated at regular time intervals. The coordination algorithm aims to prevent situations such as vehicle frequently leaving and re-entering a line or simultaneous change of two vehicles towards the same gap, being their locations too close [42].

The effects of decentralized cooperative lane-changing of traffic stability, efficiency, homogeneity, and safety are evaluated by numerical method simulations. They were driven using MATLAB, on two different scenarios, with and without an on-ramp at the right of the right lane. 600 vehicles were included in a three-lane environment, under the condition of identical driver-vehicle units. Therefore, it was expected that a stationary state of traffic flow would always be reached. Simulations are conducted using different driver behaviour models: MOBIL, FVDM and C-FVDM. Summarized, the result of numerous simulations using different combinations of scenarios and driver behaviour models, show that cooperative car-following considerably reduces the creation of shoch waves during originated from lane changes, by increasing the homogeneity of the traffic flow [42].

As a drawback, apart from assuming instantaneous lane-changing, which is far from reality, it also takes for granted that the CAV always receive the information without any lag. Analysing the effect of these two simplifications considering current state of the art of V2V technology is a logical step forward towards its real-world implementation. Likewise, the inclusion of more complete driver behaviour models would be needed to achieve more realistic results [42].

Potential fields theory application in AV

As priorly mentioned, among the fields depicted above, this work is focused on the decision-making problem. Moreover, the logic of the algorithm takes as a basis the potential-field model used in [28], which was the principle of a path planning algorithm that was simulated and later tested on an ad-hoc AV, in different stretches of a highway open to traffic, showing positive results in lane-keeping and longitudinal planning in an uncontrolled environment.

Reactive techniques have been described as the simplest and most natural way of solving motion-planning problems, especially in real-time scenarios like traffic systems, by interpreting current scenario based on a set of very few inputs, primarily the distance from the obstacles. Any reactive methodology is competent enough to cope with the task of driving straight on the road while avoiding obstacles around. Additionally, an overtaking behaviour and cooperating for the overtaking action can be modelled based on the specific choice of the reactive algorithm [29].

Within reactive techniques, potential methods are known for their ease of implementation. Its simplicity enables fast development and deployment, whereas it computationally inexpensive nature allows the algorithm to be used in systems with limited vision range, as it is the case of vehicles with LiDAR-based perception [29].

To describe the concept behind the potential fields approach, initially introduced to model the motion of robots, a negatively charged particle (the AV) in an electrostatic field is considered. This particle is attracted by positive charges and repelled by negative ones. The destination point can be considered in this case as a strongly positively charged static particle, causing the vehicle to move towards it, whereas lane boundaries and obstacles would have a negative charge. Due to the repelling forces tending to infinite as the vehicle gets closer to obstacles and boundaries, a collision is theoretically not possible in a continuous dynamics framework. However, discretization during simulation and real-time computing can end up in collisions when having insufficient resolution. No trajectory needs to be planned, the movement of the vehicle is purely reactive, it will always be attracted by the goal and will not stop moving towards it unless a very strong obstacle blocks the way and repels it. When modelling the potentials, there is no specific rule in this kind of approaches, if the moving agent is repelled from obstacles and attracted to the goal. The following figure illustrates the idea [29].

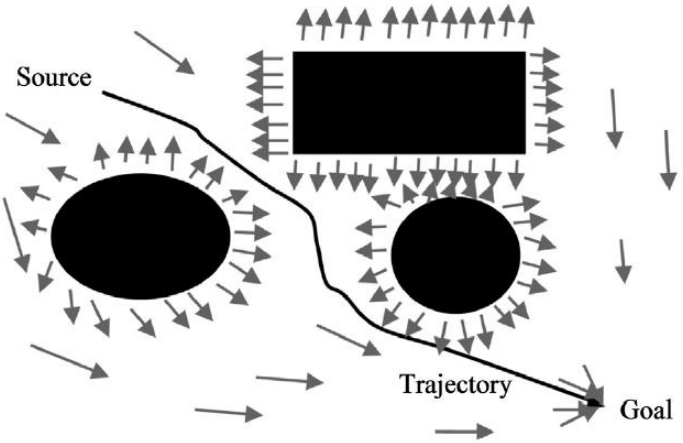

Figure 26. Representation of artificial potential fields [29].

However, the potential method cannot be directly applied, being the main reason the presence of roads within which vehicles need to be driven. Likewise, cooperation is not modelled in

regular potential approaches, whereas in traffic regular situations it is important for a vehicle to cooperate to allow another user overtaking it. Therefore, when applying potential fields model for AD motion planning, potentials calculation can be divided in lateral or longitudinal, both being interpretations of the original theory. Lateral planning mainly deals with deciding on feasibility of lane changes and the corresponding trajectory generation, acting primarily on steering actuators. Longitudinal planning, on the other hand, involves mainly speed control, with the aim of maintaining the vehicle in one lane, being steering likewise used when driving through curved paths [29].

3.3.1.1 Lateral Planning Potentials

The model that will be described is the one exposed in [29], where, as well as in the algorithm developed later in this work, it is assumed that there is a map of a road segment available where road boundaries are known. In this case, however, the road is bounded by road boundaries on both sides. The objective of the model is to produce vehicle movement at every time instant preventing it from colliding with any obstacle or road user, keeping at any time a certain safety distance. Vehicles present in the environment are assumed to have different sizes (l x w) and different preferred speeds (vpref). There is no limit to the allowed speed. The key task of the model is to decide lateral position by applying LPs, which may be positive and move the vehicle in the positive Y axis (transversal to the road path) or negative and do the opposite. The sum of potentials from all sources will determine the resultant value acting on the vehicle [29]. Calculation of individual potentials that compute in this sum are, disregarding some specific considerations taken in [29]:

- Forward potential: Generated by vehicles located in front of the ego vehicle, its value not only depends on the distance to the obstacle, but also of its relative speed to the AV. This way, an obstacle (or road user) that is moving away from the AV with positive relative speed, it would not generate any potential as no evading action should be taken [29].

$$P_{fi} = \begin{cases} 0 & v_b \geq vpref \\ \dfrac{vpref - v_b}{d_{f_i}} & v_b < vpref \end{cases}$$

  Where *Pfi* stands for frontal potential corresponding to the object *i*, *vpref* to the preferred speed of the ego vehicle, *dfi* to the relative distance between ego vehicle and the object (longitudinally to the AV trajectory) and *vb* their relative speed. It is important to note that, when several vehicles are located in front of the ego vehicle, the normal approach is to consider only the largest front potential, which theoretically does not need to correspond to the closest front vehicle [29].

- Side potential: Which in this case is not only applicable to vehicles and obstacles but also to road boundaries, which distance is measured in the transversal direction of the lane. In this case, the resultant lateral potential is the sum of maximum potentials generated at both sides of the AV, where the ones placed at its left compute negatively. A side potential of 0 would mean that the AV would continue travelling through the planned path, a negative potential that it should correct the trajectory towards right and, a positive potential, the opposite [29].

$$P_s = P_{li} + P_{ri} = -\max\left\{\left(\frac{1}{d_{l_i}}\right)\right\}^2 + \max\left\{\left(\frac{1}{d_{r_i}}\right)\right\}^2$$

In this case, relative speed is not included in the calculation as most side elements do not move against the vehicle and are not going to intercept its trajectory at any `point as the AV moves mainly longitudinally [29].

- **Diagonal potential:** Being normally only calculated for the front diagonal, it considers objects, vehicles and road boundaries places at 45 degrees of the longitudinal direction of the vehicle. The main aim of also including the calculation of diagonal potentials is foreseeing the need of trajectory correction in advance towards a smoother vehicle reaction [29].

$$P_d = P_{fl} + P_{fr} = -\left(\frac{1}{d_{fl_i}}\right)^2 + \left(\frac{1}{d_{fr_j}}\right)^2$$

- **Back potential:** Which accounts for rear-positioned vehicles with greater desired speed than the ego vehicle. Depending on the application, this kind of potential can be considered for triggering vehicle cooperation and hence drifting to a side of the lane easing rear vehicle overtake [29].

$$P_{bi} = \begin{cases} 0 & v \geq vpref_i \\ \dfrac{vpref_i - v}{d_b} & v < vpref_i \end{cases}$$

Ultimately, following a pure potential field motion planning approach, the lateral planning of the vehicle, which refers to the control of the steering, is dominated by a weighted sum of all the above potentials. Each potential is multiplied by a sensitivity factor that is calibrated to prioritize the effect of more relevant potentials over others, towards a natural and safe reactions of the vehicle [29].

$$p = \alpha \cdot p_f + \beta \cdot p_s + \gamma \cdot p_d + \delta \cdot p_b$$

Finally, this overall potential calculation is translated to steering with a simple proportional relation governed by the factor k, restricted according to vehicle rotational speed limitations [29], which is also calibrated normally through simulations trial and error.

$$\theta' = k \cdot p$$

3.3.1.2 Longitudinal Planning

As priorly stated, longitudinal planning deals with the decision-making related to the speed of the vehicle. The goal is that the vehicle continues driving at a speed as close as possible to the user desired speed, in parallel with ensuring a safe motion. In this case, the behaviour is not driven by the calculation of potentials. Instead, decisions are made by measuring the distance in the longitudinal direction. An example of longitudinal planning model would be the following one [29]:

$$v_{fi} = \begin{cases} v_{pref} & v_b \geq v_{pref} \\ \min\left(v_b + \sqrt{2 \cdot acc \cdot agg \cdot d_{fi}}\ ,\ v_{pref}\right) & v_b < v_{pref} \\ \min\left(\sqrt{2 \cdot acc \cdot agg \cdot d_{fi}}\ ,\ v_{pref}\right) & v_b = 0 \end{cases}$$

Where *dfi* is the distance to the considered obstacle or vehicle in front of the ego vehicle, longitudinally, *Vpref* is the preferred ego vehicle speed, *Vb* is the speed of the vehicle in front, being 0 when it is an obstacle, *acc* is the maximum acceleration and *agg* is defined as the *aggression factor*. This factor varies in value between 0 and 1, being a more aggressive response defined by higher accelerations and decelerations. In the model, the first case defines the case where no potential threat or collision is foreseen, driving therefore at the desired speed. In the second and third case, an object is placed in front of the AV and the relative motion between the two elements is studied to compute a desirable speed [29].

Overview and conclusion

The lane changing topic has lately been intensively treated, due to its importance in AV operation. Some automotive manufacturers start offering autopilot systems including automatic lane changing. However, despite the considerable amount of research articles available regarding the topic, not many approach the problem by providing complete, proven solutions, which resembles there is still considerable room for alternative decision-making algorithms definition and validation.

Currently, not all proposed lane changing models consider the dynamics of surrounding vehicles, limited to static motion planning. This leads to an artificial distinction between lane-changing decision-making and motion planning. Some of the works, however, intermittently apply both to increase the integration of decision-making through the dynamic lane-changing process. Amongst the authors that do make this consideration, an important amount of the works disregard mixed traffic, including only a simplified description of the geometry of the agents. Likewise, the amount of them that consider inherent time delays in the decision making and motion control process is negligible, as many of the works do not involve the conduction of field experiments and therefore certain practical matters tend to be overlooked [43].

There are, however, works that do present more complete solutions, such as the one exposed in [43], proposing a dynamic lane-changing model for AV incorporating human driver behaviour in mixed traffic. It is based on the usage of car-following and lane-keeping models, line-changing decision-making, dynamic trajectory generation and predictive control trajectory tracking, to cope with above-mentioned computation time delays. Field experiments were conducted on a large-scale test track, to validate the model. A picture of the instrumented vehicle conducting the field test is shown below [43]:

Figure 27. Bird's-eye view of a lane-changing field experiment [43].

4 Implementation

Regardless the evident tendency towards the use of artificial intelligence techniques, which in some of the works enable predictive lane-changing behaviour, the logic exposed in this work is approached in a traditional way, which is, a deterministic algorithm that produce a reactive behaviour, inspired in a potential field model. This approach represents a considerable simplification of the workload for the algorithm's development, which enables it to lie within the scope of this Mater thesis project.

The lane-changing algorithm is intended to perform in a generally controlled environment. This is, a homogeneous straight segment of highway, consisting of two lanes. The code is supposed to output lane change permission as a continuous signal.

Even before writing the code for the decision-making, it had to be considered what kind of traffic data would be treated and in which format, for the validation of the algorithm and to be able to generate results. Initially, it was planned to use real point cloud data, extracted from an instrumented vehicle from the INSIA institute, in a scenario where urban and highway scenarios are intercalated. MATLAB Lidar labeller tool allowed importing the recorded point cloud (shown in the figure below).

The data processing starts by determining the region of interest (ROI) by using a ground segmentation algorithm that enables disregarding the points belonging to the floor. Then, vehicles entering the FoV are manually labelled by taking a temporal fragment of the recording and adjusting cuboids to the dimensions of the observed vehicles. Finally, once the labelling of the desired vehicles is done, a pre-trained Kalman filter automation algorithm for tracking the labels created. The information of position, dimensions and velocity of the cuboids can then be exported into MATLAB Script editor for their use in the lane-changing algorithm programming. However, the data generated from the processing of this real point cloud proved to be inadequate for the development of a simple lane-changing algorithm. Primarily, the calculation of the virtual potential forces, together with logic applied to their results, relies heavily in an accurate knowledge of the position of the line lanes respectively to the ego vehicle and to the rest of the agents. Likewise, other minor disadvantages are the difficulties involved in obtaining datasets with variable traffic densities, in straight highway, where the ego vehicle must always stay in a known lateral position (preferably slow lane), to be able to develop the code and analyse the results in a controlled manner.

4.1 Data generation through simulation

Due to the exposed above, the alternative has been to generate the vehicle data through simulation tools, rather than extracting label data from real point cloud, towards the development of the algorithm. To this respect, MATLAB's Driving Scenario Designer app is specially conceived for algorithm development applications.

Functioning of the "Driving Scenario Designer"

The tool allows the creation of a highway with the desired dimensions, curvature, and number of lanes. It is intended to be used for simulation of relatively short road segments, however. Otherwise, the tool can easily collapse due to the amount of data generated.

Agents are sized as desired and introduced in the created road segment, by drawing independent trajectories and velocity profiles. The velocity is firstly set as constant, so that the drawn trajectories are translated into a coordinate table including data for every time interval.

After that, the discrete velocity at each time step can be modified in the table shown in the figure below.

While performing the simulation, the drawn trajectories are automatically smoothened, emulating natural driving behaviour. The output of it is a MATLAB data frame including the position, velocity, orientation, and size of all the agents at each simulated timestep, which is structured in the same way a typical vehicle tracking algorithm would do, analogously to the above-mentioned Lidar Labeller tool.

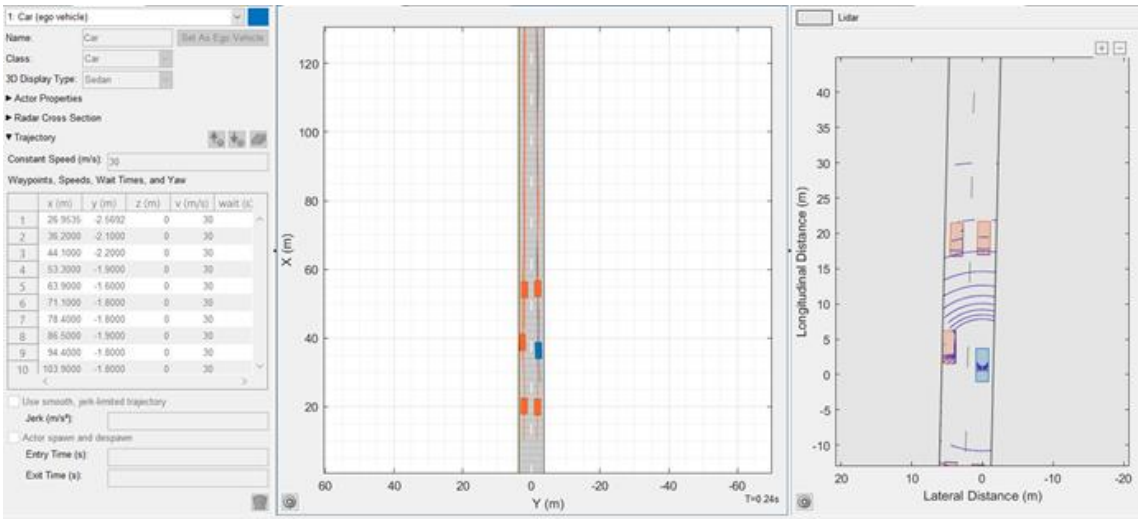

Figure 28. Agent data generation through MATLAB simulation.

An ego vehicle, amongst the generated ones, is after selected, including the setup of theoretical sensors, which type, number, key parameters (such as number of layers, range, FoV, generated noise, etc.) and location can be adjusted (see image below). An important outcome of this is the possibility of generating a simulated three-dimensional point cloud of the environment (top view at the right of Figure 29), which will depend on the mentioned selected parameters for the sensors and the generated geometries around the ego vehicle through each time step (right of Figure 30). Although theoretically, this allows analysing the vision systems from the beginning and developing the algorithm accordingly, for different setups and sensor typologies.

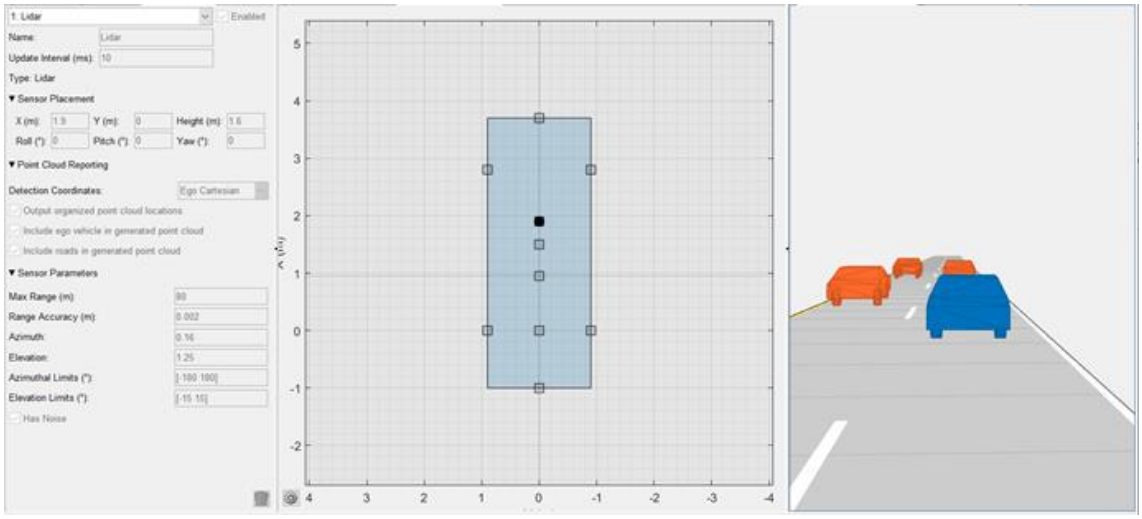

Figure 29. Theoretical sensor setup on ego vehicle towards point cloud generation through simulation in MATLAB.

Generated data sets and considerations

4.1.1.1 Initial data set

Initially, to develop the algorithm around the format and structure of the data generated through the simulation, a preliminary data set was created. In this case, a 200 m straight highway segment was drawn, divided into two lanes, regarded as the slow lane (right) and fast one (left). The road segment being drawn completely straight eases knowing the location of the lane lines at each time step, regardless the position of the ego vehicle. This represents one of the key differences between the simulated data and the one obtained by vehicle tracking from real point clouds.

An important consideration is that the Driving Scenario Designer from MATLAB does not integrate any traffic models, which means that the velocities must be manually described, and vehicle collisions occur as soon as the velocity of a rear vehicle is greater than the immediate front one and the simulation time is long enough. However, velocity profiles across a known trajectory can be calculated separately following any model (e.g., car following) and implemented after in the tool.

In this case, six agents were introduced, five of them (orange) surrounding the ego vehicle (blue) so that all considered possible agent locations (considered for the lane-changing problem) generate data to validate the algorithm. As stated above, the velocities of all the agents were set as constant. However, they were distributed in a way that no vehicle collisions occur (agents tend to separate from each in their trajectories as the simulation runs). This is simply done by assigning the highest speeds to the vehicles in front and the lowest to the ones in the back. To have data that produces representative results, the vehicles on the left lane need to overtake the ego vehicle, which are expected to produce an alternating lane-changing confirmation output, which frequency will depend on the spacing and speed of these.

For the sake of simplicity, although it is possible to introduce different sizes and types of vehicles, all the agents were selected as standard “Berlina-type” cars. In the Ego vehicle, one LiDAR sensor was theoretically positioned in the roof, centred in the transversal axis of the vehicle’s body and slightly upfront in the longitudinal one, at 1.5 m from the front end, as shown in Figure 30. It is especially relevant to know the position of the sensor respectively to the vehicle, as it will be the origin of all measurements and, therefore, of the decision-making process. This emulates the disposition of the instrumented vehicle available in the INSIA, although it might not be the optimal distribution.

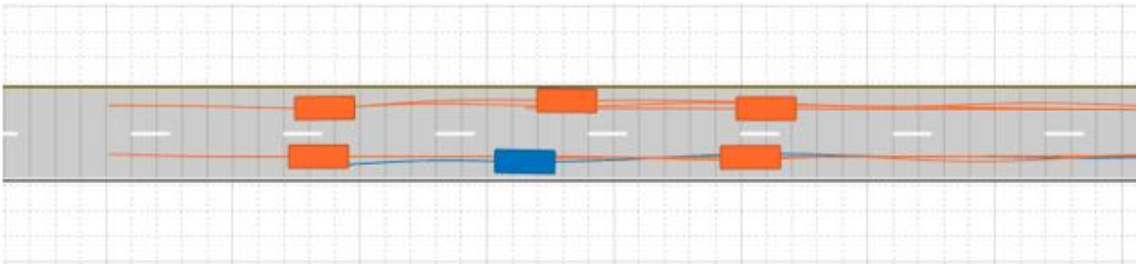

Figure 30. Image of created two-lane scenario with ego vehicle(blue) and five agents (orange).

4.2 Decision-making algorithm

Fundamentals and approach

The decision-making process is inspired in a potential field robot control approach. However, one of the key differences lies on the fact that the resulting calculated potentials do not directly drive the control system of the ego vehicle. Instead, their values are used only to evaluate the safety of a potential lane-change. The subsequent motion planning and control algorithm in charge of performing the line-change itself are not part of the scope of the project.

