

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO



Diseño e implementación de un sistema de detección y rastreo de  
vehículos para el modelo a escala AutoNOMOS, y simulación de  
una red VANET

**TESIS**

PARA OBTENER EL TÍTULO DE  
INGENIERO EN MECATRÓNICA

P R E S E N T A

**ISRAEL FONSECA ZÁRATE**

**A S E S O R:** MTRO. RAFAEL GREGORIO GAMBOA HIRALES

CIUDAD DE MÉXICO

2024

“Con fundamento en los artículos 21 y 27 de la Ley Federal del Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada **“Diseño e implementación de un sistema de detección y rastreo de vehículos para el modelo a escala AutoNOMOS, y simulación de una red VANET”**, otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la Biblioteca Raúl Baillères Jr., autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una contraprestación.”

**Israel Fonseca Zárate**

---

Fecha

---

Firma

## **Resumen**

Dentro de la investigación realizada sobre vehículos autónomos, un punto importante es la detección y el seguimiento de otros participantes en el tráfico que puedan ser riesgos potenciales, tanto para el vehículo propio como para otros. Razón por la cual se habla de lograr un entorno colaborativo entre vehículos de esta naturaleza. Como objetivo de este trabajo se presenta una implementación sobre el AutoModelCar de un sistema de detección y rastreo de vehículos. Esto desde un enfoque que requiera poco poder de procesamiento y logre resultados cercanos en tiempo real para el modelo a escala con que se cuenta. Para ello se eligió la combinación de filtros de cascada con máquinas de soporte vectorial para la detección, y el uso del filtro de Kalman para el rastreo. Después, pensando en un entorno colaborativo, se busca que los vehículos compartan entre sí sus detecciones mediante una red VANET basada en el modelo “vehicle to vehicle” (V2V). Para ello se simula y experimenta con el protocolo “Ad hoc On-Demand Distance Vector Routing” (AODV) y el estándar “Dedicated Short-Range Communications” (DSRC). Si bien en vehículos autónomos el método indiscutible son las redes neuronales, con los métodos usados se logró alcanzar un buen desempeño que, para otras aplicaciones, puede ser una opción relevante cuando se buscan lograr resultados en tiempo real y con bajo poder de procesamiento. Mientras que en redes VANET, se experimentó sobre los estándares que se están probando hoy en día viéndolos desde un enfoque orientado a aplicaciones de seguridad. Con esto en mente se buscó tener bajos intervalos de transmisión entre el envío de mensajes (se esperaría fueran capaces de enviar cada 0.2 a 0.5 segundos) y un bajo “Round-Trip Time” (RTT). Como se

demuestra, el modelo V2V puede funcionar si toma en cuenta las condiciones de tráfico, para adaptar parámetros como la tasa de envíos. Aquí, los principales retos son la velocidad de los nodos, y situaciones con una alta densidad de nodos, esto último, se puede mejorar si se añade el uso de infraestructura en lugares como ciudades, pasando a un modelo V2X. Donde se espera que la infraestructura cumpla solo un rol de coordinador, y que aproveche las ventajas de una comunicación directa entre vehículos, como son la latencia y cobertura geográfica.

El trabajo completo y todos los programas se pueden encontrar en:  
<https://github.com/ifonsecaz/Autonomous-Vanets>

**Palabras clave:** Maquinas de soporte vectorial, Filtros de cascada, Filtro de Kalman, Protocolo AODV, Estándar IEEE 802.11p (DSRC), Detección, Rastreo, Vehículos autónomos, Redes vehiculares.

## **Agradecimientos**

Quiero expresar mi agradecimiento a todas las personas que me han impulsado y acompañado durante toda mi carrera, que me permitieron concluir esta etapa e hicieron posible este trabajo.

Primero que nada, agradezco a mi familia. A mi mamá María del Carmen y mi papá Cesar Fonseca quienes me han dado todo su apoyo, y han hecho un gran esfuerzo para ayudarme con mis objetivos profesionales y académicos. Por brindarme toda su confianza y haberme forjado, gracias por haberme dado el soporte y cariño necesario. A mi hermana Isabel, gracias por acompañarme por este camino y todas las porras que me has dado, ahora es también tu turno experimentar esta parte y cerrar una etapa.

Igualmente, gracias por acompañarme en esta travesía a mis abuelos y tíos que me brindaron todo su apoyo y son parte de este logro también.

A mi asesor Rafael Gamboa, le agradezco toda su dedicación y paciencia, gracias a su guía y orientación fue posible este trabajo.

Igualmente, a todos los profesores que me acompañaron durante la carrera, que me guiaron y brindaron sus conocimientos. Especialmente al profesor Ante Salcedo, gracias por su motivación para elegir este tema, y al profesor Marcelo Mejía que me permitió darle los toques finales al trabajo.