4.2.1.1 Considered potential fields

Essentially, in a standard setup, where the ego car could be surrounded by vehicles, the decision-making would be relying on the ones immediately around it, which are the 8 positions represented in the figure below (B1 to B8). Each of them computes a different potential. The front, rear and diagonal vehicles generate a speed and position dependant, whereas the lateral ones only rely on their position. Although not of them might be essential, all the traditional potentials in the control problem can be implemented in the lane-changing one:

- The speed of the front vehicle is relevant and that will drive the decision of the need for a lane change, depending on whether the vehicle directly in front has a speed higher or lower than the desired speed of the ego vehicle.
- Likewise, the speed of the rear vehicle, although disregarded in this version of the algorithm, can be used as a way of evaluating the safety of a lane change. If located in the slow lane and the rear agent is approaching the ego one at a considerable speed, it can be foreseen that it will try to overtake it, and then the manoeuvre of lane-changing prior to that becomes unsafe. In the opposite way, in located in the fast lane, a rapidly approaching rear agent represents, according to general traffic rules, that the ego car should seek for a change to the slow lane, allowing the rear agent to overtake. The speed dependant value of this potential can therefore be used as a threshold to perform the changing action.
- Likewise, the speed of the vehicles identified as diagonal needs to be considered, as they have a direct impact in the safety of the change. When positioned in the slow lane, the ego vehicle should only confirm the possibility of changing when either the vehicle in the back diagonal has a speed low enough and a distance long enough to ensure the dynamic safety distance will not be compromised once the ego stans in its way. In case of the front diagonal, it must be considered in the same way, as a vehicle positioned in the front part of the objective lane can have a negative relative velocity to the ego one and in the same manner compromise the safety of the change.
- Side potentials are however only distance dependant, as the expected outcome of its calculation is only to confirm that the destination lane is empty of vehicles.

- However, not only the potentials generated by the surrounding vehicles are considered, but the ones originated by the dynamically changing distances to the lane markings as well. This information would traditionally be used for lane keeping in the control problem, but it is used instead for evaluating the relevance of vehicle potentials. When side vehicles exercise a potential smaller than the one of the side lanes (which implies a larger distance) it is assumed that the destination lane.

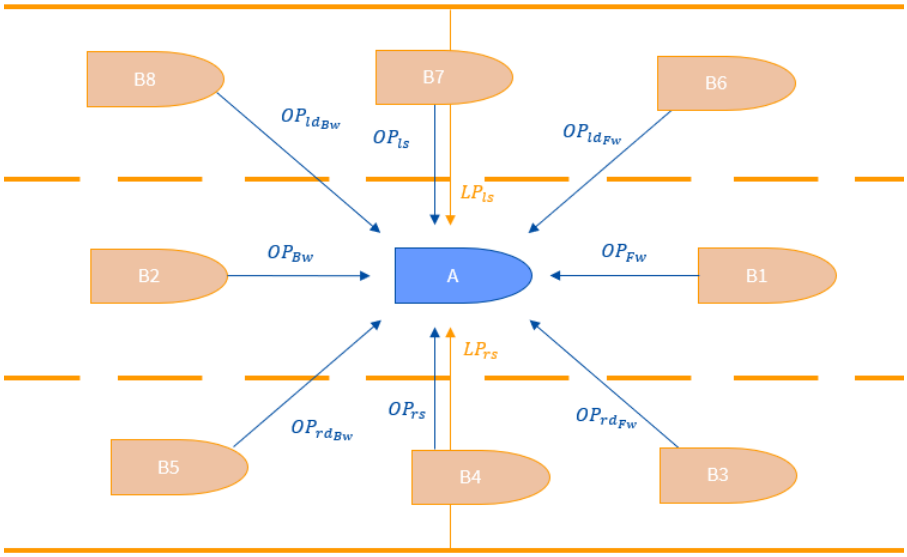

Figure 31. Representation of potentials and agents' distribution in a line-changing simplified environment.

4.2.1.2 High-level algorithm principles

In its first iteration, the high-level idea of the algorithm is depicted below, free of constraints linked to the programming language that will ultimately be used. It does not consider all the possible potentials above, but only the regarded as essential ones, in a two-lane road segment, when the ego vehicle, staying in the slow lane, pretends to find a safe gap in the fast one. These are, the left side potential, the left diagonal ones (front and rear) and the distances to the diagonal vehicles, in the direction of the trajectory, towards ensuring safety distance during and after the change. Instead of the front potential, only the speeds are used to make the same decision about lane-changing desire.

Driving through the high-level algorithm, like in most of the lane-changing models, the ego vehicle default situation would respond to a longitudinal model, in this case within the slow lane. Whether implementing a car following or free flow model in the actual longitudinal planning problem would depend on the existence of a front vehicle, discerned as a result of the calculation of the front potential, which is not considered here. Likewise, the steering control could rely on the side and diagonal potentials.

Initially, it is checked whether there is a vehicle in the front, in the affirmative case, the algorithm evaluates whether the vehicle in front (within the vision range) has a speed lower than the desired speed of the ego vehicle, which could be user-defined or set as the maximum allowable speed of the two-lane road. However, in the case of a front vehicle speed closely varying around the desired speed would not be an adequate threshold from a control perspective, as it would lead to inefficient or unnecessary lane changing operations. Therefore, a wait time is set. As soon as the first signal of front vehicle speed being lower than the desired

gets activated, a waiting time variable gets initialized and continues increasing for every time step unless the speed of the front vehicle increases over the desired one. Only when the waiting time threshold is reached, the algorithm skips the loop and enters a lane change intention mode.

Although it is not meant for the obtention of results through simulation, a symbolic way of visualizing this intention an enhance vehicle cooperation at this stage would be to turn on the left indicator light as the change intention persists.

Once having entered the change intention mode, different potentials (which were not relevant up until this point) start getting computed and sequentially evaluated following a defined order. In first place, it is checked that there is no vehicle circulating right to the side in the destination lane, which in this case is always the fast lane to the right. As a non-speed dependant potential, in this case it was decided to neglect the differentiation between line marking potential and vehicle potential, being the las one the only one considered. It is checked whether this potential is lower than a certain threshold, which would effectively correspond to the line marking potential calculation of the left line of the left lane in a later stage of the algorithm definition. In the affirmative case, the process moves forward.

The next one to be checked is the back diagonal potential, in charge of confirming the risk of vehicles approaching from the back in the fast lane. The result of the calculation is logically dependant on the speed, as well as the distance. Analogously as depicted in the theory section, large distances between the approaching vehicle and the ego one would return low potential calculations and the opposite would happen with the speed. In this case, the threshold OPdlim needs to be set empirically, as the equation returns any real number, depending on the desired driving aggressiveness or safety level.

Likewise, the distance to the back-diagonally approaching vehicle must be considered independently before considering another vehicle, as its position relative to the ego car should respect a safety distance after the lane change. As the speed calculation for the potential is relative to the one of the ego vehicles, a vehicle positioned too close for a safe change, with a speed around the one of the user, would return a potential close to zero and otherwise confirm the change viability, which is not acceptable. The threshold for this distance check can be the traditional safety distance, which is established in the literature as the length of the ego vehicle. It is highly recommended to use a so called “dynamic” safety distance, which considers the speed of the agents, as greater speeds involve greater braking distances, given a normally fixed reaction time of around 2 seconds. However, a static safety distance is regarded as sufficient in this case, as the relative speed was priorly evaluated with the calculation of the back diagonal potential, which threshold can be established considering this topic.

The calculation and evaluation of the front diagonal potential (of left-lane positioned vehicles) is performed after. The relevance of this action is similar to the above, but is given less priority in the sequence, given the fact that a fast-approaching vehicle in the fast lane is considerably more likely than a vehicle in the fast lane positioned in front braking to speeds lower than the flow in the slow lane. In any case, the calculation and evaluation are done analogously, as the potentials take as a reference the position and speed of the ego vehicle and therefore can be equally applied in both directions. The same thing occurs once again with the safety distance.

Once these three potential vehicle locations in the fast lane are checked, giving values higher than the thresholds in the case of the distances and lower in the case of potentials, the algorithm returns a confirmation signal for the change, which is continuous in time. During this process, the false result of any of these steps in the sequence would lead to exiting the lane-intention mode.

The speed of the vehicle in front is checked once again. If it remains lower than the desired, the time counter remains increasing, and the sequence of checks starts again. Otherwise, the

waiting time variable is reset to zero and the algorithm runs from the beginning, as described above.

4.2.1.3 Expected overall behaviour

The response of the high-level algorithm under different scenarios would be:

1. If there is no vehicle positioned in front of the ego one, the change intention would remain inactive, and the detection loop would continue running.

2. If there is a vehicle detected in front and its speed is greater than the desired, change intention would remain as False and the algorithm will continue checking the existence of such vehicle and comparing its speed until the speed decreases to the threshold.

3. If there is a vehicle in front with a speed alternating around the one of the ego one, the time limit for the wait will not be reached and again, the logic will remain withing the vehicle existence check and speed comparison.

4. If a front vehicle has a considerable low speed, this will be reflected in reaching the waiting time, and the ego car will activate the change intention.

5. If there are no vehicles detected in the fast lane, it will directly return a change confirmation, which is the end of the scope of the algorithm. This is not shown in the figure above, as it is not explicitly checked, but involved instead in the way the data of surrounding vehicles is treated in the coding phase.

6. If there is a vehicle detected side to side in the destination lane (left) the change will not be confirmed, and the logic will go back to the initial front-vehicle check and speed comparison loop.

7. If there is not a vehicle at the side, but one approaching from the back, back diagonal potential will be compared with the priorly calibrated threshold and distance in the direction of the trajectory will be the last checked prior change confirmation.

8. If there is a vehicle in the front, in the fast lane, the safety of the change will be checked analogously.

9. If there any of the vehicle positions are perceived as occupied in parallel, the corresponding checks will be done in order of priority, as shown in the diagram, and the confirmation will only be given if the individual safety checks are confirmed (which can never happen with a vehicle side to side, but can happen with a vehicle approaching from the back and another one braking from the front, if the gap is big enough in terms of distances and speeds).

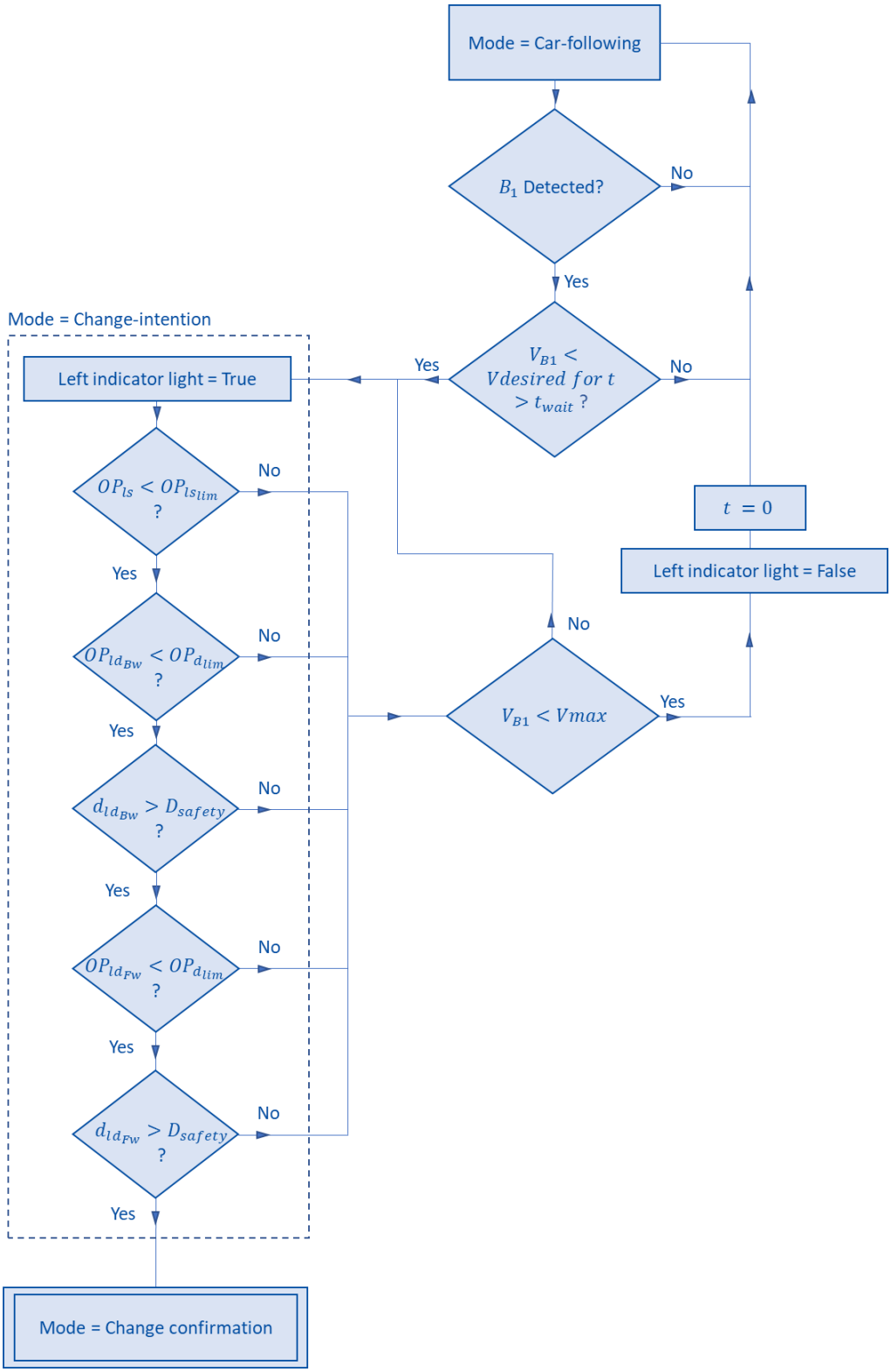

Figure 32. High level depiction of the lane-change algorithm.

Algorithm inputs and considerations

One of the most significant simplifications of this work is directly linked with the information the algorithm will receive as an input. For it to correctly perform in such a straightforward manner, it should know in advance what are the positions of all the agents respectively to the line markings. As priorly mentioned, this premise conflicts with the direct use of real LiDAR data, which is why the simulation alternative was taken.

However, its usage is in any case feasible, conceptually, as retrieving such information externally is within current state of the art of the autonomous driving filed. It will be assumed in this case the existence of electronic maps installed in the vehicle containing vectorized line marking information.

Likewise, GPS location and internal measurement unit (IMU) is assumed to provide accurate position and speed of the ego vehicle relative to the lane markings. The perception through LiDAR, together with priorly implemented vehicle recognition and tracking algorithms should provide an array of detected agents within the range of the sensor, including:

- Surrounding agents centre positions or, precisely, the centre of the cuboid resultant of the vehicle tracking result, respectively to the LiDAR location (dx,dy).

- The speed of such cuboids, which is given as a one-dimensional scalar, as their speed is assumed aligned with their trajectories, in the direction of the traffic flow.

- Vehicles dimensions (w,l), which is, the dimension of the cuboids, relevant depending on the vehicle location respectively to the ego vehicle, so that the position of corresponding sides of the agents (front, back, left side) are checked depending on the situation, as an addition to the position of their centres.

Other information that should be available is:

- The desired speed of the ego car, given either as a user input or a road maximum allowable speed, although this two tend to be the same in real world usage, it becomes irrelevant for the scope of the work as it will be passed to the algorithm as a constant in any case.

- The maximum waiting time, which is likewise assumed as user defined and a constant value in the decision-making (ideally, this value should be adjusted in real time depending on the density of the traffic, as some situations will require a higher “patience” of the user, for the sake of travel efficiency, when traffic jams occur.

- The relative position of the LiDAR sensor in the vehicle, which is, the distance to the front, back, left, and right side of the body (Lf, Lb, Wl, Wr), being the height considered as not relevant for this work. These metrics are necessary to establish real distances between ego and agents.

Finally, certain thresholds for the safety checks, such as OPldbw and OPldfw, need to be available in advance as a result of the analysis of the behaviour of the algorithm towards its calibration, which in any case is given as a pre-determined value and does not change through the operation of the logic.

Reference system and nomenclature

As mentioned, the origin of the reference system for the position of the agents is located in the main LiDAR sensor, so that the measurements of all agents' positions, and line markings are relative to the ego vehicle. To calculate the potentials of the markings and the agents, the first need to be catalogued depending on the occupied zone relative to the origin.

Without a system for directly correlating potential values with the occupied area, and assuming the availability of the accurate electronic maps, a straightforward way of dealing with the area assignation is defining the field of view (FoV) into delimited areas. If a straight road is assumed, these areas can be defined using a cartesian system, where the x and y locations of the cuboids clustering the agents are directly compared to these limits for the cataloguing task.

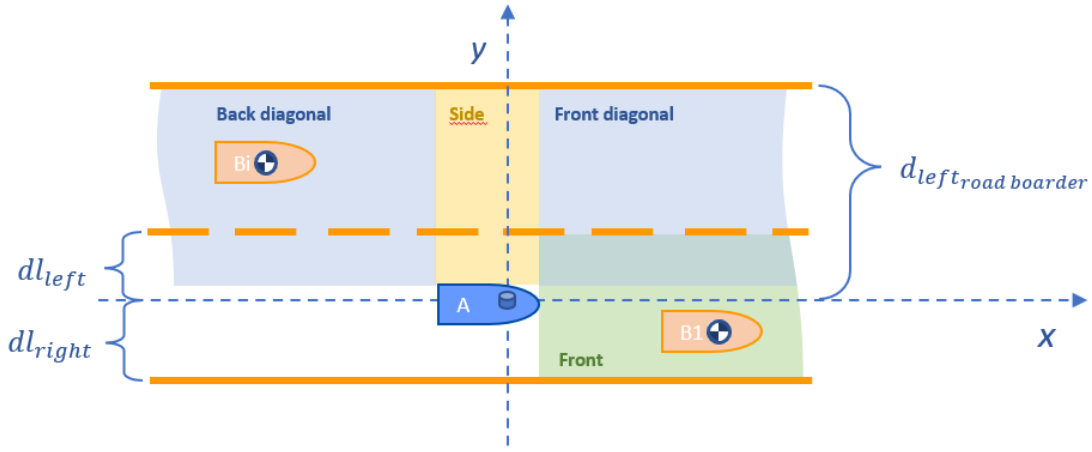

Figure 33. Chosen reference system for agents and lane marking location definition.

Likewise, the dimension of the agents is defined by their width and length, being the measures taken to the origin from the centre of the rectangle. With regards to the dimensions of the ego vehicle, they are taken from the main sensor location to the extremes of the body, which in this case are well known, unlike the case of the other agents, as vehicle recognition algorithms normally return cuboids which size is changing in time within a certain margin of error.

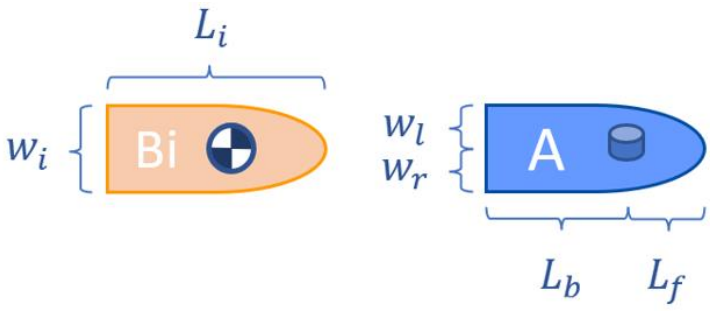

Figure 34. Definition of agents (Bi) and ego vehicle (A) dimension and location metrics.

In addition, a glossary of all the variables that will be considered in the detailed version of the algorithm is shown below:

- $d_{l_{right}}$: Distance from ego vehicle (A) to next lane line on the right side.

- $d_{l_{left}}$: Distance from ego vehicle (A) to next lane line on the left side.
- $d_{leftroadboarder}$: Distance from ego vehicle (A) to the line limit lane on the left side.
- W_i: Width of detected vehicle in detected vehicles array.
- W_l: Distance from the LIDAR position to the left side of the vehicle.
- W_r: Distance from the LIDAR position to the right side of the vehicle.
- L_b: Distance from the LIDAR position to the back of the vehicle.
- L_f: Distance from the LIDAR position to the front of the vehicle.
- B_i: Any not ego vehicle (A) detected and tracked, going from B1 to Bn, in the detected vehicles array.
- B_n: Last vehicle detected and tracked.
- V_{des}: Desired speed of the ego vehicle.
- V_a: Actual speed of the ego vehicle.
- V_{b1}: Actual speed of the vehicle directly in front of the ego vehicle.
- Y_{b_i}: Y coordinate in the reference System showed above of vehicle i.
- X_{b_i}: X coordinate in the reference System showed above of vehcile i.
- t_0: Initial time record for waiting time measurement.
- t_{clock}: Actual time in the considered instance.
- OP_s: Obstacle potential (differentiation from lane marking potential) corresponding to the side zone.
- OP_{ldfw}: Obstacle potential corresponding to the left diagonal zone in front of the ego vehicle.
- OP_{ldbw}: Obstacle potential corresponding to the left diagonal zone at behind the ego vehicle.

Algorithm drive-through

Below can be found the final version of the algorithm, still expressed as a logic diagram. A second iteration was needed to add the necessary detail when considering the format of the data given as an input, as a bridge between the high-level algorithm logic and the programming of the code. The algorithm, which can be found below, follows the same structure as the high-level one. I t divided in two modules, which again correspond to car-following mode and change intention mode.

As the data input is a dataset where agents positions and speeds are given at every timestamp, along the duration of the simulation, a simple way of implementing a time-dependant logic has been to run the complete algorithm within every time stamp (when the conditions are given to switch to change-intention mode), which should be feasible, if execution time remains lower than the sample frequency. This way, if a new agent enters the FoV it will simply be considered in the same way as the existing ones in the following timestamp, and the opposite would happen if an agent exits the FoV, which would mean that it simply stops being considered. This is possible due to the perceived agents being renamed (reassigned to agent variables) for every iteration.

Driving through the diagram, the task of cataloguing the agents is embedded in the logic, simply by using the thresholds of the areas depicted priorly. At the beginning of every time stamp, the car-following portion of the logic is executing, scanning through the position of all the agents given as an input.

4.2.1.4 Car-following section

The first section of the car-following mode intends to generate an array of vehicles positioned in front of the ego vehicle. Starts taking from the data set, for the corresponding timestamp, the

position of the first stored vehicle. It compares its X position (following the cartesian framework shown above, therefore in the longitudinal direction of the road), with the dimension of the front of the ego vehicle (Lf), which is measured relative to the origin (position of the LiDAR) and therefore it is directly implying that the agent is in the front. In the negative case, that agent is disregarded and will not be stored in the "vehiclesinfront" array, otherwise, its lateral position will be checked to be within the limits of the lane markings corresponding to the lane the ego vehicle is driving through. This is, dlright and dlleft, defined, again, relative to the position of the LiDAR. This will differentiate front vehicles in current ego vehicle lane from the ones in the adjacent ones.

Once this is checked, in the affirmative case, the label of the agent (numeric values ranging from 1 to n) and its position in the timestamp will be stored in the array. The loop will run until the position of the n vehicle is evaluated. Then, only the vehicle positioned directly in front will be considered, which is done by taking the minimum recorded X position in the array. At this moment, the counter for the measurement of the waiting time is initialized. If the speed of the considered vehicle is lower than the desired speed, then the priorly initialized time variable difference with the timestamp time label is checked to be higher than the maximum waiting time. Otherwise, the speed comparison will run again until the condition is reached.

4.2.1.5 Change-intention section

Only if this occurs, the value of the variable that designates the mode changes, "Carfollowingmode" is changed from 1 to 0 and the opposite happens with the variable "Changeintentionmode". This variable value change for the modes is not explicitly necessary for the decision-making, but used instead for tracing the decision-making process when extracting the results of the algorithm after having processed the data.

Within the same timestamp, the second part of the algorithm is only executed if the conditions above are met. In this phase, new arrays are generated prior to potential calculation, to locate the vehicles in the corresponding zones, analogously to the above and following the sequence priorly explained. Firstly, the X location of all agents are checked (an optimization possibility would be to only check vehicles that are not priorly located in another zone), when having a positive X position value, it is checked to be lower than the given distance between the front of the ego vehicle and the LiDAR location, in the opposite case, the negative distance from the back of the ego car to the LiDAR is used instead.

$$B_{i\,x\,coord} + \frac{B_{i_L}}{2} > -L_b \quad ; \quad B_{i\,x\,coord} - \frac{B_{i_L}}{2} < L_f$$

The dimensions of the agent are considered by subtracting or adding, respectively, the half-length of the agents to the position of their centre. This way, even if an agent is partially located to the side of the ego vehicle it will be regarded as a side agent, as vehicles completely positioned to the side parallel to the AV will never occur.

For this iteration of the algorithm, based on an ideal two-lane road, there is no actual need of checking the Y location of the agents for vehicles on the side, as it is assumed, if the X location of the agent is within the limits of the body of the ego vehicle, that it is positioned in the adjacent lane. In the case of implementation under different circumstances, or a real scenario, where there are chances of identifying parked agents or other objects to the side of the road, which should not be considered as side vehicles, a simple Y value check should be used using the distance to the next lane boarder as a threshold for it to be considered.

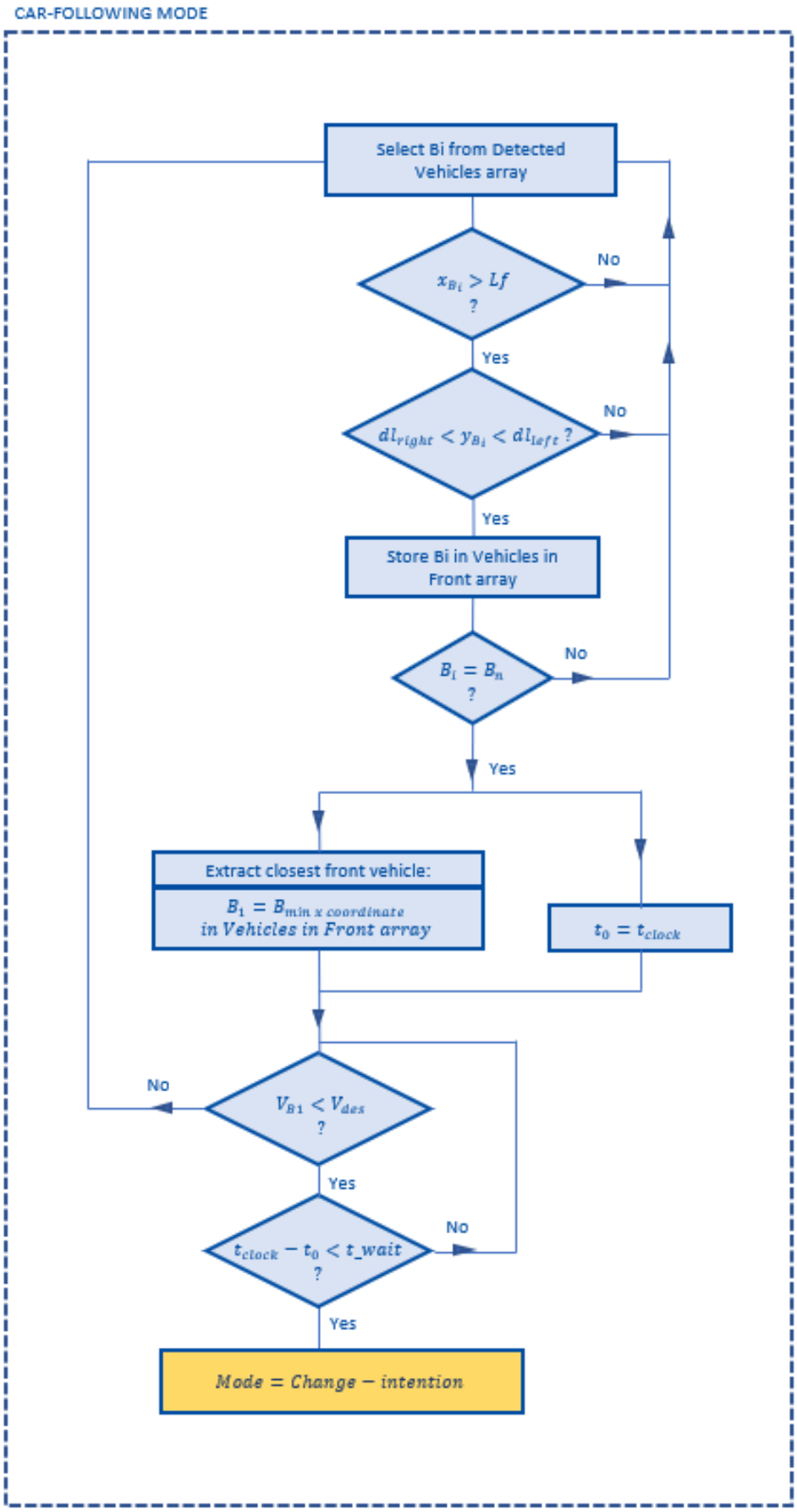

Figure 35. Programmatic version of the lane changing algorithm. Car following module.

Agents that fulfil the above conditions are stored in the corresponding array of side vehicles data. This array now includes a calculation of the side potential (considering again the above-mentioned), which is obtained as the inverse of the squared distance between the LiDAR position and the closest side of the body of the side agent (left side in this case):

$$OPs_{B_i} = 1/\left(B_{i_y} - \left(\frac{w_i}{2}\right)\right)^2$$

Again, only the maximum potential amongst the vehicles on the side is considered, which is checked against a threshold that is established based on the position of the left lane boarder line, giving greenlight for the next check if the potential of the agent is lower.

$$OPs_{B_i} = 1/\left(B_{i_y} - \left(\frac{w_i}{2}\right)\right)^2 < OPs_{lim} = 1/d^2_{leftboarder}$$

If the condition is not fulfilled, despite what is shown in the diagram below (which a recheck of the front vehicle speed, analogously to the high-level approach) all the sequential checks will be performed regardless the result of the result of their evaluation. This enables plotting the result of all check signals when testing the code, reducing the amount of different simulated scenarios needed to validate it.

Side check is followed by the back diagonal check, where the vehicles are identified to belong or not to the back diagonal area by checking: whether their x position is behind the rear side of the ego vehicle, whether its y location is to the left of the left side of the ego vehicle and whether at least the interior side of the body of the agent is closer to the ego vehicle than the left adjacent lane left boarder:

$$x_{B_i} < -L_b \;;\; y_{B_i} > W_l \;;\; y_{B_i} - \frac{w_i}{2} < d_{left\ boarder}$$

The selected agents get their back diagonal potentials calculated, as directly proportional their relative longitudinal speeds and inversely proportional to the squared diagonal distance between agents, calculated using x and y position values of both vehicles as below. Although traditional potentials are calculated based on speeds and distances in the same direction, this misalignment of these two in the proposed algorithm is considered a non-impacting simplification:

$$OP_{ld_{Bw}} = \frac{V_{B_i} - V_A}{d^2_{ld_{bw}}} = \frac{V_{B_i} - V_A}{\left(\sqrt{\left(x_A - x_{B_i}\right)^2 + \left(y_{B_i} - y_A\right)^2}\right)^2} = \frac{V_{B_i} - V_A}{\left(x_A - x_{B_i}\right)^2 + \left(y_{B_i} - y_A\right)^2}$$

Once calculated for all the agents, the safety threshold considered here against the maximum of the potentials is ≤ 0. This indicates that, either there is no vehicle within the range of the LiDAR positioned in the back diagonal, or the tracked vehicle/s are decreasing speed relatively

to the ego vehicle. Depending on the range of the LiDAR, this threshold might be too restrictive (certain positive relative speeds when having enough distance might be acceptable. Therefore, this limit is regarded as a calibration element, which should be obtained through the testing of the algorithm.

However, following the above logic for the back diagonal, there is a chance that the maximum obtained potential corresponds to a vehicle that is not positioned the closest to the ego vehicle. In this case, the position of this vehicle would not be evaluated towards the lane-changing confirmation, even though the distance to the ego vehicle might not respect the minimum safety distance, which is established as twice the distance between vehicles and gets checked for all the vehicles in the back diagonal array:

$$\max x_{B_i} < -(2 \cdot L_b + L_f + \frac{L_i}{2})$$

In the case of the front diagonal, the same logic is followed, with the difference of the sign changes needed to adjust to the different zone:

$$OP_{ld_{Bw}} = \frac{V_A - V_{B_i}}{\left(x_{B_i} - x_A\right)^2 + \left(y_{B_i} - y_A\right)^2} \quad ; \quad \min x_{B_i} > 2 \cdot L_f + L_b + \frac{L_i}{2}$$

Having these three arrays generated and potentials and distances checked, as described above, for every timestamp, the result is a value of 0 (False) or 1 (True) for the viability of a lane change, together with the rest of the calculations, which is described in detail in the following section.

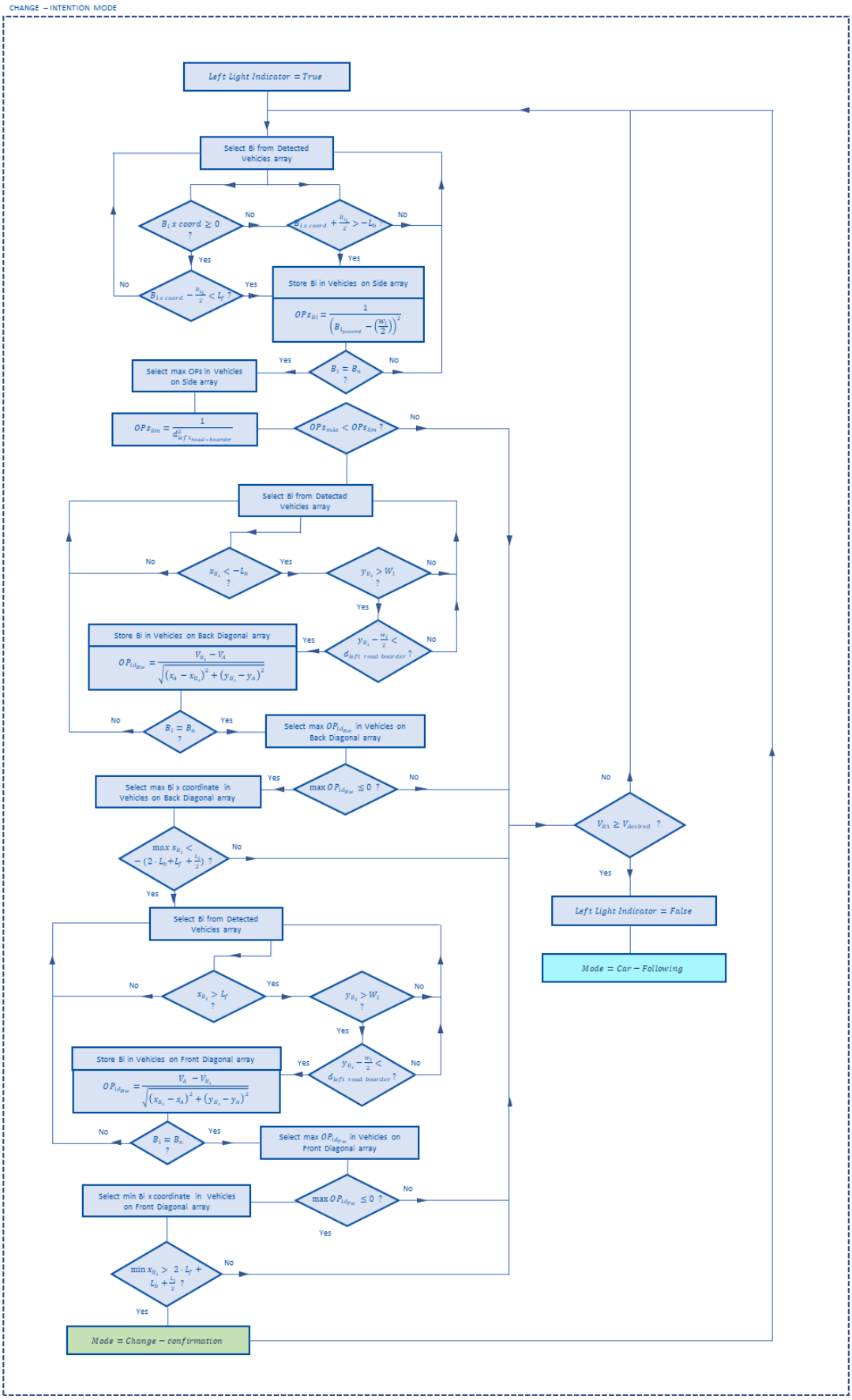

Figure 36. Programmatic version of the lane changing algorithm. Car following module.

Expected algorithm outputs

The outputs of the algorithm, after having computed the simulation data are stored in different arrays that cover the duration of the simulation. This way, having one value per timestamp, these outputs can be represented as continuous signals, with the temporal resolution achieved during the simulated data generation.

Ultimately, the lane confirmation signal is the expected outcome for the algorithm, which would then be used for triggering lateral planning mechanisms for the act of lane-change. However, especially during the development and refining of the algorithm, it is important to have visibility of the algorithm internal process, which is why the key variables used in the logic are also stored as continuous signals and represented, as shown in the two figures below.

These graphs show, in this case, the outputs of the program after having processed the initial data set depicted in the "Data generation through simulation" chapter, used primarily for validating the logic. This data set contains the trajectories of six vehicles, including the ego one, following a path at different speeds, as described above, for a total duration of 450 cs (4.5 seconds).

In the simulated scenario, a succession of three vehicles will continuously be overtaking the ego vehicle. To avoid collisions (reminding that the algorithm does not involve vehicle control), the front vehicle is programmed to drive faster than the ego one, and the one behind slower. However, the desired speed of the ego vehicle is pre-set as higher than the one of the front vehicles, to enable the second phase of the algorithm to be executed.

Likewise, considering the computational load of the data set generation, the waiting time of the ego vehicle has been set to 0, so that the change intention is enabled as soon as it is checked, for the first timestamp, that the speed of the front vehicle is less than the desired one. Therefore, as shown in Figure 38 (a), the "Chanintention" signal is returned as a continue "True" value (1).

With respect to the side vehicle check, it returns a value of 1 when the vehicle on the side is not present. In the simulation, the vehicle directly on the side overpasses the ego vehicle at the beginning of the simulation, which can be observed in the approximately half second change in the "SIDECHECK" output signal Figure 38 (b).

The front diagonal check, in this case, as it is expected, always remained as "True" (Figure 38 (c)), implying the safety check was successful, as the agent has a higher speed than the ego vehicle and therefore a negative potential, according to the front diagonal potential definition given above. Likewise, the safety distance check was always fulfilled.

The calculation of the front diagonal potential (Figure 38 (d)), which is the basis for obtaining the above signal, returns a parabolic curve of increasing negative values, as the vehicle in the front diagonal has a constant positive speed greater than the ego vehicle and the distance between them increases with the time.

The back diagonal check, on the contrary (Figure 39 (b)), constantly returns a "False" response. This is judged as a correct result, as the vehicle positioned in the back diagonal is approaching the ego vehicle with a positive relative speed. In Figure 39 (d), therefore, can be observed how the value of the potential parabolically increases with the quadratic reduction of the speed. This happens independently from the safety distance check, as the threshold for a positive back diagonal potential is preliminarily set to zero. Although the output of the code is as desired, this should be revised, as a positive potential, if a dynamically defined safety distance is respected, should be considered as a safe manoeuvre. This would alternatively be achieved by empirically setting up a potential threshold greater than zero.

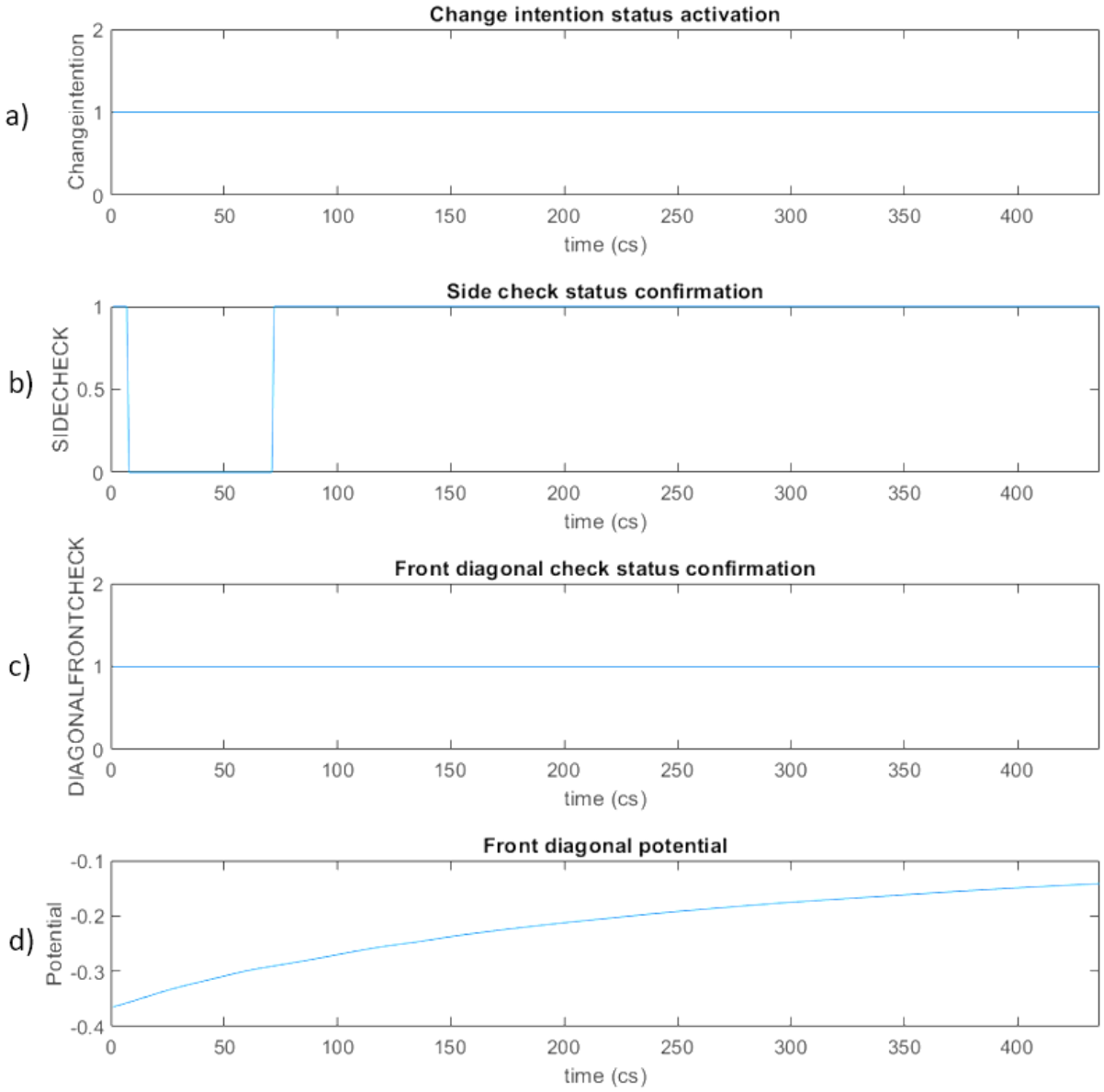

Figure 37. First set of algorithm outputs for the development dataset.

The side potential is considerably representative (Figure 39 (c)). It shows variable values from the beginning of the simulation, until approximately 70 cs ahead, when the curve instantly becomes zero. This represents the fraction of the simulation where the side vehicle is situated within the side range, respectively to the ego vehicle. The curve becomes zero when the side agent overtakes the ego vehicle. The variation shown until it becomes zero is explained by the smooth curves in the trajectories of the agents, when being "manually" drawn, in the Y direction. This variation does not appear in the diagonal potentials, as it is in those cases the X distance direction the one considered, which is constantly increasing, as the velocities are defined as constant.

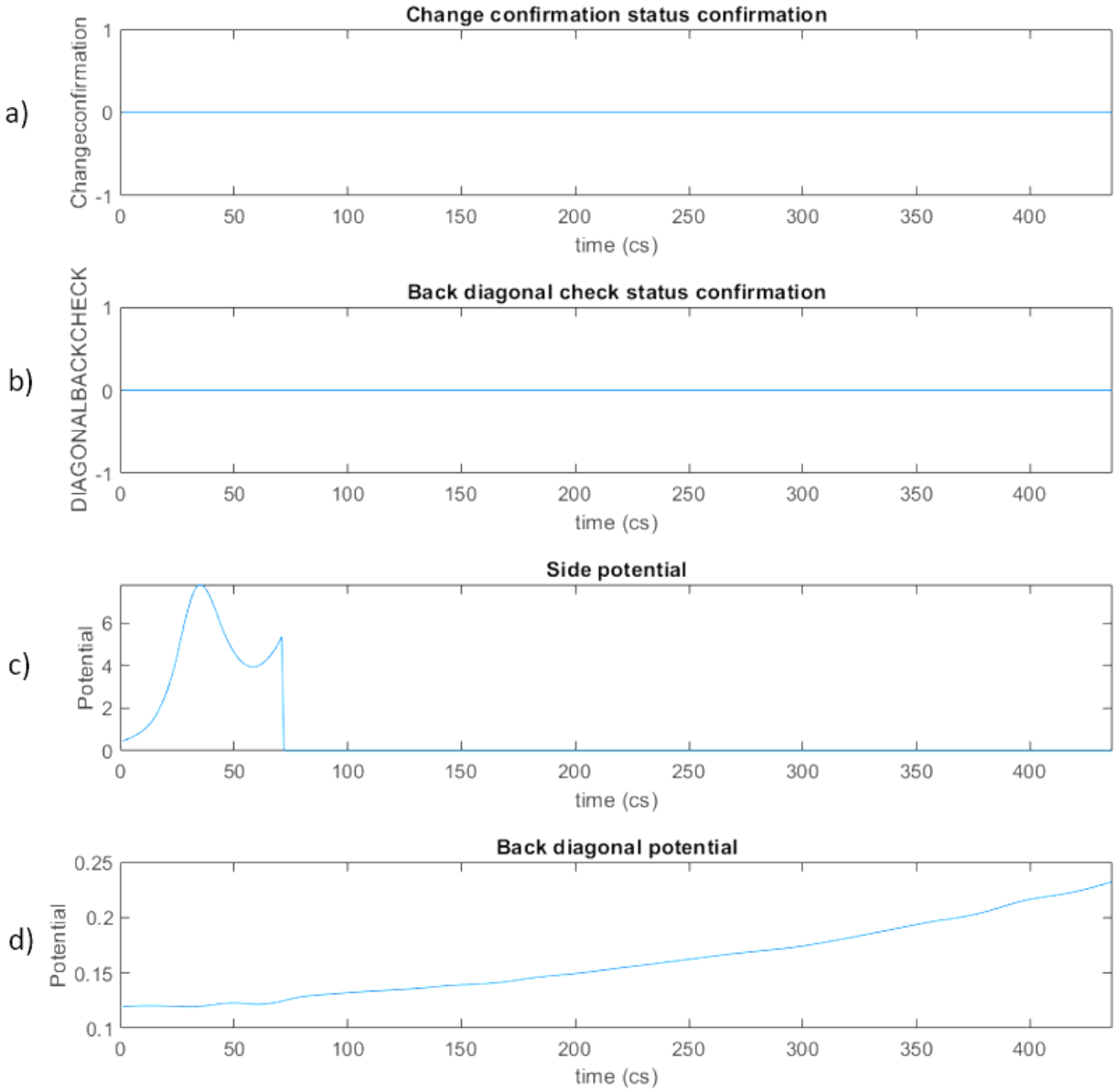

Figure 38 Second set of algorithm outputs for the development dataset.

After having executed the second part of the code, responsible for applying the logic of the algorithm (the first part of the code, as explained below, generates the simulated data), an execution time report was extracted, which is shown in Figure 40. It can be observed that the total execution time is below 1.8 seconds. As the simulated time corresponds to 4.5 seconds, which is considerably longer, it implies the possibility of executing the algorithm faster than real time data input, which makes it potentially suitable for real time data processing in a real autonomous driving scenario.

4.3 Code structure

MATLAB code for driving scenario data generation

The "Driving Scenario Simulation", as mentioned above, returns a MATLAB code that generates the simulated data. It includes the creation of the geometry of the road segment, vehicles, and their trajectories (actors), parameters of the LIDAR sensor placed on the ego vehicle

(sensor)and the corresponding point cloud. This code is included in the appendixes below, being structured as follows:

The main function links the data generation code with the decision-making algorithm, passing the generated data struct. This struct is created using a series of auxiliary functions, which is first initialized under the name “allData”, with a while loop that runs until the past simulation time is reached. In includes the time variable, poses of the agents for each time frame, the detected objects (which does not apply when vehicle tracking is not utilized, detected lanes (which in this case are created as explained above), the subsequent point clouds generated by the tool and a series of virtual instrument measurements.

The first auxiliar function (“createSensor”) creates the LiDAR sensor/s that are mainly used for the posterior generation of the simulated pointcloud, which are indexed with natural numbers and whose specifications are defined by the user. “UpdateInterval” represents the measurement frequency of the sensor (which in this case is set to 0.1 and should be less than the created timesteps in the simulation, to validate the use of the algorithm under real conditions). “SensorLocation” defines the position of the sensor as a vector including lateral and longitudinal coordinates, situated in preliminarily at 1.9 meters from the front and laterally cantered. “ElevationLimits” represents the maximum measurement angles that define the width of the FoV in the vertical dimension, being set in a range of -15 to 15 degrees.

“CreateDrivingScenario” generates the data that is key for this version of the algorithm (scenario and actors), as the prior sensor creation is only used in this case to establish the origin of the space segmentation and distance measurements. Firstly, the road segment is defined as a matrix containing the introduced points that conform its centre line, followed by the number of lanes utilized. The width of the lanes is standardized to 3.6 meters, although it can be defined differently. The ego vehicle is identified with the ID value of 1 and its trajectory is declared as a three dimensional vector indicating its initial position “Position”, and a matrix (“waypoints”) containing the x, y, z(=0) values introduced for as much points as desired, within the limits of the priorly created road segment. The speed (“speed”) is created as another matrix, with as many columns as points created for the trajectory, including scalar values on each column for the speed in the direction of the path. The rest of the agents are created analogously, with the only different of the “ClassID” and the “Name”, which defines them as different variables.

MATLAB code for decision-making algorithm implementation

The algorithm code, also shown in the appendixes, starts by initializing a set of variables. Three variables are used for the waiting time consideration. “t0” is a simple time counter, “t_clock” retrievers the time value of the evaluated timestep and “t_wait” is a user-defined maximum waiting time. All of them are initialized as 0 every time the algorithm is run (as mentioned above, “t_wait” is set to zero to accelerate the decision-making process. “Vdes” is also user-defined, representing the desired speed, and it has been initialized, with a value of 50m/s, higher than the fastest agent, with the same purpose. The different safety checks, “SIDE CHECK”, “DIAGONAL BACK CHECK” and “DIAGONAL FRONT CHECK” are initially defined as “false”. Likewise, the arrays that will contain the information of the designated vehicles in the different zones are created as a set of zeros, extracting the number of agents from the data frame.

Having the necessary variables initialized, the actual algorithm is organized inside a “for” loop that runs the logic of the algorithm through the complete data frame, being executed once every time step, until all the timesteps are evaluated. Inside this loop, a set of second level “for” loops run through the information of all the agents (the data frame is structured by containing this information for all vehicles inside each time step). Inside this for loops, the logic of the high-level algorithm, priorly exposed, is applied by a set of “if/else” operations. All the information necessary is obtained for applying these logic gates, such as vehicle positions, dimensions,

speeds, relative to the ego vehicle, are obtained by precisely referring to the exact location in the data frame structure ("allData"). The calculation of the potentials is directly stored by adding columns of information in the data frame (created by the simulation code).

A highly simplified version of the key section of the code is shown below to give an insight of the high-level structure:

```
SIMPLIFIED CODE DECISION-MAKING SECTION:
%-------------------------------------------------------------

for (loop running through all timestamps)
    for (loop running through all vehicle positions in one timestamp)
        if (set of rules to determine vehicle in front position)
              (record vehicle position)
        end
    end
    for (loop running through all vehicle speeds in one timestamp)
      if (set of rules to determine vehicle in front speed)
            (record vehicle position)

      end
    end
    if (provided that the waiting time limit is reached)
            Changeintention(l) = 1;
            for (loop running through all vehicle positions in one timestamp)
                if (set of conditions to find vehicles on the side)
                      (Calculate side potential)
                      if (limit is not reached)
                            Sidecheck = True
                      end
                end
            end
            for (loop running through all vehicle positions in one timestamp)
                if (set of conditions to find vehicles on the back diagonal)
                      (Calculate back diagonal potential)
                      if (limit is not reached)
                            backdiagonalcheck = True
                      end
                end
            end
            for (loop running through all vehicle positions in one timestamp)
                if (set of conditions to find vehicles on the front diagonal)
                      (Calculate back diagonal potential)
                      if (limit is not reached)
                            frontdiagonalcheck = True
                      end
                end
            end
            if (all checks are true)
                Changeconfirmation(l) = 1;
            else
                Changeconfirmation(l) = 0;
            end
    else
            Changeintention
    end
end
```

Figure 39. High level code structure.

5 Results

The algorithm has been tested against a set of scenarios designed after typical lane-changing situations. Some of them were used to fine tune the code and calibrate some of the potential thresholds and the last two to check the behaviour of the algorithm under specific situations that were not planned during its design phase.

The driving scenarios vary in terms of agents count, trajectories and speeds, but they all consist of a 1000-meter straight section of road. To reduce computing time sampling frequency has been reduced from 100 to 10 Hz.

The variables measured are the outputs of the signals versus simulation time, measured in centiseconds (cs) and deciseconds (ds), depending on the simulation resolution.

The list of conducted tests is:

- A variable relative speed in a unidirectional road, used to set up the back diagonal potential threshold, consists of the ego vehicle, positioned permanently in the slow lane at a constant speed of 80 km/h, following a vehicle in front with the same constant speed. Both cars are overtaken by a single vehicle, positioned on the fast lane, likewise with a constant positive relative speed.
- A reverse variable speed in a bi-directional road, which in this case is used to set the threshold for the front diagonal potential, as vehicles driving in the opposite direction are considered as the worst-case scenario of generated potential for such road segment. Besides switching the direction of the fast lane and the studied speeds, the scenario is designed in the same way as the prior case.
- A variable intensity test, where a car-following situation is generated in the fast lane at intensity values extracted from a Greenshields's lineal traffic model. The expected outcome of the test is identifying the intensity level at which lane-change confirmations start to occur, defining the functional limits of the road. For the sake of simplicity, the ego vehicle is positioned at zero speed.
- A rear overtaking, having the ego vehicle positioned as above, a single agent is approaching from the back at a greater speed, moving to the fast lane, overtaking the ego vehicle, and re-joining the slow lane. Vehicle trajectories are not crossed at any point. The trajectory of the overtaking agent is manually defined and adjusted using a jerk-limited smoothening function. So far, only single potentials were registered. This simple test aims to review the code when an agent moves through the different road sections, generating all the considered potentials.
- A traffic congestion situation, were both lanes are close to the maximum capacity of the road of 160 veh/km. The scenario is similar to the variable intensity one, but in this case both lanes are filled with vehicles and the speed of the vehicle is not null. The intention of the test is to review change confirmation results when the lane occupied by the ego vehicle is slightly slower and faster than the adjacent one.

5.1 Variable relative speed

The overtaking vehicle is set to a speed for each test iteration that varies from 120 km/h of relative speed to 90, 60 and 30, as observed in the figure below.

The back diagonal potential was initially set within the order of magnitude of the one corresponding to the average of the tested relative speeds and distance equal to the dynamic safety distance. The result of averaging this potential for the different speed was 0.075, which was then set to 0.01 for back diagonal potentials and 0.02 for front diagonal potentials. This difference comes from the belief of a front diagonal potential being positive represents a

situation of higher risk, due to the reduced awareness and reaction capability of the events occurring in the back of a driver, as opposed to the front, when the overtaking manoeuvre is being executed by another vehicle.

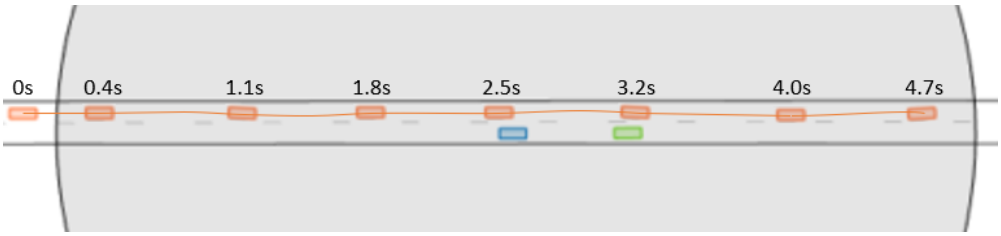

Figure 40. Testing driving scenario for relative speed of 120 KPH.

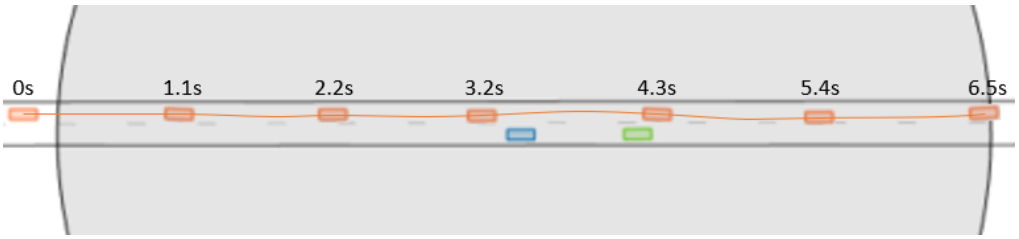

Figure 41. Testing driving scenario for relative speed of 90 KPH.

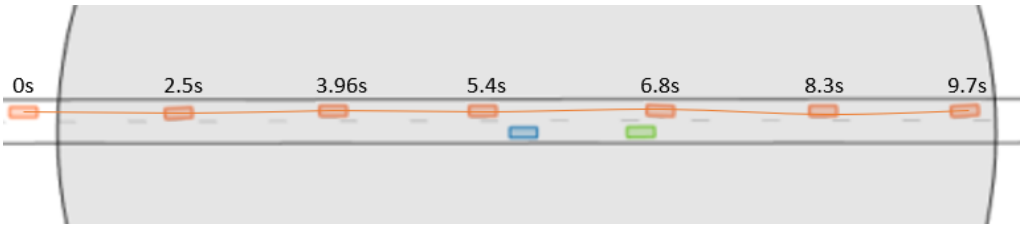

Figure 42. Testing driving scenario for relative speed of 60 KPH.

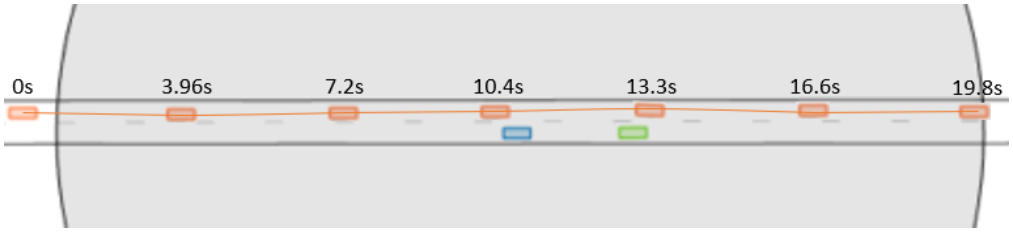

Figure 43. Testing driving scenario for relative speed of 30 KPH.

The following figure shows the type of signals obtained by the algorithm in such situation. There is a back diagonal potential being obtained from the beginning of the simulation, below the safety threshold initially, providing a lane change confirmation for a moment. The back diagonal potential transforms into a side potential as the vehicle overtakes and the lane change confirmation goes back to positive as soon as the vehicle leaves the left side zone. Front diagonal potentials are not shown, as negative potentials are not calculated in this version of the code.

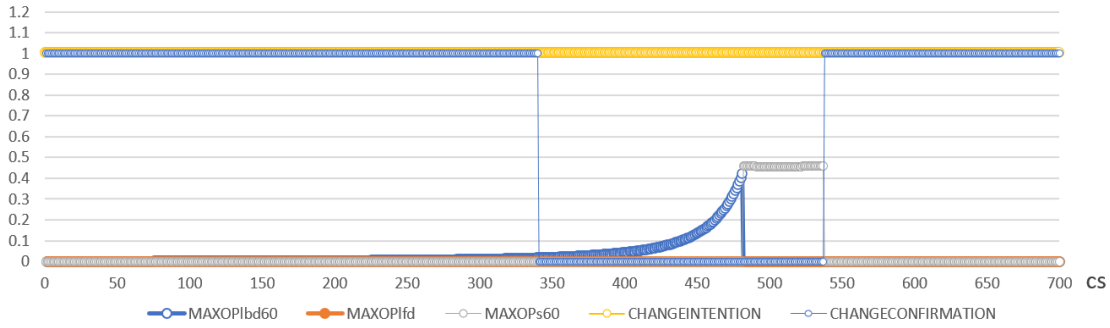

Figure 44. Output signals obtained from a relative speed test of the algorithm.

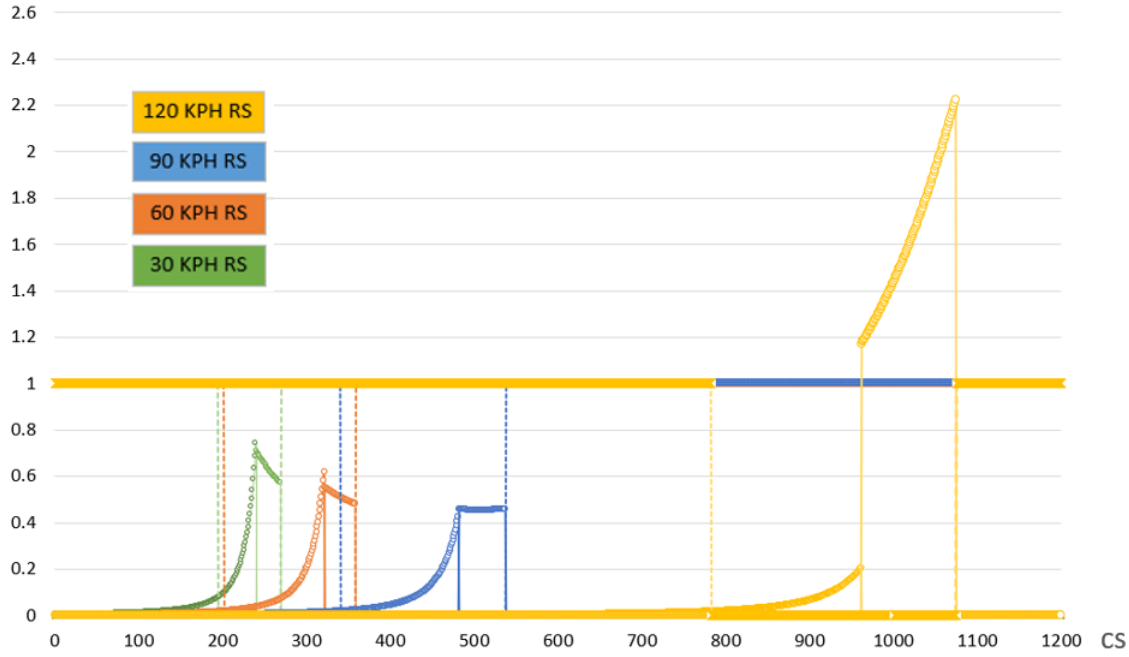

Figure 45. Comparative of the signals obtained for the tested relative speed scenarios.

The above graph shows a superposition of the potentials and lane confirmation signals for all the cases. If the time for the moment of lane change confirmation activation is compared to the vehicle positions through time, shown above, it can be observed that, as expected, higher relative speeds produce higher potentials. Higher potentials result in a prohibition to change lane when having greater gaps between the overtaking and the ego vehicle, however, the amount of time of changing prohibition considerably increases as relative speed decreases.

5.2 Reverse variable relative speed

After having tested the variable speed overtaking situation, it becomes of interest to compare these results to the case of a two-direction road shown below:

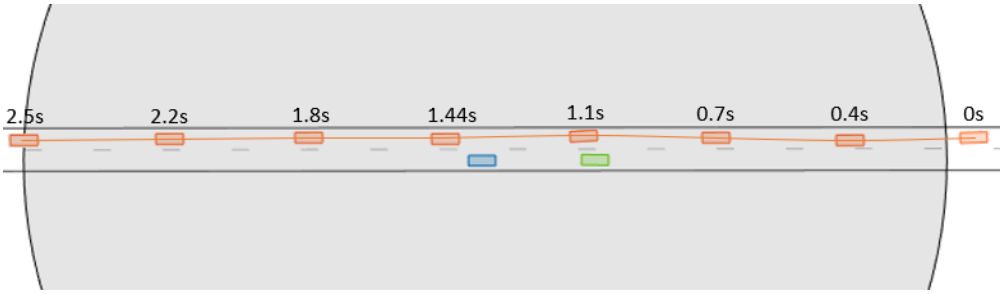

Figure 46. Testing driving scenario for reverse relative speed of 230 KPH.

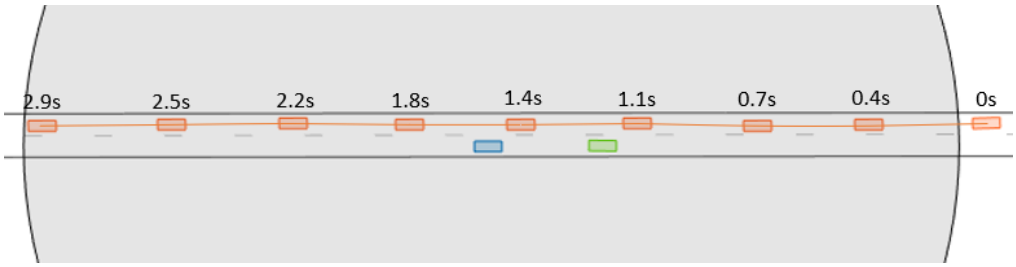

Figure 47. Testing driving scenario for reverse relative speed of 200 KPH.

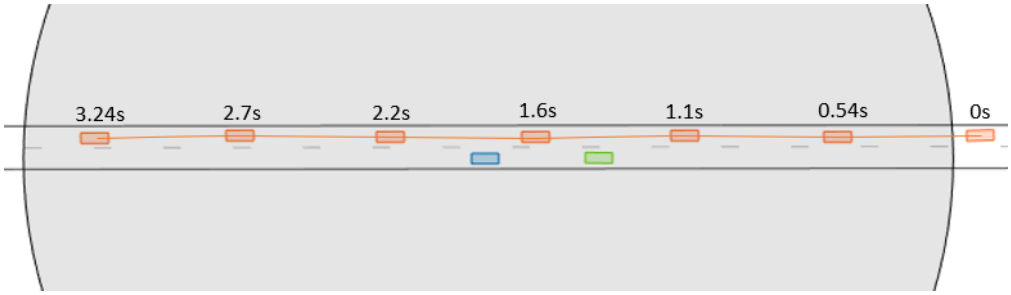

Figure 48. Testing driving scenario for reverse relative speed of 170 KPH.

The obtained signals are similar to the prior case, with the only distinction of the diagonal potential shown is the front one, instead of the back one:

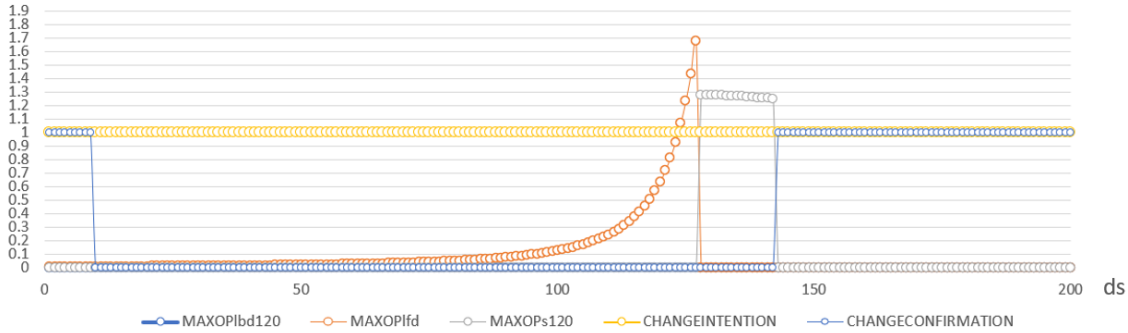

Figure 49. Output signals obtained from a reverse relative speed test of the algorithm.

The potentials obtained are considerably higher for the same absolute vehicle speeds, as the relative speed of the vehicle approaching is the result of adding the absolute speed of both

agents. This results, again, in higher distances to the start of the change prohibition, but greater positive change confirmation times.

It is important to highlight that, regardless the increase of distance where the prohibition occurs, it occurs within the limits of the range of the LiDAR (80m). The following figure shows a comparative:

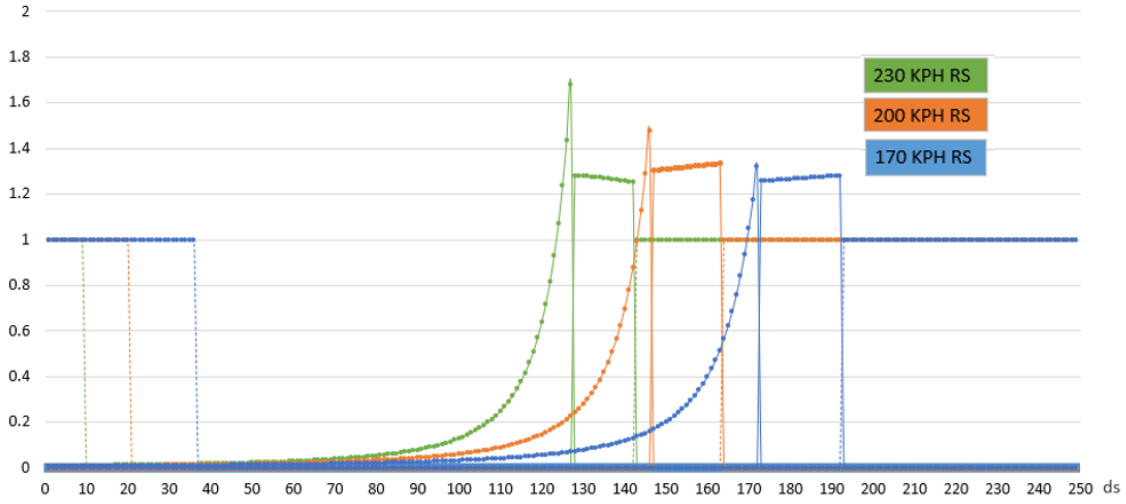

Figure 50. Comparative of the signals obtained for the tested reverse relative speed scenarios.

5.3 Variable traffic intensity

Once the decision-making thresholds were set, the functional limitations of the algorithm, which is directly connected to the potential limits employed, are related to the amount of lane change confirmation in a scenario of variable traffic intensity. The same straight section of road was again used, and the road parameters were calculated assuming a linear traffic model (as exposed in the theoretical section of this work), of a maximum capacity of 160Veh/h per lane. Applying Greenshields linear model:

$$K_c = 160\frac{veh}{km} \quad ; \quad V_l = 100\frac{km}{h}$$

$$Maximum\ intensity = i_m = \frac{V_l * K_c}{4} = 4000\frac{veh}{h * lane}$$

$$Traffic\ density\ at\ maximum\ intensity = \frac{K_c}{2} = 80\ Veh/km$$

$$Speed\ at\ maximum\ intensity = V_m = \frac{i_m}{K_m} = 50\ km/h$$

The intensity-density curve shown below represents traffic conditions of the considered scenario, per lane, responding to the equation:

$$i = V_l\left(K - \frac{K^2}{K_c}\right)$$

Which is populated as shown below:

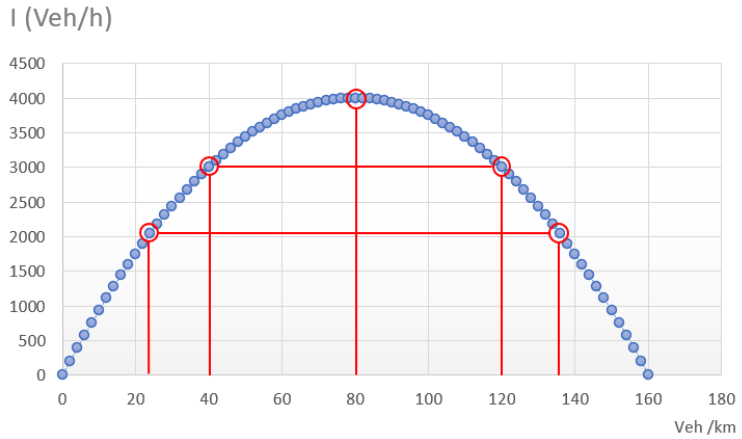

Figure 51. Traffic intensity calculation following Greenshields's lineal model. Testing at Imax, 3000 and 2000 veh/h.

Within the curve, 5 different conditions were tested, corresponding to three different intensity levels: Maximum intensity, 3000 veh/h & 40 veh/km, 3000 veh/h & 120 veh/km, 2000 veh/h & 24 veh/km and 2000 veh/h & 136 veh/km, as shown in Figure 52.

The results of the test can are contained in the Figure 53. The red line represents the change confirmation signal. When observing the results, it becomes evident that it is vehicle density in the destination lane what defines the possibility of exercising a lane change in a car-following traffic flow scenario.

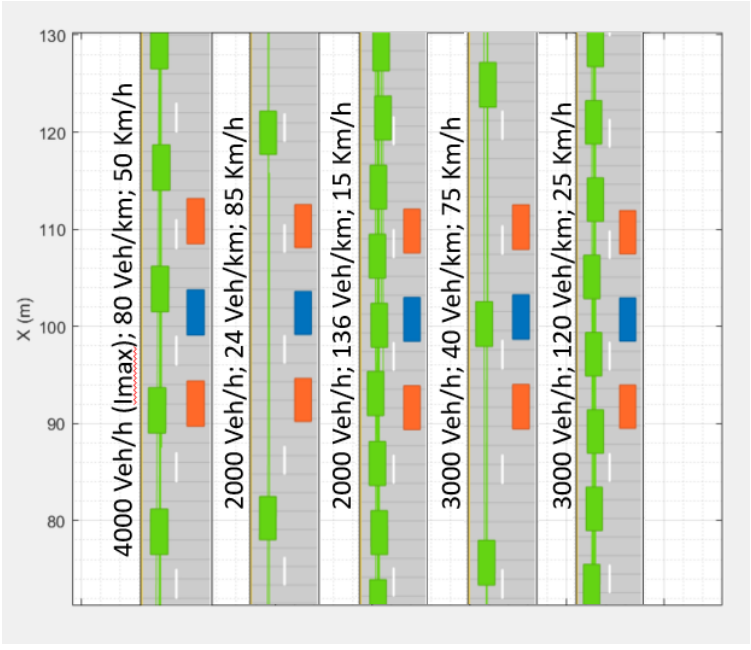

Figure 52. Tested driving scenarios for different intensity values.

In the fourth case, the change confirmation signal shortly activates within the gaps between vehicles in the fast lane, providing the functional limit for the algorithm. This limit is set to 24 veh/km, corresponding to a traffic intensity of cca. 2000 veh/h and a traffic flow average speed of 85 km/h.

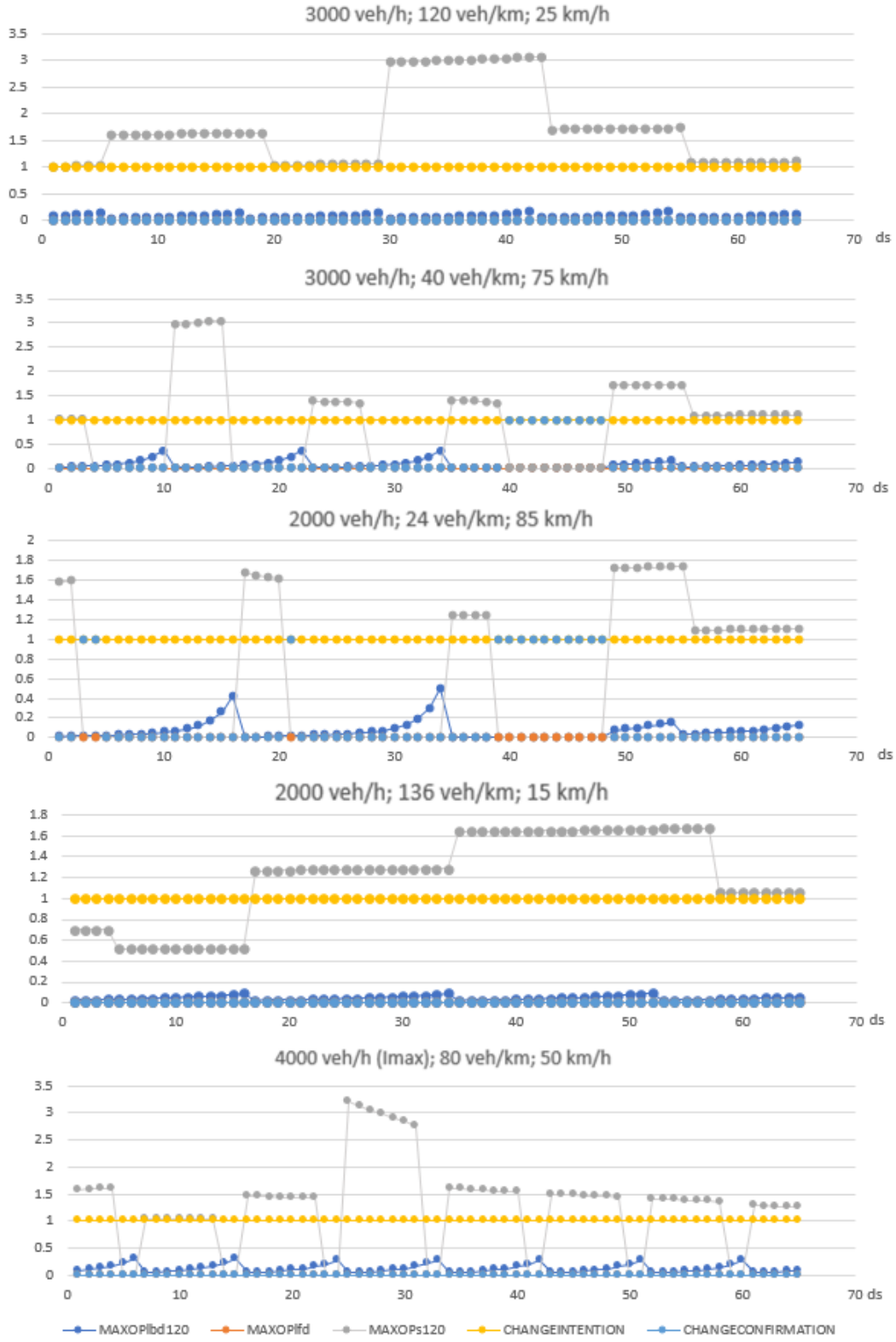

Figure 53. Signals obtained for the tested variable intensity scenario conditions.

However, it needs to be noted that this is the traffic density at which lane changes start to become plausible when the Ego vehicle is stopped. As the speed of the ego vehicle increases, until reaching left lane speed conditions (considering the left lane the fast lane), the calculated back diagonal potential reduces linearly and therefore the gaps for lane changing confirmations likewise linearly increase in time, having as an ultimate limit the dynamic safety distance once the speed of both lanes get paired.

5.4 Back overtaking

Traffic scenarios were also created to test a couple of specific situations. In first place, an overtake scenario, where the Ego vehicle is following a car in the slow lane at 100 km/h and a vehicle positioned in the back, at a constant speed of 120 km/h, positioned in the same lane, moves to the fast lane to perform an overtake. The objective of testing this scenario is to confirm the behaviour of the algorithm when a vehicle is moving through all the different defined zones of perception for the ego vehicle. The scenario can be seen in the image below:

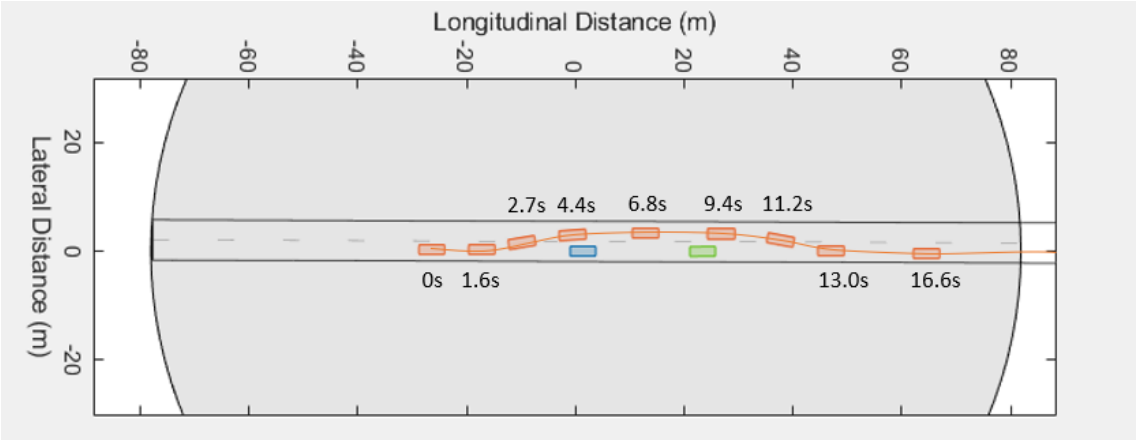

Figure 54. Testing driving scenario for a back overtaking situation.

The image below shows the mentioned signals. Change intention, for the sake of test clarity, has been forced to 1 from the beginning, by stating a desired speed higher than the vehicle in front. The back diagonal potential and the side potentials get activated in this case, varying according to the shape of the trajectory of the overtaking vehicle. The lane change confirmation signal behaves as expected. The front diagonal potential doesn't get activated as, as mentioned above, the code does not read, nor need, negative potentials, result of relative speeds being negative (vehicles moving away from the ego vehicle).

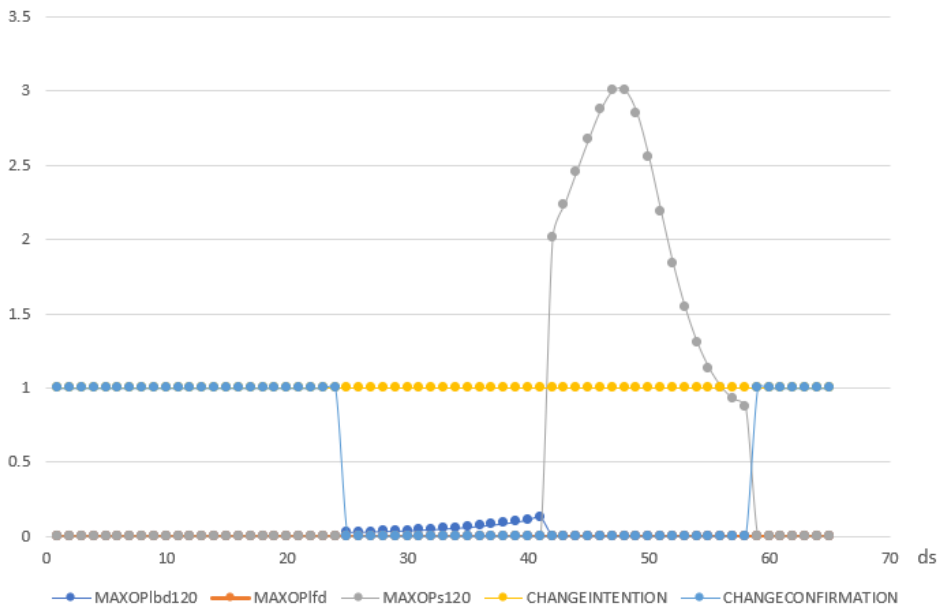

Figure 55. Obtained signals for the back overtaking tested scenario.

5.5 Traffic congestion situation

In last place, a traffic jam situation scenario was designed. After the variable intensity test, it is known that no lane change circumstance will be confirmed. However, in case a gap is created between vehicles, the algorithm should not cause a lane change if the traffic flow is faster in the occupied lane, regardless of it being theoretically the slow lane. This is important from the micro perspective of the autonomous vehicle user, but also from the macro perspective of the traffic flow, as unnecessary lane changes tend to negatively impact traffic congestion scenarios.

The following image shows a view of the 3D representation of the simulated traffic jam scenario:

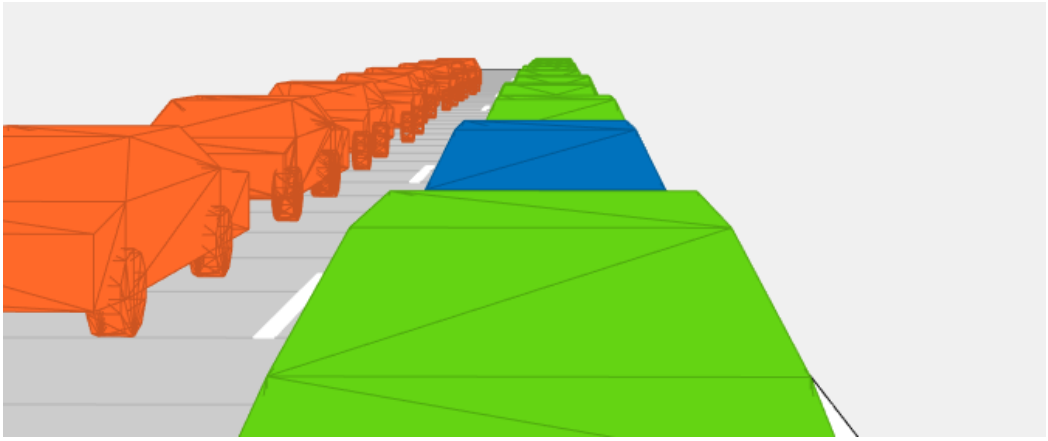

Figure 56. Ego-centric view of traffic jam simulation scenario.

The scenarios can be observed in the two images below. As situation of maximum density cannot be tested (160 veh/km per lane), as traffic speed in that case would be zero. Instead, the

fast lane (orange vehicles) has been granted with a traffic density of 140 veh/km (flow speed of 12.5 km/h) and the slow lane a traffic density of 155 veh/km (flow speed of 3.12 km/h).

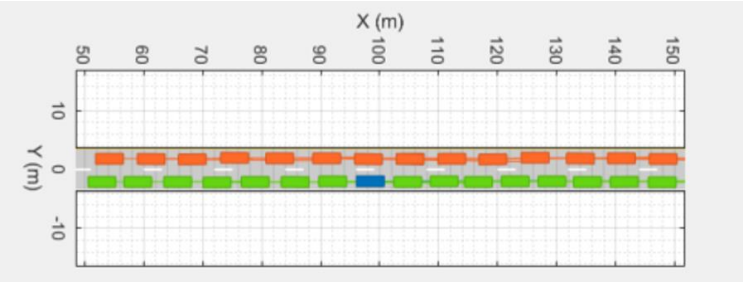

Figure 57. Testing driving scenario for a traffic congestion situation.

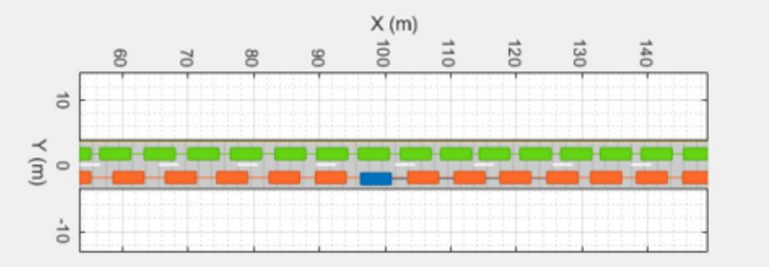

Figure 58. Testing driving scenario for traffic congestion situation with faster right lane.

The main outcome of the test is to observe that, when the ego vehicle (blue) is located in the fast lane, front diagonal potentials start being obtained, as the relative speed of the vehicles in the front diagonal becomes positive. Due to the lane change threshold for the front diagonal being considerably larger than the back one, it drastically reduces the chances of having a lane change confirmed in such situation. The signals obtained are displayed in the graphs below:

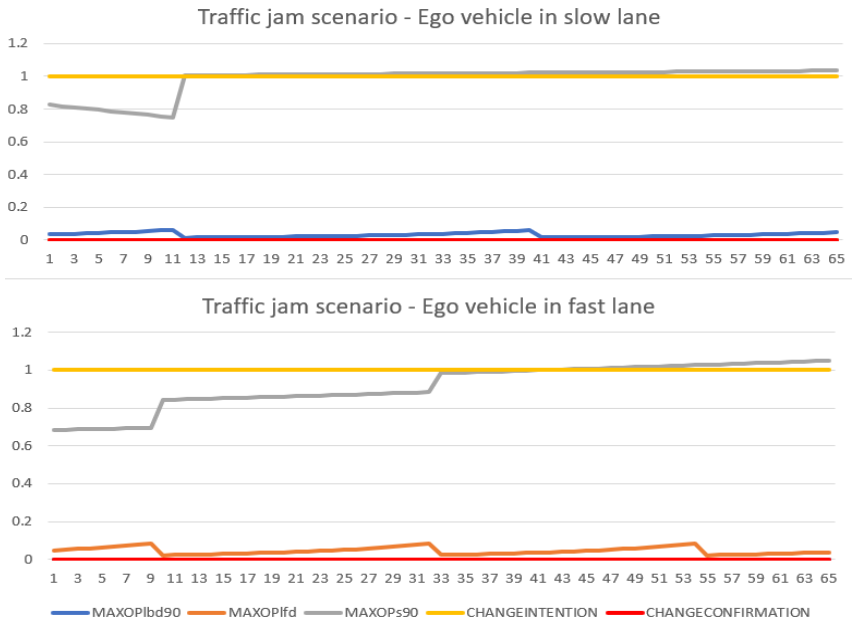

Figure 59. Output signals obtained for both tested traffic congestion scenario.

6 Conclusions and Future Work

This chapter concludes the realization of the project, by reviewing the level of completion of the objectives formulated at its start, followed by a discussion of results, and needed future development, towards a potential implementation of such decision-making algorithm.

This work has been concluded with the creation, coding, and testing of a deterministic decision-making algorithm for lane changing, based on a reactive model of surrounding agents' movement potentials. Deterministic, traditional algorithms have been considered to still outcome machine-learning stochastic models in decision-making robustness. The work presented has therefore prioritized lane-changing safety, sacrificing the versatility of artificial intelligence-based approaches. Within the logic used, a chain of safety confirmations has been established, analysing the motion of all the existing agents inside the field of view every one hundredth of a second, to ensure the feasibility of the lane-change. These confirmations are, likewise, defined after a model that considers not only distance to the surrounding agents, but also speed and locations. The potentials increase quadratically with the speed of the approaching agent, increasing its sensitivity to aggressive-driving users and lane-changing situations with high inter-lane speed differences.

Reactive models are not able to predict human behaviour, unlike cutting-edge NN based ones. This has a negative impact in the side of lane-changing efficiency. However, the use of a traditional algorithm ensures safety, while still representing a considerable increase of the speed of the decision-making process, compared to human driven vehicles. As described priorly, current functional limit of the code has been established in obtaining lane-change green light for traffic densities in the objective lane of 136 vehicles per kilometre, driving at a maximum speed of 15 km/h. Under these circumstances, an agent would be overtaking the agent every 2.2 seconds. Considering a human reaction time, from perception to controls actuation of two seconds, lane changes would not be viable under these conditions if it wasn't the case of an autonomous vehicle, where decisions are made every 1cs.

LiDAR perception has been considered since the concept generation phase. The result is an algorithm that ensures compatibility with the data obtained of these sensors. They generate point clouds that are then processed in real-time, by already established software, to eliminate irrelevant data and identify and track vehicles, clustering them into cuboids. This software returns the speed, dimensions, and orientation of such vehicles. The code has been written receiving these industry-established inputs, which were generated by a MATLAB tool that likewise considers LiDAR as the primary source of perception, delivering such inputs in the same data-frame format. Finally, it has also been proved that the decisions made can be done within a field of view of 80 meters, typical from standard LiDAR sensors used in the sector of autonomous driving.

A set of driving scenarios were created, representing realistic situations for the lane changing problem within the environment considered for the project. The tests conducted ranged from simple relative speeds tests (30 to 120 KPH), reverse relative speed in bidirectional roads, variable intensity car-following traffic scenarios, overtaking manoeuvres, and traffic congestion situations. Variable intensity scenarios were designed after the results obtained from a traffic Greenshields lineal model. Speed values were set to constant, assuming variable speeds definition unnecessarily increase complexity of the simulation. The trajectories of the vehicles were realistically manually drawn and smoothen by a jerk-limited trajectory function.

The testing phase enabled the refinement of the code and establishment of the decision-making potentials. It was found that potentials above 0.01 represent motions above the

dynamic minimum safety distance, setting it to 0.02 for motions in the front diagonal. Likewise, a functional limit, as above-mentioned, was obtained.

However, there is considerable work that has been delegated to further research.

Once an algorithm has been able to obtain successful results with ideal data; the next step towards a complete implementation would be generating point clouds corresponding to the created scenarios and recognizing and tracking actors (vehicles). This point cloud generation will correspond to programmatically placing theoretical LIDAR sensors in the simulate ego-vehicle, and the algorithm will be run with the point cloud as the only input, together with the already calibrated metrics. For this purpose, the Driving Scenario Designer Tool enables virtual point cloud generation of the different scenarios, once specified virtual sensors positions and specifications.

Likewise, it is needed to program different LiDAR configurations within the simulation, to obtain an optimum solution towards the functioning of the algorithm. A higher number of sensors placed on the ego vehicle will impact the quality of the point cloud. Likewise, the type of LIDAR will influence the reach of the perception and point cloud density. As the use of these sensors has also significant cost implications, the objective will be to define the smallest amount of devices with which the algorithm can normally function, as well as the optimum position in the vehicle.

A deeper analysis on LiDAR perception limitation needs to be done. Being perception systems limitations one of the main aspects to overcome within autonomous driving, the final outcome of this work will be comparing ideal-perception results (the ones obtained prior to the programming of the theoretical LIDAR) with the ones for the optimum LIDAR disposition, being able to analyse its limitations for lane-changing decision making. This will lead to a final assessment on the feasibility of using LIDAR as a single sensing unit.

7 Planning and budget

This work has been developed along three phases, consuming each of them an approximate of 30% of the total project time plan, starting with a research phase concerning general knowledge of intelligent transport systems, autonomous vehicles, and traffic theory. These studies were followed by a state-of-the-art information gathering regarding perception and location systems, focusing on LiDAR sensors and existing location technologies that backed up the assumption of knowing the position of lane markings. However, the biggest amount of research time was invested in compiling data belonging to up-to-date scientific articles in the field of lane changing and decision making, clearly differentiating between machine learning based works and traditional ones.

The second phase has been focused on the conceptual design of the decision-making algorithm, compatible with LiDAR perception and electronic maps for location. Given the large size of the subject, framing the problem to fit within the scope of a master thesis conformed the initial effort of this second phase, acting as a bridge between the research of the first phase and the conceptualization of the algorithm. This problem framing consisted of establishing an adequate simplified situation for the algorithm to perform, together with a number of assumptions. Next, the fundamental equations acting as the lowest-level decision-making mechanism were elaborated based in a model of potentials typically used for robot control, followed by the establishment of the basic logic driving the decision-making. This logic has been chosen mainly considering the needs and safety of the user in the chosen scenario of a straight section of two-lane highway.

The final phase was software-oriented, starting by the elaboration of a programmatic-level algorithm creation. This time, the logic was focused on the type of data being processed, the programming language to be used and the set of assumptions priorly established. Secondly, a considerable amount of time was invested in choosing a solution to generate the data the algorithm was going to process. This had to be planned in advance, as the programming task should be linked to the obtained format of data. Instead of programming the generation of the data, despite the flexibility it would imply, a Matlab pre-programmed tool was chosen instead, given the considerable shortening of the task it implied when creating the driving scenario and the generation of agents with its trajectories. A portion of the time was assigned to master the tool. Once a sample set of data was obtained in a Matlab dataset format, the first iteration of the code, following the logic of the programmatic-level algorithm, was written, and compiled. Debugging the code to properly access the necessary data through the different time frames implied most of the programming effort.

Still within the third phase, the practical part of the project was paused for an approximate of two months, focusing on report writing. Next, as a final step, a set of testing scenarios were conceived, linked to real lane-changing situations, with the aim of calibrating the decision-making variables, finding the functional limits of the code, and testing its behaviour under more specific situations. A portion of the specific situations had to be discarded (not described in this report) due to substantial compatibility problems with the code. For the rest of them, the code was adjusted accordingly. Finally, the obtained results were processed and analysed, the conclusions obtained can be found in the corresponding section of this text.

In addition, the preparation of the thesis report has consumed approximately 30% of the 19 months taken to execute the project. The Gant chart shown below shows the time consumed by every task. The content of the project, however, could have been developed within an approximation of four months, following a 40h weekly working hours scheme. Instead, it has been carried out in parallel with a full-time occupation, spending an average of 8.5 hours per week.

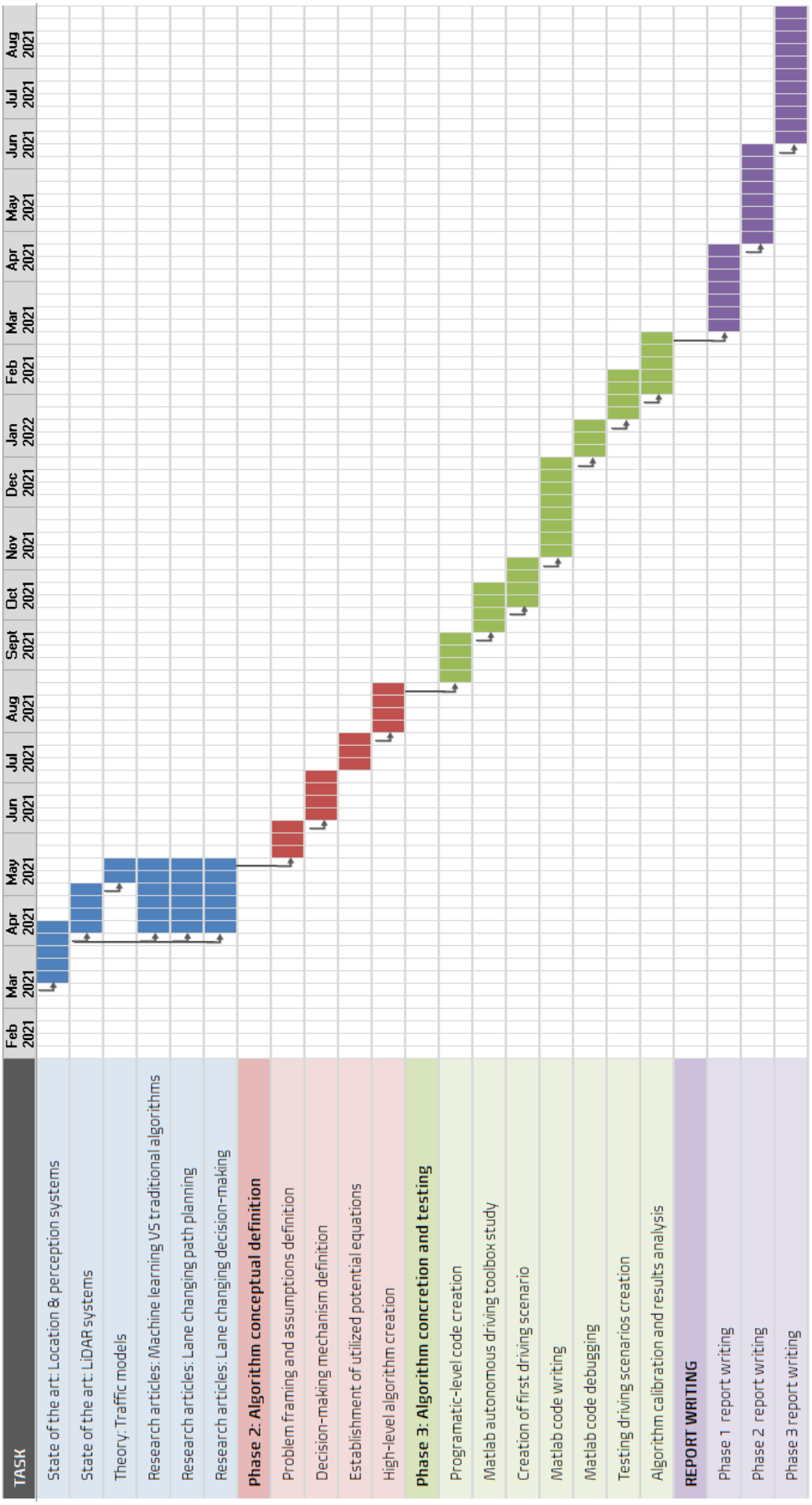

Figure 60. Project time plan.

Regarding project budgeting, being a project within the theoretical side of the autonomous driving problem and being carried out as a Master thesis project, using a personal computer, the cost results minimal. The expenses that can however be considered are the following:

Description	Quantity	Unit Cost (€)	Total Cost (€)
Operational costs			
Software licenses			
Matlab standard license (year)	2	49.00 €	98.00 €
Man labor			
Engineering professor wage (brute) (hrs)	40	35.00 €	1,400.00 €
Engineering student wage (brute) (hrs)	375.00	10.40 €	3,900.00 €
Fixed assets cost			
Hardware			
Corporate laptop: Thinkpad. Intel i5 processor (units) *5 years estimated lifespan	0.03	1,500.00 €	50.00 €
Personal headset: Logitec (units)	1	14.37 €	14.37 €
Personal mouse: Lenovo (units)	1	13.03 €	13.03 €
		TOTAL	**5,475.40 €**

8 References

[1] IEA, «Key World Energy Statistics 2020,» IEA, Paris, 2020.

[2] R. Kala, On-Road Intelligent Vehicles, Butterworth-Heinemann, 2016.

[3] O. Svenson, «Are we all less risky and more skillful than our fellow drivers?,» *Acta Psychologica,* vol. 47, nº 2, 1981.

[4] World Health Organisation, «Global Status Report on Road Safety 2018,» 2018.

[5] NHTSA's National Center for Statistics and Analysis , «Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey,» *TRAFFIC SAFETY FACTS,* 2015.

[6] U.S. Department of Transportation, «transportation.gov,» 1 August 2019. [En línea]. Available: https://www.transportation.gov/briefing-room/us-department-transportation-releases-policy-automated-vehicle-development. [Último acceso: 12 October 2021].

[7] Commitee on Traffic Flow Theory and Characteristics (AHB45), Traffic Flow Theory. A State-of-the-Art Report, Commitee on Traffic Flow Theory and Characteristics (AHB45), 2021.

[8] S. Hoogendoorn y V. Knoop, Traffic flow theory and modelling, in the Transport System and Transport Policy, 2013.

[9] H. Wang, J. Li, Q.-Y. Chen y N. Daiheng, «Logistic modeling of the equilibrium speed–density relationship,» *Transportation Research Part A,* nº 45, pp. 554-566, 2011.

[10] J. J. Olstam y A. Tapani, Comparison of Car-following models, Swedish National Road and Transport Research Institute, 2004.

[11] A. Perallos, U. Hernandez-Jayo, E. Onieva y I. García-Zuazola, Intelligent Transport Systems. Technologies and applications, John Wiley & Sons, Ltd., 2016.

[12] S. A. Shaheen, «Intelligent Transportation Systems,» de *Reference Module in Earth Systems and Environmental Sciences*, 2013.

[13] R. Kala, «Basics of Autonomous Vehicles,» de *On-Road Intelligent Vehicles*, Elsevier, 2016.

[14] T. Yu, H. Huang, N. Jiang y T. D. Acharya, «Study on Relative Accuracy and Verification Method of High-Definition Maps for Autonomous Driving,» *International Journal of Geo-Information,* 2021.

[15] V. Ilci y C. Toth, «High Definition 3D Map Creation Using GNSS/IMU/LiDAR Sensor Integration to Support Autonomous Vehicle Navigation,» *Sensors,* 2019.

[16] José M.Armingol, et. al., «Environmental Perception for Intelligent Vehicles,» de *Intelligent Vehicles. Enabling Technologies*, Elsevier, 2018.

[17] A. Chen y C. Asawa, «Going beyond the bounding box with semantic segmentation,» The Gradient, 2018.

[18] Anolytics.ai, *How to Improve Computer Vision in Autonomous Vehicles using Image Annotation Services?,* https://www.anolytics.ai/blog/how-to-improve-autonomous-vehicles-image-annotation/, 2020.

[19] J. Kocić, N. Jovičić y V. Drndarević, «Sensors and Sensor Fusion in Autonomous Vehicles,» *26th Telecommunications forum TELFOR,* 2018.

[20] C. Hill, «Coherent Focused Lidars for Doppler Sensing of Aerosols and Wind,» *Remote Sensing,* 2018.

[21] J. Jung, E. Che, M. J. Olsen y C. Parrish, «Efficient and robust lane marking extraction from mobile lidar point clouds,» *ISPRS Journal of Photogrammetry and Remote Sensing,* 2019.

[22] C. Hubmann, M. Becker, D. Althoff, D. Lenz y C. Stiller, «Decision Making for Autonomous Driving considering Interaction and Uncertain Prediction of Surrounding Vehicles,» *IEEExplore,* 2017.

[23] C.-J. Hoel, K. Wolff y L. Laine, «Tactical Decision-Making in Autonomous Driving by Reinforcement Learning with Uncertainty Estimation,» *arXiv,* 2020.

[24] F. Leon y M. Gavrilescu, «A Review of Tracking, Prediction and Decision Making Methods for Autonomous Driving,» *Technical University of Iasi,* 2019.

[25] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare y J. Pineau, «An Introduction to Deep Reinforcement Learning,» *arXiv,* 2018.

[26] S. James, G. Konidaris y B. Rosman, «An Analysis of Monte Carlo Tree Search,» *Conference: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17),* 2017.

[27] K. Bin Isa y A. Bin Jantan, «An Autonomous Vehicle Driving Control System,» *TEMPUS Publications,* 2005.

[28] C. Martínez y F. Jiménez, «Implementation of a Potential Field-Based Decision-Making Algorithm on Autonomous Vehicles for Driving in Complex Environments,» *Sensors,* 2019.

[29] R. Kala, «Potential-Based Planning,» de *On-Road Intelligent Vehicles*, Elsevier, 2016.

[30] R. Roriz, J. Cabral y T. Gomes, «Automotive LiDAR Technology: A Survey».

[31] B. Behroozpour, P. A. M. Sandborn, M. C. Wu y B. E. Boser, «Lidar System Architectures and Circuits,» *IEEE Communications Magazine,* 2017.

[32] C. Rablau, «Lidar: a new self-driving vehicle for introducing optics to broader engineering and non-engineering audiences,» *Fifteenth Conference on Education and Training in Optics and,* 2019.

[33] Z. Wang, X. Zhao, Xu Zhigang, X. Li y X. Qu, «Modeling and field experiments on autonomous vehicle lane changing with surrounding human-driven vehicles,» *Computer-Aided Civil and Infrastructure Engineering,* nº 36, pp. 877-889, 2020.

[34] M. Rahman, M. Chowdhury, S. Member, Y. Xie y Y. He, «Review of Microscopic Lane-Changing Models and Future Research Opportunities,» *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,* vol. 14, nº 4, 2013.

[35] S. Moridpour, M. Sarvi y G. Rose, «Lane Changing Models: A Critical Review,» *The International Journal of Transportation Research,* 2010.

[36] D. Yang, S. Zheng, C. Wen, P. J. Jin y B. Ran, «A dynamic lane-changing trajectory planning model for automated vehicles,» *Transportation Research Part C,* pp. 228-247, 2018.

[37] D. Chen, L. Jiang, Y. Wang y Z. Li, «Autonomous Driving using Safe Reinforcement Learning

by Incorporating a Regret-based Human Lane-Changing Decision Model,» *2020 American Control Conference,* 2020.

[38] B. Mirchevska, C. Pek, M. Werling, M. Althoff y J. Boedecker, «High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning,» *21st International Conference on Intelligent Transportation Systems (ITSC),* 2018.

[39] W. Wang, T. Qie, C. Yang, W. Liu, C. Xiang y K. Huang, «An Intelligent Lane-Changing Behavior Prediction and Decision-Making Strategy for an Autonomous Vehicle,» *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS,* vol. 69, nº 3, 2022.

[40] V. G. Lopez, F. L. L. Lewis, M. Liu, Y. Wan, S. Nageshrao y D. Filev, «Game-Theoretic Lane-Changing Decision Making and Payoff Learning for Autonomous Vehicles,» *IEEE Transactions on Vehicular Technology,* 2022.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra y M. Riedmiller, «Playing Atari with Deep Reinforcement Learning,» *DeepMind Technologies,* 2013.

[42] J. NIE, J. ZHANG, W. DING, X. WAN, X. CHEN y B. RAN, «Decentralized Cooperative Lane-Changing Decision-Making for Connected Autonomous Vehicles,» *IEEEAccess,* 2016.

[43] Z. Wang, X. Zhao, Z. Xu, X. Li y X. Qu, «Modeling and field experiments on autonomous vehicle lane changing with surrounding human-driven vehicles,» *Computer-Aided Civil and Infrastructure Engineering,* 2020.

Appendix A. Decision-making algorithm code

```
%--------------------------------------------------------------------------
%DECISION MAKING ALGORITHM FOR LANE-CHANGING BASED ON LIDAR PERCEPTION
%--------------------------------------------------------------------------


% 1. VEHICLE DATA LOAD
%---------------------------------
% Vehicle data the algorithm is loaded using the function described in the
% functions file:

[allData, scenario, sensor] = SCRIPT_TestingDS4_BackOvertaking();


% 2. INITIALIZATION OF LOCAL VARIABLES
%-----------------------------------------


t0 = 0; %[s] Time value storage.
Vb1 = 0; %[m/s] Speed of the vehicle in front.
Vdes = 33.33; %[m/s] Desired speed of the ego vehicle.
t_clock = 0; %[s] Current time during data analysis.
t_wait = 0; %[s] Waiting time until change intention is activated.
Yleftboarder = 3.6; %[m]Road left boarder y position (not exported).
Yrightboarder = -3.6; %[m] Road right boarder y position (not exported).
% d_safety = scenario.Actors(1,sensor.EgoVehicleActorID).Length; % [m]Safety
distance between vehicles.
d_safety = 200; %DELETE AFTER TRIALS
Xb1 = 0; %Front distance to closest vehicle in front of ego vehicle.
SIDECHECK = false; %Safety check prior to change confirmation.
DIAGONALBACKCHECK = false; %Safety check prior to change confirmation.
DIAGONALFRONTCHECK = false; %Safety check prior to change confirmation.
SizeofScenario.Actors = size(scenario.Actors); %Extraction of matrix
containing number of actors
Posvehinfront = zeros(SizeofScenario.Actors(1,2),1); %Initialization of
variable containing position of actors in front of ego vehicle
Lb = sensor.SensorLocation(1) +
scenario.Actors(1,sensor.EgoVehicleActorID).RearOverhang;
Lf = scenario.Actors(1,sensor.EgoVehicleActorID).Length -
(sensor.SensorLocation(1) +
scenario.Actors(1,sensor.EgoVehicleActorID).RearOverhang);

%COUNTERS:
k = 0;
l = 0;

%CALIBRATION THRESHOLDS:
OPlbdlim = 0.02;
OPlfdlim = 0.01;

% 3. DECISION MAKING. GENERATION OF CHANGE CONFIRMATIONS.
%------------------------------------------------------------

for i = allData
    %Temporal duration of the ego vehicle travel
    l = l+1;
    t_clock = allData(l).Time;
    for j = 1:SizeofScenario.Actors(1,2)
    k = k+1;
        if k ~= sensor.EgoVehicleActorID
            if allData(l).ActorPoses(k).Position(1) -
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(1) > Lf %If
considered actor is in front of ego vehicle (LIDAR position)
```

```
                if allData(l).ActorPoses(k).Position(2) < 0 &&
allData(l).ActorPoses(k).Position(2) > -3.6  %If considered actor is inside
the lane of ego vehicle
                    allData(l).ActorPoses(k).Vehicleinfront = true; %A column
in the struct is created identifying if the actor considered is in front of
the ego vehicle
                else
                    allData(l).ActorPoses(k).Vehicleinfront =
false; %Otherwise it is asumed it is not in front
                end
            else
                allData(l).ActorPoses(k).Vehicleinfront = false; %Otherwise it
is asumed it is not in front
            end
        end
    end
    k = 0;
    for j = 1:SizeofScenario.Actors(1,2) %Scanning array of actors in
timestamp
        k = k+1;
        if k ~= sensor.EgoVehicleActorID %To not consider Ego vehicle
            if allData(l).ActorPoses(k).Vehicleinfront == true %If the actor
has been addressed to be in front of the ego vehicle
                Posvehinfront(k) = allData(l).ActorPoses(k).Position(1); %Its
position is recorded for comparison in next irteration of the loop
            else
                Posvehinfront(k) = 100000000; %Asigning a high value to be
able to extract the minimum after
            end
        else
            Posvehinfront(k) = 100000000; %Asigning a high value to be able to
extract the minimum after
        end
    end
    k = 0;
    for j = 1:SizeofScenario.Actors(1,2) %Scanning array of actors in
timestamp
        k = k+1;
        if allData(l).ActorPoses(k).Position(1) == min(Posvehinfront) &&
allData(l).ActorPoses(k).Vehicleinfront == true  %Consider only the closest
actor in front
             Vb1 = allData(l).ActorPoses(k).Velocity(1); %Record its speed
             Xb1 = allData(l).ActorPoses(k).Position(1); %Record its position
             if Vb1 < Vdes && (Xb1 -
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(1))<= d_safety %If
its speed in that timestamp is lower than the one desired and close enough to
ego vehicle
                 t0 = t0 + t_clock; %Waiting time increases
             else
                 t0 = 0; %Waiting time restarts
             end
        end
    end
    k = 0;
    if t0 >= t_wait %If waiting time exceeds the wait limit
             allData(l).Changeintention = 1; %Value of 1 asigned to variable
to plot afirmative change intention timestamps
             Changeintention(l) = 1;
             for j = 1:SizeofScenario.Actors(1,2) %Scanning array of actors in
timestamp
                 k = k+1;
                 if allData(l).ActorPoses(k).Position(2) >= 0 % If vehicle is
on the left lane
                     if allData(l).ActorPoses(k).Position(1) -
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(1) >= 0 %If X
location of actor is positive (relatively to ego vehicle)
                         if (allData(l).ActorPoses(k).Position(1) -
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(1)) -
```

```
(scenario.Actors(1,k).Length/2) <= Lf %And if its position is from 0 to the
length of the ego vehicle
                            allData(l).ActorPoses(k).Vehicleonside =
true; %The vehicle is confirmed to be on the side
                            allData(l).ActorPoses(k).OPs =
1/(allData(l).ActorPoses(k).Position(2) +
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(2) -
(scenario.Actors(1,k).Width/2))^2; %Its potential is calculated as a function
of the y distance to the ego vehicle
                        else
                            allData(l).ActorPoses(k).OPs = 0;
                        end
                    else %If X location of actor is negative (relatively to
ego vehicle)
                        if (allData(l).ActorPoses(k).Position(1) -
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(1)) +
(scenario.Actors(1,k).Length/2) > -Lb%And if its position is from 0 to (-) the
length of the ego vehicle
                            allData(l).ActorPoses(k).Vehicleonside =
true; %The vehicle is confirmed to be on the side
                            allData(l).ActorPoses(k).OPs =
1/(allData(l).ActorPoses(k).Position(2) +
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(2) -
(scenario.Actors(1,k).Width/2))^2;%Its potential is calculated as a function
of the y distance to the ego vehicle
                        else
                            allData(l).ActorPoses(k).OPs = 0;
                        end
                    end
                else
                    allData(l).ActorPoses(k).Vehicleonside = false;
                    allData(l).ActorPoses(k).OPs = 0;
                end
                Sidepotentials(k) = allData(l).ActorPoses(k).OPs;
            end
            k = 0;
            MAXOPs(l) = max(Sidepotentials);
            OPs_lim = 1/(Yleftboarder +
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(2))^2;%Limit side
potential is calculated as a function of the distance to the left boarder of
the road
            if MAXOPs(l) < OPs_lim %If the maximum calculated value of OPs is
less than the safety limit
                SIDECHECK(l) = true;
            else
                SIDECHECK(l) = false;
            end
            for j = 1:SizeofScenario.Actors(1,2) %Scanning array of actors in
timestamp
                k = k+1;
                if (allData(l).ActorPoses(k).Position(1) -
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(1) +
scenario.Actors(1,k).Length/2) < -Lb %If actor is situated behind the ego
vehicle
                    if allData(l).ActorPoses(k).Position(2) -
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(2) >
scenario.Actors(1,sensor.EgoVehicleActorID).Width/2 %And it is placed to its
left side
                        if (allData(l).ActorPoses(k).Position(2) -
scenario.Actors(1,k).Width/2) < Yleftboarder %And its placed in the next lane
to the left side
                            allData(l).ActorPoses(k).Vehicleinbackdiagonal =
true; % Actor is designated to be located in the back diagonal
                            allData(l).ActorPoses(k).OPlbd =
(allData(l).ActorPoses(k).Velocity(1)-
allData(l).ActorPoses(sensor.EgoVehicleActorID).Velocity(1))/(sqrt((allData(l)
.ActorPoses(sensor.EgoVehicleActorID).Position(1)-
allData(l).ActorPoses(k).Position(1))^2+(allData(l).ActorPoses(k).Position(2)-
```

```
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(2))^2))^2; %Its
potential is calculated as a function of the distance to the ego vehicle and
its speed
                            else
                                allData(l).ActorPoses(k).Vehicleinbackdiagonal =
false;
                                allData(l).ActorPoses(k).OPlbd = 0;
                            end
                        else
                            allData(l).ActorPoses(k).Vehicleinbackdiagonal =
false;
                            allData(l).ActorPoses(k).OPlbd = 0;
                        end
                    else
                        allData(l).ActorPoses(k).Vehicleinbackdiagonal = false;
                        allData(l).ActorPoses(k).OPlbd = 0;
                    end
                    Backdiagonalpotentials(k) = allData(l).ActorPoses(k).OPlbd;
                end
                k = 0;
                MAXOPlbd(l) = max(Backdiagonalpotentials);
                if MAXOPlbd(l) <= OPlbdlim %If there is a vehicle with back
diagonal potential means its in the range of the lidar and getting closer,
otherwise:
                    DIAGONALBACKCHECK(l) = true; %Safety check is confirmed (less
restrictive threshhold can substitute 0 after calibration)
                else
                    DIAGONALBACKCHECK(l) = false; %Safety check not confirmed
                end
                for j = 1:SizeofScenario.Actors(1,2) %Scanning array of actors in
timestamp
                    k = k+1;
                    if (allData(l).ActorPoses(k).Position(1) -
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(1) -
scenario.Actors(1,k).Length/2) > Lf %If actor is situated in front of the ego
vehicle
                        if allData(l).ActorPoses(k).Position(2) -
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(2) >
scenario.Actors(1,sensor.EgoVehicleActorID).Width/2 %And it is placed to its
left side
                            if (allData(l).ActorPoses(k).Position(2) -
scenario.Actors(1,k).Width/2) < Yleftboarder %And its placed in the next lane
to the left side
                                allData(l).ActorPoses(k).Vehicleinfrontdiagonal =
true; % Actor is designated to be located in the front diagonal
                                allData(l).ActorPoses(k).OPlfd =
(allData(l).ActorPoses(sensor.EgoVehicleActorID).Velocity(1)-
allData(l).ActorPoses(k).Velocity(1))/(sqrt((allData(l).ActorPoses(k).Position
(1)-
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(1))^2+(allData(l).Act
orPoses(k).Position(2)-
allData(l).ActorPoses(sensor.EgoVehicleActorID).Position(2))^2))^2; %Its
potential is calculated as a function of the distance to the ego vehicle and
its speed
                            else
                                allData(l).ActorPoses(k).Vehicleinfrontdiagonal =
false;
                                allData(l).ActorPoses(k).OPlfd = 0;
                            end
                        else
                            allData(l).ActorPoses(k).Vehicleinfrontdiagonal =
false;
                            allData(l).ActorPoses(k).OPlfd = 0;
                        end
                    else
                        allData(l).ActorPoses(k).Vehicleinfrontdiagonal = false;
                        allData(l).ActorPoses(k).OPlfd = 0;
                    end
```

```
                Frontdiagonalpotentials(k) = allData(l).ActorPoses(k).OPlfd;
            end
            k = 0;
            MAXOPlfd(l) = max(Frontdiagonalpotentials);
            if MAXOPlfd(l) <= OPlfdlim %If there is a vehicle with front
diagonal potential means its in the range of the lidar and getting closer,
otherwise:
                DIAGONALFRONTCHECK(l) = true; %Safety check is confirmed
(less restrictive threshhold can substitute 0 after calibration)
            else
                DIAGONALFRONTCHECK(l) = false; %Safety check not confirmed
            end
            if SIDECHECK(l) && DIAGONALBACKCHECK(l) &&
DIAGONALFRONTCHECK(l) %If all checks are confirmed
                allData(l).Changeconfirmation = 1; %vehicle has green light
to start lane changing. Value of one in the timestamp is asigned for
representation
                Changeconfirmation(l) = 1;
            else
                allData(l).Changeconfirmation = 0; %Change confirmation not
achieved in the timestamp. Value of 0 asigned for representation
                Changeconfirmation(l) = 0;
            end
    else
        allData(l).Changeintention = 0; %No intention/need to change. Value of
0 asigned for representation
        Changeintention(l) = 0;
    end
end


% 4. STATUS GRAPHIC REPRESENTATION
%---------------------------------
% Representation of the decision making status through the simulation time

figure(1);
hold on;
grid on;

nexttile([1 3])
plot(Changeintention);
xlim([0 1]);
title('Change intention status activation')
xlabel('time (ds)')
ylabel('Changeintention')

nexttile([1 3])
plot(Changeconfirmation);
xlim([0 1]);
title('Change confirmation status confirmation')
xlabel('time (ds)')
ylabel('Changeconfirmation')

nexttile([1 3])
plot(SIDECHECK);
xlim([0 1]);
title('Side check status confirmation')
xlabel('time (ds)')
ylabel('SIDECHECK')

nexttile([1 3])
plot(DIAGONALBACKCHECK);
xlim([0 1]);
title('Back diagonal check status confirmation')
xlabel('time (ds)')
ylabel('DIAGONALBACKCHECK')

nexttile([1 3])
```

```
plot(DIAGONALFRONTCHECK);
xlim([0 1]);
title('Front diagonal check status confirmation')
xlabel('time (ds)')
ylabel('DIAGONALFRONTCHECK')

nexttile([1 3])
plot(MAXOPs);
xlim([0 1]);
title('Side potential')
xlabel('time (ds)')
ylabel('Potential')

nexttile([1 3])
plot(MAXOPlfd);
xlim([0 1]);
title('Front diagonal potential')
xlabel('time (ds)')
ylabel('Potential')

nexttile([1 3])
plot(MAXOPlbd);
xlim([0 1]);
title('Back diagonal potential')
xlabel('time (ds)')
ylabel('Potential')


hold off
```

Appendix B. Simulation scenario generation code

```
%-------------------------------------------------------------------------
%DECISION MAKING ALGORITHM FUNCTIONS
%-------------------------------------------------------------------------

% 1. VEHICLE DATA LOAD
%---------------------------------
% Vehicle data the algorithm is based on is generated through the Matlab
% aplication Driving scenario Designer. The code exported from the
% scneraio created is shown below. It includes the creation of the geometry
% of the road segment, vehicles and their trayectories (actors), parameters
% of the LIDAR sensor placed on the ego vehicle (sensor)and the
% corresponding point cloud.

function [allData, scenario, sensor] = Decision_making_algorithm_functions()
%DrivingScenarioSimulation2LineHighway200MSegment - Returns sensor detections
%    allData = DrivingScenarioSimulation2LineHighway200MSegment returns sensor
detections in a structure
%    with time for an internally defined scenario and sensor suite.
%
%    [allData, scenario, sensors] =
DrivingScenarioSimulation2LineHighway200MSegment optionally returns
%    the drivingScenario and detection generator objects.

% Generated by MATLAB(R) 9.10 (R2021a) and Automated Driving Toolbox 3.3
(R2021a).
% Generated on: 10-Aug-2021 12:04:31

% Create the drivingScenario object and ego car
[scenario, egoVehicle] = createDrivingScenario;

% Create all the sensors
sensor = createSensor(scenario);

allData = struct('Time', {}, 'ActorPoses', {}, 'ObjectDetections', {},
'LaneDetections', {}, 'PointClouds', {}, 'INSMeasurements', {});
running = true;
while running

    % Generate the target poses of all actors relative to the ego vehicle
    poses = targetPoses(egoVehicle);
    time  = scenario.SimulationTime;

    % Generate detections for the sensor
    laneDetections = [];
    objectDetections = [];
    insMeas = [];
    if sensor.HasRoadsInputPort
        rdmesh = roadMesh(egoVehicle,min(500,sensor.MaxRange));
        [ptClouds, isValidPointCloudTime] = sensor(poses, rdmesh, time);
    else
        [ptClouds, isValidPointCloudTime] = sensor(poses, time);
    end

    % Aggregate all detections into a structure for later use
    if isValidPointCloudTime
        allData(end + 1) = struct( ...
            'Time',       scenario.SimulationTime, ...
            'ActorPoses', actorPoses(scenario), ...
            'ObjectDetections', {objectDetections}, ...
            'LaneDetections', {laneDetections}, ...
            'PointClouds',   {ptClouds}, ... %#ok<AGROW>
            'INSMeasurements',   {insMeas}); %#ok<AGROW>
    end
```

```
    % Advance the scenario one time step and exit the loop if the scenario is
complete
    running = advance(scenario);
end

% Restart the driving scenario to return the actors to their initial positions.
restart(scenario);

% Release the sensor object so it can be used again.
release(sensor);
end

%%%%%%%%%%%%%%%%%%%%
% Helper functions %
%%%%%%%%%%%%%%%%%%%%

% Units used in createSensors and createDrivingScenario
% Distance/Position - meters
% Speed             - meters/second
% Angles            - degrees
% RCS Pattern       - dBsm

function sensor = createSensor(scenario)
% createSensors Returns all sensor objects to generate detections

% Assign into each sensor the physical and radar profiles for all actors
profiles = actorProfiles(scenario);
sensor = lidarPointCloudGenerator('SensorIndex', 1, ...
    'UpdateInterval', 0.01, ...
    'SensorLocation', [1.9 0], ...
    'MaxRange', 80, ...
    'ElevationLimits', [-15 15], ...
    'ActorProfiles', profiles);
end

function [scenario, egoVehicle] = createDrivingScenario
% createDrivingScenario Returns the drivingScenario defined in the Designer

% Construct a drivingScenario object.
scenario = drivingScenario;

% Add all road segments
roadCenters = [0 0 0;
    200 0 0];
laneSpecification = lanespec(2);
road(scenario, roadCenters, 'Lanes', laneSpecification, 'Name', 'Road');

% Add the ego vehicle
egoVehicle = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [26.953502590134 -2.56922384789884 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'Name', 'Car');
waypoints = [26.953502590134 -2.56922384789884 0;
    36.2 -2.1 0;
    44.1 -2.2 0;
    53.3 -1.9 0;
    63.9 -1.6 0;
    71.1 -1.8 0;
    78.4 -1.8 0;
    86.5 -1.9 0;
    94.4 -1.8 0;
    103.9 -1.8 0;
    115.4 -2 0;
    126.1 -1.8 0;
    135.5 -1.5 0;
    145.2 -1.3 0;
    156.4 -1.3 0;
```

```
    165.6 -1.5 0;
    175.6 -1.3 0;
    187 -1.3 0;
    200 -1.5 0];
speed = [30;30;30;30;30;30;30;30;30;30;30;30;30;30;30;30;30;30;30;30;30];
trajectory(egoVehicle, waypoints, speed);

% Add the non-ego actors
car1 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [45.2 -1.9 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car1');
waypoints = [45.2 -1.9 0;
    199.5 -1.4 0];
speed = [29;29];
trajectory(car1, waypoints, speed);

car2 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [10.2 -1.6 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car2');
waypoints = [10.2 -1.6 0;
    18.1 -1.8 0;
    25 -1.8 0;
    32.9 -1.8 0;
    40.4 -1.8 0;
    47.8 -1.8 0;
    55.7 -2 0;
    61.8 -2 0;
    68.3 -1.8 0;
    75.3 -2.1 0;
    82.2 -2.1 0;
    89.7 -1.7 0;
    98.6 -1.7 0;
    108.1 -1.6 0;
    118.8 -1.6 0;
    129.5 -1.5 0;
    138.3 -1.7 0;
    147.5 -1.6 0;
    159 -1.4 0;
    167.8 -1.6 0;
    177.9 -1.4 0;
    188.9 -1.5 0;
    200.4 -1.6 0];
speed =
[31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31;31];
trajectory(car2, waypoints, speed);

car3 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [43 2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car3');
waypoints = [43 2 0;
    199.9 1.7 0];
speed = [36;36];
trajectory(car3, waypoints, speed);

car4 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [28.3 2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
```

```
    'Name', 'Car4');
waypoints = [28.3 2 0;
    37.3 2.6 0;
    45.7 2.6 0;
    53.1 2.6 0;
    62.3 2.4 0;
    72 2.1 0;
    83.2 2.4 0;
    93.9 2.1 0;
    103.1 2.1 0;
    113.3 2.4 0;
    123 2.4 0;
    134.5 1.9 0;
    145.7 2.1 0;
    157.2 2.4 0;
    166.9 2.4 0;
    176.8 2.6 0;
    185.8 2.1 0;
    196.2 2.6 0;
    200.5 2.4 0];
speed = [34;34;34;34;34;34;34;34;34;34;34;34;34;34;34;34;34;34;34];
trajectory(car4, waypoints, speed);

car5 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [10.2 2.2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car5');
waypoints = [10.2 2.2 0;
    19.4 2.12 0.01;
    26.17 2 0.01;
    34.05 2.32 0.01;
    40.71 2.26 0.01;
    49.8 2.2 0;
    57 2.4 0;
    65.9 2.2 0;
    74.5 2.1 0;
    82.2 2 0;
    91.1 1.9 0;
    106.2 2 0;
    115.8 2 0;
    125.1 2 0;
    137.1 1.7 0;
    148.6 1.9 0;
    160.2 1.7 0;
    169.7 1.7 0;
    179 1.9 0;
    188.4 1.8 0;
    194.9 1.9 0;
    201.6 2 0];
speed = [32;32;32;32;32;32;32;32;32;32;32;32;32;32;32;32;32;32;32;32;32;32;32];
trajectory(car5, waypoints, speed);

end


% TEST DATA GENERATION: RelativeSpeed30KPH
%-----------------------------------------

function [allData, scenario, sensor] = SCRIPT_TestingDS1_RelativeSpeed30KPH()
%TestingDS1_RelativeSpeed30KPH - Returns sensor detections
%    allData = TestingDS1_RelativeSpeed30KPH returns sensor detections in a
structure
%    with time for an internally defined scenario and sensor suite.
%
%    [allData, scenario, sensors] = TestingDS1_RelativeSpeed30KPH optionally
returns
```

```
%    the drivingScenario and detection generator objects.

% Generated by MATLAB(R) 9.10 (R2021a) and Automated Driving Toolbox 3.3
(R2021a).
% Generated on: 13-Aug-2022 19:33:11

% Create the drivingScenario object and ego car
[scenario, egoVehicle] = createDrivingScenario;

% Create all the sensors
sensor = createSensor(scenario);

allData = struct('Time', {}, 'ActorPoses', {}, 'ObjectDetections', {},
'LaneDetections', {}, 'PointClouds', {}, 'INSMeasurements', {});
running = true;
while running

    % Generate the target poses of all actors relative to the ego vehicle
    poses = targetPoses(egoVehicle);
    time  = scenario.SimulationTime;

    % Generate detections for the sensor
    laneDetections = [];
    objectDetections = [];
    insMeas = [];
    if sensor.HasRoadsInputPort
        rdmesh = roadMesh(egoVehicle,min(500,sensor.MaxRange));
        [ptClouds, isValidPointCloudTime] = sensor(poses, rdmesh, time);
    else
        [ptClouds, isValidPointCloudTime] = sensor(poses, time);
    end

    % Aggregate all detections into a structure for later use
    if isValidPointCloudTime
        allData(end + 1) = struct( ...
            'Time',       scenario.SimulationTime, ...
            'ActorPoses', actorPoses(scenario), ...
            'ObjectDetections', {objectDetections}, ...
            'LaneDetections', {laneDetections}, ...
            'PointClouds',   {ptClouds}, ... %#ok<AGROW>
            'INSMeasurements',   {insMeas}); %#ok<AGROW>
    end

    % Advance the scenario one time step and exit the loop if the scenario is
complete
    running = advance(scenario);
end

% Restart the driving scenario to return the actors to their initial positions.
restart(scenario);

% Release the sensor object so it can be used again.
release(sensor);

%%%%%%%%%%%%%%%%%%%%
% Helper functions %
%%%%%%%%%%%%%%%%%%%%

% Units used in createSensors and createDrivingScenario
% Distance/Position - meters
% Speed             - meters/second
% Angles            - degrees
% RCS Pattern       - dBsm

function sensor = createSensor(scenario)
% createSensors Returns all sensor objects to generate detections

% Assign into each sensor the physical and radar profiles for all actors
```

```
profiles = actorProfiles(scenario);
sensor = lidarPointCloudGenerator('SensorIndex', 1, ...
    'UpdateInterval', 0.01, ...
    'SensorLocation', [1.9 0], ...
    'MaxRange', 80, ...
    'ElevationLimits', [-15 15], ...
    'ActorProfiles', profiles);

function [scenario, egoVehicle] = createDrivingScenario
% createDrivingScenario Returns the drivingScenario defined in the Designer

% Construct a drivingScenario object.
scenario = drivingScenario;

% Add all road segments
roadCenters = [0 0 0;
    1000 0 0];
laneSpecification = lanespec(2);
road(scenario, roadCenters, 'Lanes', laneSpecification, 'Name', 'Road');

% Add the ego vehicle
egoVehicle = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [90 -1.95853652867245 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'Name', 'Car');
waypoints = [90 -1.95853652867245 0;
    159.4 -2 0;
    404.8 -1.5 0;
    666.1 -1.6 0;
    814.1 -1.6 0;
    919.5 -0.8 0;
    994.4 -1.2 0];
speed = [22.22;22.22;22.22;22.22;22.22;22.22;22.22];
waittime = [0;0;0;0;0;0;0];
trajectory(egoVehicle, waypoints, speed, waittime);

% Add the non-ego actors
car1 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [109.432586974582 -1.94694709135225 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car1');
waypoints = [109.432586974582 -1.94694709135225 0;
    132.2 -1.9 0;
    218.4 -1.8 0;
    325.2 -1.7 0;
    420.7 -2 0;
    556.3 -1.9 0;
    685 -2 0;
    845.2 -2 0;
    959 -2.2 0;
    994.9 -1.9 0];
speed = [21.94;21.94;21.94;21.94;21.94;21.94;21.94;21.94;21.94;21.94];
waittime = [0;0;0;0;0;0;0;0;0;0];
trajectory(car1, waypoints, speed, waittime);

car5 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [4.75145285889966 1.59879061177704 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car5');
waypoints = [4.75145285889966 1.59879061177704 0;
    56.3 1.6 0;
    106.1 1.9 0;
    150.6 1.6 0;
```

```
    256.2 1.5 0;
    330 1.9 0;
    542.8 2.1 0;
    767.8 1.9 0;
    993.2 2.2 0];
speed = [30.55;30.55;30.55;30.55;30.55;30.55;30.55;30.55;30.55];
trajectory(car5, waypoints, speed);



% TEST DATA GENERATION: ReverseRelativeSpeed170KPH
%------------------------------------------------


function [allData, scenario, sensor] =
SCRIPT_TestingDS2_ReverseRelativeSpeed170KPH()
%TestingDS2_ReverseRelativeSpeed170KPH - Returns sensor detections
%    allData = TestingDS2_ReverseRelativeSpeed170KPH returns sensor detections
in a structure
%    with time for an internally defined scenario and sensor suite.
%
%    [allData, scenario, sensors] = TestingDS2_ReverseRelativeSpeed170KPH
optionally returns
%    the drivingScenario and detection generator objects.

% Generated by MATLAB(R) 9.10 (R2021a) and Automated Driving Toolbox 3.3
(R2021a).
% Generated on: 14-Aug-2022 19:11:10

% Create the drivingScenario object and ego car
[scenario, egoVehicle] = createDrivingScenario;

% Create all the sensors
sensor = createSensor(scenario);

allData = struct('Time', {}, 'ActorPoses', {}, 'ObjectDetections', {},
'LaneDetections', {}, 'PointClouds', {}, 'INSMeasurements', {});
running = true;
while running

    % Generate the target poses of all actors relative to the ego vehicle
    poses = targetPoses(egoVehicle);
    time  = scenario.SimulationTime;

    % Generate detections for the sensor
    laneDetections = [];
    objectDetections = [];
    insMeas = [];
    if sensor.HasRoadsInputPort
        rdmesh = roadMesh(egoVehicle,min(500,sensor.MaxRange));
        [ptClouds, isValidPointCloudTime] = sensor(poses, rdmesh, time);
    else
        [ptClouds, isValidPointCloudTime] = sensor(poses, time);
    end

    % Aggregate all detections into a structure for later use
    if isValidPointCloudTime
        allData(end + 1) = struct( ...
            'Time',       scenario.SimulationTime, ...
            'ActorPoses', actorPoses(scenario), ...
            'ObjectDetections', {objectDetections}, ...
            'LaneDetections', {laneDetections}, ...
            'PointClouds',   {ptClouds}, ... %#ok<AGROW>
            'INSMeasurements',   {insMeas}); %#ok<AGROW>
    end

    % Advance the scenario one time step and exit the loop if the scenario is
complete
```

```
    running = advance(scenario);
end

% Restart the driving scenario to return the actors to their initial positions.
restart(scenario);

% Release the sensor object so it can be used again.
release(sensor);

%%%%%%%%%%%%%%%%%%%%
% Helper functions %
%%%%%%%%%%%%%%%%%%%%

% Units used in createSensors and createDrivingScenario
% Distance/Position - meters
% Speed             - meters/second
% Angles            - degrees
% RCS Pattern       - dBsm

function sensor = createSensor(scenario)
% createSensors Returns all sensor objects to generate detections

% Assign into each sensor the physical and radar profiles for all actors
profiles = actorProfiles(scenario);
sensor = lidarPointCloudGenerator('SensorIndex', 1, ...
    'UpdateInterval', 0.01, ...
    'SensorLocation', [1.9 0], ...
    'MaxRange', 80, ...
    'ElevationLimits', [-15 15], ...
    'ActorProfiles', profiles);

function [scenario, egoVehicle] = createDrivingScenario
% createDrivingScenario Returns the drivingScenario defined in the Designer

% Construct a drivingScenario object.
scenario = drivingScenario;

% Add all road segments
roadCenters = [0 0 0;
    1000 0 0];
laneSpecification = lanespec(2);
road(scenario, roadCenters, 'Lanes', laneSpecification, 'Name', 'Road');

% Add the ego vehicle
egoVehicle = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [90 -1.95853652867245 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'Name', 'Car');
waypoints = [90 -1.95853652867245 0;
    159.4 -2 0;
    404.8 -1.5 0;
    666.1 -1.6 0;
    814.1 -1.6 0;
    919.5 -0.8 0;
    994.4 -1.2 0];
speed = [22.22;22.22;22.22;22.22;22.22;22.22;22.22];
waittime = [0;0;0;0;0;0;0];
trajectory(egoVehicle, waypoints, speed, waittime);

% Add the non-ego actors
car1 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [109.566237360126 -1.82271394615129 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car1');
waypoints = [109.566237360126 -1.82271394615129 0;
```

```
    132 -2.1 0;
    218.49 -2.29 0.01;
    324.5 -1.9 0;
    420.3 -2 0;
    557.3 -2.1 0;
    684.5 -2 0;
    845.6 -1.6 0;
    957.8 -1.6 0;
    994 -2.2 0];
speed = [21.94;21.94;21.94;21.94;21.94;21.94;21.94;21.94;21.94;21.94];
waittime = [0;0;0;0;0;0;0;0;0;0];
trajectory(car1, waypoints, speed, waittime);

car5 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [175.2 1.89879061177704 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car5');
waypoints = [175.2 1.89879061177704 0;
    3.1 2.3 0];
speed = [25;25];
trajectory(car5, waypoints, speed);



% TEST DATA GENERATION: BackOvertaking
%-----------------------------------------

function [allData, scenario, sensor] = SCRIPT_TestingDS4_BackOvertaking()
%TestingDS4_BackOvertaking - Returns sensor detections
%    allData = TestingDS4_BackOvertaking returns sensor detections in a
structure
%    with time for an internally defined scenario and sensor suite.
%
%    [allData, scenario, sensors] = TestingDS4_BackOvertaking optionally
returns
%    the drivingScenario and detection generator objects.

% Generated by MATLAB(R) 9.10 (R2021a) and Automated Driving Toolbox 3.3
(R2021a).
% Generated on: 15-Aug-2022 17:55:03

% Create the drivingScenario object and ego car
[scenario, egoVehicle] = createDrivingScenario;

% Create all the sensors
sensor = createSensor(scenario);

allData = struct('Time', {}, 'ActorPoses', {}, 'ObjectDetections', {},
'LaneDetections', {}, 'PointClouds', {}, 'INSMeasurements', {});
running = true;
while running

    % Generate the target poses of all actors relative to the ego vehicle
    poses = targetPoses(egoVehicle);
    time  = scenario.SimulationTime;

    % Generate detections for the sensor
    laneDetections = [];
    objectDetections = [];
    insMeas = [];
    if sensor.HasRoadsInputPort
        rdmesh = roadMesh(egoVehicle,min(500,sensor.MaxRange));
        [ptClouds, isValidPointCloudTime] = sensor(poses, rdmesh, time);
    else
        [ptClouds, isValidPointCloudTime] = sensor(poses, time);
    end
```

```
    % Aggregate all detections into a structure for later use
    if isValidPointCloudTime
        allData(end + 1) = struct( ...
            'Time',       scenario.SimulationTime, ...
            'ActorPoses', actorPoses(scenario), ...
            'ObjectDetections', {objectDetections}, ...
            'LaneDetections', {laneDetections}, ...
            'PointClouds',   {ptClouds}, ... %#ok<AGROW>
            'INSMeasurements',   {insMeas}); %#ok<AGROW>
    end

    % Advance the scenario one time step and exit the loop if the scenario is
complete
    running = advance(scenario);
end

% Restart the driving scenario to return the actors to their initial positions.
restart(scenario);

% Release the sensor object so it can be used again.
release(sensor);

%%%%%%%%%%%%%%%%%%%%
% Helper functions %
%%%%%%%%%%%%%%%%%%%%

% Units used in createSensors and createDrivingScenario
% Distance/Position - meters
% Speed             - meters/second
% Angles            - degrees
% RCS Pattern       - dBsm

function sensor = createSensor(scenario)
% createSensors Returns all sensor objects to generate detections

% Assign into each sensor the physical and radar profiles for all actors
profiles = actorProfiles(scenario);
sensor = lidarPointCloudGenerator('SensorIndex', 1, ...
    'SensorLocation', [1.9 0], ...
    'MaxRange', 80, ...
    'ElevationLimits', [-15 15], ...
    'ActorProfiles', profiles);

function [scenario, egoVehicle] = createDrivingScenario
% createDrivingScenario Returns the drivingScenario defined in the Designer

% Construct a drivingScenario object.
scenario = drivingScenario('SampleTime', 0.1);

% Add all road segments
roadCenters = [0 0 0;
    1000 0 0];
laneSpecification = lanespec(2);
road(scenario, roadCenters, 'Lanes', laneSpecification, 'Name', 'Road');

% Add the ego vehicle
egoVehicle = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [77.8 -1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'Name', 'Car');
waypoints = [77.8 -1.8 0;
    126.7 -1.7 0;
    169.5 -1.7 0;
    208.1 -1.5 0;
    256.7 -1.8 0;
    300.6 -1.8 0;
```

```
    372.7 -1.6 0;
    429.6 -1.6 0;
    488.8 -1.6 0;
    537.9 -1.7 0;
    633.9 -1.7 0;
    701.5 -2.1 0;
    810.9 -1.6 0;
    873.5 -2.1 0;
    927.5 -1.9 0;
    993 -1.6 0];
speed =
[27.77;27.77;27.77;27.77;27.77;27.77;27.77;27.77;27.77;27.77;27.77;27.77;27.77
;27.77;27.77;27.77];
trajectory(egoVehicle, waypoints, speed);

% Add the non-ego actors
car1 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [99.8 -1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car1');
waypoints = [99.8 -1.8 0;
    137.2 -1.7 0;
    175 -2 0;
    212.6 -1.7 0;
    262 -1.7 0;
    305.9 -2.1 0;
    376.3 -1.8 0;
    433.5 -1.8 0;
    492.4 -2 0;
    540.5 -2 0;
    636.8 -2.3 0;
    703.8 -2 0;
    813.9 -2 0;
    875.4 -2 0;
    931.3 -1.8 0;
    996.9 -1.9 0];
speed =
[27.5;27.5;27.5;27.5;27.5;27.5;27.5;27.5;27.5;27.5;27.5;27.5;27.5;27.5;27
.5];
trajectory(car1, waypoints, speed);

car2 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [50 -1.6 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car2');
waypoints = [50 -1.6 0;
    103.6 -1.5 0;
    114.6 -1.5 0;
    154.4 0.9 0;
    194.9 1.8 0;
    211.2 1.8 0;
    233.7 1.5 0;
    253.5 1.5 0;
    289.5 1.4 0;
    319.1 1.3 0;
    359.1 1.3 0;
    394.9 1.3 0;
    414.4 0.6 0;
    444.1 -1.2 0;
    462.9 -1.5 0;
    480.1 -1.2 0;
    545.5 -2 0;
    724.7 -2 0;
    997.2 -1.4 0];
```

```
speed =
[33.33;33.33;33.33;33.33;33.33;33.33;33.33;33.33;33.33;33.33;33.33;33.33;33.33;33.33
;33.33;33.33;33.33;33.33;33.33;33.33;33.33];
waittime = [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];
trajectory(car2, waypoints, speed, waittime);


% TEST DATA GENERATION: TrafficJamSituation
%-------------------------------------------------


function [allData, scenario, sensor] = SCRIPT_TestingDS5_TrafficJamSituation()
%TestingDS5_TrafficJamSituation - Returns sensor detections
%    allData = TestingDS5_TrafficJamSituation returns sensor detections in a
structure
%    with time for an internally defined scenario and sensor suite.
%
%    [allData, scenario, sensors] = TestingDS5_TrafficJamSituation optionally
returns
%    the drivingScenario and detection generator objects.

% Generated by MATLAB(R) 9.10 (R2021a) and Automated Driving Toolbox 3.3
(R2021a).
% Generated on: 15-Aug-2022 17:23:00

% Create the drivingScenario object and ego car
[scenario, egoVehicle] = createDrivingScenario;

% Create all the sensors
sensor = createSensor(scenario);

allData = struct('Time', {}, 'ActorPoses', {}, 'ObjectDetections', {},
'LaneDetections', {}, 'PointClouds', {}, 'INSMeasurements', {});
running = true;
while running

    % Generate the target poses of all actors relative to the ego vehicle
    poses = targetPoses(egoVehicle);
    time  = scenario.SimulationTime;

    % Generate detections for the sensor
    laneDetections = [];
    objectDetections = [];
    insMeas = [];
    if sensor.HasRoadsInputPort
        rdmesh = roadMesh(egoVehicle,min(500,sensor.MaxRange));
        [ptClouds, isValidPointCloudTime] = sensor(poses, rdmesh, time);
    else
        [ptClouds, isValidPointCloudTime] = sensor(poses, time);
    end

    % Aggregate all detections into a structure for later use
    if isValidPointCloudTime
        allData(end + 1) = struct( ...
            'Time',       scenario.SimulationTime, ...
            'ActorPoses', actorPoses(scenario), ...
            'ObjectDetections', {objectDetections}, ...
            'LaneDetections', {laneDetections}, ...
            'PointClouds',   {ptClouds}, ... %#ok<AGROW>
            'INSMeasurements',   {insMeas}); %#ok<AGROW>
    end

    % Advance the scenario one time step and exit the loop if the scenario is
complete
    running = advance(scenario);
end

% Restart the driving scenario to return the actors to their initial positions.
```

```
restart(scenario);

% Release the sensor object so it can be used again.
release(sensor);

%%%%%%%%%%%%%%%%%%%%
% Helper functions %
%%%%%%%%%%%%%%%%%%%%

% Units used in createSensors and createDrivingScenario
% Distance/Position - meters
% Speed             - meters/second
% Angles            - degrees
% RCS Pattern       - dBsm

function sensor = createSensor(scenario)
% createSensors Returns all sensor objects to generate detections

% Assign into each sensor the physical and radar profiles for all actors
profiles = actorProfiles(scenario);
sensor = lidarPointCloudGenerator('SensorIndex', 1, ...
    'SensorLocation', [1.9 0], ...
    'MaxRange', 80, ...
    'ElevationLimits', [-15 15], ...
    'ActorProfiles', profiles);

function [scenario, egoVehicle] = createDrivingScenario
% createDrivingScenario Returns the drivingScenario defined in the Designer

% Construct a drivingScenario object.
scenario = drivingScenario('SampleTime', 0.1);

% Add all road segments
roadCenters = [0 0 0;
    200 0 0];
laneSpecification = lanespec(2);
road(scenario, roadCenters, 'Lanes', laneSpecification, 'Name', 'Road');

% Add the ego vehicle
egoVehicle = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [96.9 -2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'Name', 'Car');
waypoints = [96.9 -2 0;
    127 -1.9 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(egoVehicle, waypoints, speed, waittime);

% Add the non-ego actors
car1 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [109.4 -2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car1');
waypoints = [109.4 -2 0;
    140 -1.9 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car1, waypoints, speed, waittime);

car2 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [90.5 -2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
```

```
    'Name', 'Car2');
waypoints = [90.5 -2 0;
    120 -2.1 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car2, waypoints, speed, waittime);

car3 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [115.233333333333 -2.1 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car3');
waypoints = [115.233333333333 -2.1 0;
    145 -2 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car3, waypoints, speed, waittime);

car4 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [121.466666666667 -2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car4');
waypoints = [121.466666666667 -2 0;
    151 -2 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car4, waypoints, speed, waittime);

car5 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [127.7 -2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car5');
waypoints = [127.7 -2 0;
    157.7 -1.9 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car5, waypoints, speed, waittime);

car6 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [84.1333333333333 -2.1 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car6');
waypoints = [84.1333333333333 -2.1 0;
    115 -2.3 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car6, waypoints, speed, waittime);

car7 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [77.3666666666667 -2.1 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car7');
waypoints = [77.3666666666667 -2.1 0;
    107 -2.1 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car7, waypoints, speed, waittime);

car8 = vehicle(scenario, ...
```

```
    'ClassID', 1, ...
    'Position', [70.9 -2.2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car8');
waypoints = [70.9 -2.2 0;
    101 -2.1 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car8, waypoints, speed, waittime);

car9 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [64.2333333333333 -2.1 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car9');
waypoints = [64.2333333333333 -2.1 0;
    94 -2.1 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car9, waypoints, speed, waittime);

car10 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [57.4666666666667 -2.1 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car10');
waypoints = [57.4666666666667 -2.1 0;
    87 -2.1 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car10, waypoints, speed, waittime);

car11 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [133.7 -2.1 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car11');
waypoints = [133.7 -2.1 0;
    164 -2 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car11, waypoints, speed, waittime);

car12 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [139.933333333333 -2.1 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car12');
waypoints = [139.933333333333 -2.1 0;
    170 -1.9 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car12, waypoints, speed, waittime);

car13 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [124.3 2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car13');
waypoints = [124.3 2 0;
    154 1.9 0];
speed = [3.47;3.47];
```

```
waittime = [0;0];
trajectory(car13, waypoints, speed, waittime);

car14 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [51.4 -2.1 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car14');
waypoints = [51.4 -2.1 0;
    81.4 -2 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car14, waypoints, speed, waittime);

car15 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [103.233333333333 -2.1 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car15');
waypoints = [103.233333333333 -2.1 0;
    133 -2 0];
speed = [0.8;0.8];
waittime = [0;0];
trajectory(car15, waypoints, speed, waittime);

car16 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [146 1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car16');
waypoints = [146 1.8 0;
    176 1.7 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car16, waypoints, speed, waittime);

car17 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [138.933333333333 1.9 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car17');
waypoints = [138.933333333333 1.9 0;
    169 1.9 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car17, waypoints, speed, waittime);

car18 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [131.966666666667 1.9 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car18');
waypoints = [131.966666666667 1.9 0;
    162 1.5 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car18, waypoints, speed, waittime);

car19 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [124.3 2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
```

```
    'Name', 'Car19');
waypoints = [103.1 1.8 0;
    126 1.3 0];
speed = [3.47;3.47];
trajectory(car19, waypoints, speed);

car20 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [117.133333333333 1.7 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car20');
waypoints = [117.133333333333 1.7 0;
    147 2.1 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car20, waypoints, speed, waittime);

car21 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [110.166666666667 1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car21');
waypoints = [110.166666666667 1.8 0;
    140 1.9 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car21, waypoints, speed, waittime);

car22 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [103.1 1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car22');
waypoints = [103.1 1.8 0;
    133 1.9 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car22, waypoints, speed, waittime);

car16 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [96.0333333333333 1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car16');
waypoints = [96.0333333333333 1.8 0;
    126 1.3 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car16, waypoints, speed, waittime);

car17 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [88.9666666666667 1.9 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car17');
waypoints = [88.9666666666667 1.9 0;
    119 1.9 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car17, waypoints, speed, waittime);

car18 = vehicle(scenario, ...
    'ClassID', 1, ...
```

```
    'Position', [81 1.9 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car18');
waypoints = [81 1.9 0;
    111 1.9 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car18, waypoints, speed, waittime);

car19 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [73.3333333333333 2 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car19');
waypoints = [73.3333333333333 2 0;
    103 1.7 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car19, waypoints, speed, waittime);

car20 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [66.1666666666666 1.7 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car20');
waypoints = [66.1666666666666 1.7 0;
    96 2.1 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car20, waypoints, speed, waittime);

car21 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [59.2 1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car21');
waypoints = [59.2 1.8 0;
    89 1.5 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car21, waypoints, speed, waittime);

car23 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [52.1333333333333 1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car23');
waypoints = [52.1333333333333 1.8 0;
    82 1.9 0];
speed = [3.47;3.47];
waittime = [0;0];
trajectory(car23, waypoints, speed, waittime);

car24 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [146.386071690422 -2.15636626102658 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car24');
waypoints = [146.386071690422 -2.15636626102658 0;
    176.6 -2.1 0];
speed = [0.8;0.8];
waittime = [0;0];
```

```
trajectory(car24, waypoints, speed, waittime);


% TEST DATA GENERATION: ariableIntensityAtImax4000Veh_hour
%-------------------------------------------------------------


function [allData, scenario, sensor] =
SCRIPT_TestingDS6_VariableIntensityAtImax4000Veh_hour()
%TestingDS6_VariableIntensityAtImax4000Veh_hour - Returns sensor detections
%    allData = TestingDS6_VariableIntensityAtImax4000Veh_hour returns sensor
detections in a structure
%    with time for an internally defined scenario and sensor suite.
%
%    [allData, scenario, sensors] =
TestingDS6_VariableIntensityAtImax4000Veh_hour optionally returns
%    the drivingScenario and detection generator objects.

% Generated by MATLAB(R) 9.10 (R2021a) and Automated Driving Toolbox 3.3
(R2021a).
% Generated on: 14-Aug-2022 20:24:49

% Create the drivingScenario object and ego car
[scenario, egoVehicle] = createDrivingScenario;

% Create all the sensors
sensor = createSensor(scenario);

allData = struct('Time', {}, 'ActorPoses', {}, 'ObjectDetections', {},
'LaneDetections', {}, 'PointClouds', {}, 'INSMeasurements', {});
running = true;
while running

    % Generate the target poses of all actors relative to the ego vehicle
    poses = targetPoses(egoVehicle);
    time  = scenario.SimulationTime;

    % Generate detections for the sensor
    laneDetections = [];
    objectDetections = [];
    insMeas = [];
    if sensor.HasRoadsInputPort
        rdmesh = roadMesh(egoVehicle,min(500,sensor.MaxRange));
        [ptClouds, isValidPointCloudTime] = sensor(poses, rdmesh, time);
    else
        [ptClouds, isValidPointCloudTime] = sensor(poses, time);
    end

    % Aggregate all detections into a structure for later use
    if isValidPointCloudTime
        allData(end + 1) = struct( ...
            'Time',       scenario.SimulationTime, ...
            'ActorPoses', actorPoses(scenario), ...
            'ObjectDetections', {objectDetections}, ...
            'LaneDetections', {laneDetections}, ...
            'PointClouds',   {ptClouds}, ... %#ok<AGROW>
            'INSMeasurements',   {insMeas}); %#ok<AGROW>
    end

    % Advance the scenario one time step and exit the loop if the scenario is
complete
    running = advance(scenario);
end

% Restart the driving scenario to return the actors to their initial positions.
restart(scenario);

% Release the sensor object so it can be used again.
```

```
release(sensor);

%%%%%%%%%%%%%%%%%%%%
% Helper functions %
%%%%%%%%%%%%%%%%%%%%

% Units used in createSensors and createDrivingScenario
% Distance/Position - meters
% Speed             - meters/second
% Angles            - degrees
% RCS Pattern       - dBsm

function sensor = createSensor(scenario)
% createSensors Returns all sensor objects to generate detections

% Assign into each sensor the physical and radar profiles for all actors
profiles = actorProfiles(scenario);
sensor = lidarPointCloudGenerator('SensorIndex', 1, ...
    'SensorLocation', [1.9 0], ...
    'MaxRange', 80, ...
    'ElevationLimits', [-15 15], ...
    'ActorProfiles', profiles);

function [scenario, egoVehicle] = createDrivingScenario
% createDrivingScenario Returns the drivingScenario defined in the Designer

% Construct a drivingScenario object.
scenario = drivingScenario;

% Add all road segments
roadCenters = [0 0.18 0;
    200 0.18 0];
laneSpecification = lanespec(2);
road(scenario, roadCenters, 'Lanes', laneSpecification, 'Name', 'Road');

% Add the ego vehicle
egoVehicle = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [100 -1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'Name', 'Car');
waypoints = [100 -1.8 0;
    100.72 -1.8 0.01];
speed = [0.1;0.1];
waittime = [0;0];
trajectory(egoVehicle, waypoints, speed, waittime);

% Add the non-ego actors
car1 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [50 2.00252094940925 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car1');
waypoints = [50 2.00252094940925 0;
    150 1.8 0];
speed = [13.89;13.89];
waittime = [0;0];
trajectory(car1, waypoints, speed, waittime);

car4 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [62.5 2.31198495416241 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car4');
waypoints = [62.5 2.31198495416241 0;
    162.5 1.8 0];
```

```
speed = [13.89;13.89];
waittime = [0;0];
trajectory(car4, waypoints, speed, waittime);

car6 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [25 2.00049045543055 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car6');
waypoints = [25 2.00049045543055 0;
    125 1.8 0];
speed = [13.89;13.89];
waittime = [0;0];
trajectory(car6, waypoints, speed, waittime);

car8 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [12.5 1.90036090940659 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car8');
waypoints = [12.5 1.90036090940659 0;
    112.5 1.8 0];
speed = [13.89;13.89];
waittime = [0;0];
trajectory(car8, waypoints, speed, waittime);

car10 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [0 1.81154044679363 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car10');
waypoints = [0 1.81154044679363 0;
    100 1.8 0];
speed = [13.89;13.89];
waittime = [0;0];
trajectory(car10, waypoints, speed, waittime);

car11 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [75 1.89347045161036 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car11');
waypoints = [75 1.89347045161036 0;
    164 1.8 0];
speed = [13.89;13.89];
waittime = [0;0];
trajectory(car11, waypoints, speed, waittime);

car22 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [109.4 -1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car22');
waypoints = [109.4 -1.8 0;
    110.12 -1.8 0.01];
speed = [0.1;0.1];
waittime = [0;0];
trajectory(car22, waypoints, speed, waittime);

car16 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [90.6 -1.8 0], ...
    'Mesh', driving.scenario.carMesh, ...
```

```
    'PlotColor', [255 105 41] / 255, ...
    'Name', 'Car16');
waypoints = [90.6 -1.8 0;
    91.32 -1.8 0.01];
speed = [0.1;0.1];
waittime = [0;0];
trajectory(car16, waypoints, speed, waittime);

car24 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [87.5 1.70502766036107 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car24');
waypoints = [87.5 1.70502766036107 0;
    187.5 1.8 0];
speed = [13.89;13.89];
waittime = [0;0];
trajectory(car24, waypoints, speed, waittime);

car19 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [37.5 2.0004280241103 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car19');
waypoints = [37.5 2.0004280241103 0;
    137.5 1.8 0];
speed = [13.89;13.89];
waittime = [0;0];
trajectory(car19, waypoints, speed, waittime);

car2 = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [100 1.90502766036107 0], ...
    'Mesh', driving.scenario.carMesh, ...
    'PlotColor', [100 212 19] / 255, ...
    'Name', 'Car2');
waypoints = [100 1.90502766036107 0;
    200 2 0];
speed = [13.89;13.89];
waittime = [0;0];
trajectory(car2, waypoints, speed, waittime);
```