## TABLA DE CONTENIDO

<i>Tabla de contenido.....</i>	<b>6</b>
<i>Índice de figuras.....</i>	<b>10</b>
<i>Índice de tablas.....</i>	<b>19</b>
<b>1     Introducción .....</b>	<b>21</b>
1.1    Contexto.....	21
1.2    Identificación del problema.....	22
1.3    Objetivos.....	24
1.4    Metodología.....	25
1.5    Organización del documento.....	26
<b>2     Análisis.....</b>	<b>29</b>
2.1    Requerimientos .....	29
2.1.1    Requerimientos funcionales .....	29
2.1.2    Requerimientos no funcionales .....	32
2.2    Restricciones .....	33
2.3 Alcance .....	34
2.4 Trabajos relacionados.....	35

<b>3</b>	<b><i>Marco teórico</i></b>	<b>39</b>
<b>3.1</b>	<b>Detección de objetos</b>	<b>39</b>
3.1.1	Clasificador de cascada .....	41
3.1.2	HOG+SVM .....	44
3.1.3	Redes neuronales convolucionales .....	48
<b>3.2</b>	<b>Rastreo de objetos</b>	<b>49</b>
3.2.1	Filtro de Kalman .....	50
3.2.2	Caso multivariable.....	53
<b>3.3</b>	<b>Comunicación entre vehículos</b>	<b>54</b>
3.3.1	Patrones de comunicación.....	56
3.3.2	Modelos V2X, V2I y V2V .....	59
3.3.3	DSRC.....	61
<b>4</b>	<b><i>Diseño</i></b>	<b>67</b>
<b>4.1</b>	<b>Arquitectura</b>	<b>67</b>
<b>4.2</b>	<b>Detección de objetos</b>	<b>76</b>
4.2.1	Técnicas de detección .....	78
4.2.2	Comparación con redes neuronales convolucionales.....	81
<b>4.3</b>	<b>Rastreo de objetos</b>	<b>82</b>
<b>5</b>	<b><i>Codificación</i></b>	<b>86</b>
<b>5.1</b>	<b>Entrenamiento del filtro de cascada</b>	<b>86</b>

5.2 Entrenamiento de SVM .....	89
5.3 Detección y nodo para el vehículo.....	98
5.4 Rastreo .....	107
<b>6     <i>Resultados en la detección y rastreo</i></b> .....	<b>117</b>
<b>7     <i>Diseño de la Red VANET</i>.....</b>	<b>135</b>
7.1     Arquitectura .....	135
7.2     Simulación 1 .....	139
7.3     Simulación 2 .....	142
7.4 Contenido de los mensajes .....	145
<b>8     <i>Implementación de una red VANET</i> .....</b>	<b>147</b>
8.1     Arquitectura y requisitos de la red VANET .....	147
8.2     Implementación.....	148
8.2.1     AODV.....	148
8.2.2     IEEE 802.11p .....	154
<b>9     <i>Resultados de la red VANET</i> .....</b>	<b>164</b>
9.1     Resultados simulación 1 .....	164
9.1.1     Diferentes velocidades e intervalos.....	167
9.2     Resultados simulación 2 .....	172
9.2.1     Diferentes densidades de nodos.....	176

<b>10      Conclusiones .....</b>	<b>194</b>
<b>Referencias .....</b>	<b>198</b>
<b>Apéndice 1: Documentación .....</b>	<b>205</b>
Documentación AutoModelCar .....	205
Nodos Programados.....	234
<b>Apéndice 2: Tutorial AutoModelCar.....</b>	<b>238</b>
Guía de ROS.....	238
Transformaciones .....	253
Uso de la cámara .....	258

## ÍNDICE DE FIGURAS

Figura 1-1 etapas en la programación extrema. Fuente: (Pressman, 2010) ..... 26

Figura 3-1 Ejemplo de la operación realizada en lbp, primero si el valor del pixel es menor al centro se toma 1, a continuación, partiendo de la esquina superior derecha se le asigna como  $2^0$ , de ahí se da una vuelta en sentido de las manecillas del reloj, aumenta. Fuente: (Rosebrock 2015).....43

Figura 3-2 Ejemplo de la separación de dos variables por un margen de separación, se muestra la separación óptima que logra el mayor margen entre las dos variables, fuente: (“opencv: introduction to support vector machines” s. F.).....46

Figura 3-3 Tabla de diferentes kernels para linealizar un problema. Fuente:(Kecman 2005).....47

Figura 3-4 Pasos del filtro de Kalman, hay una inicialización a partir de la que se realiza la predicción, en el siguiente paso, sí se tienen las medidas se realiza la actualización, además, se puede agregar un paso de control. Fuente: (Becker s. F.).....51

Figura 3-5 Fuente: (Hartenstein y Laberteaux 2010) capas del estándar DSRC, se compone de dos partes principales, el estándar IEEE 802.11p e IEEE 1609 o wave.....62

Figura 3-6 Características de la banda 5850-5925 MHz. Fuente: (“identificación de necesidades de espectro para sistemas de transporte inteligente en la banda 5850-5925 MHz”, 2021).....	62
Figura 3-7 Fuente: (Kenney 2011) encabezados de capa 2.....	63
Figura 3-8 Fuente: (Kenney 2011) encabezados de la subcapa LLC y SNAP.....	65
Figura 3-9 Fuente: (Kenney 2011) formato de wave short message.....	66
Figura 4-1 En el recuadro es el nombre del nodo, en los círculos los tópicos que publican. Son los tópicos los que interconectan con otros nodos. El nodo manual_control, a partir de la velocidad y steering se comunica con los motores. Lane_navigation en realidad son un conjunto de nodos que se encargan del seguimiento de línea.....	69
Figura 4-2 Pasos para la detección .....	73
Figura 4-3 Pasos para el rastreo.....	74
Figura 4-4 Pasos para el entrenamiento de máquinas de soporte vectorial .....	75
Figura 5-1 Ejemplos de vehículos .....	86
Figura 5-2 Ejemplos de imágenes negativas para el entrenamiento .....	87
Figura 5-3 Imágenes de la parte trasera de vehículos .....	90

Figura 5-4 Imágenes de vehículos de lado.....	90
Figura 5-5 Función load_images .....	90
Figura 5-6 Con blocksize (64,64), cellsize y blockstride (8,8), con solo 8x8 celdas no es suficiente para extraer las características de la imagen, por lo que el resultado es muy pobre.	92
Figura 5-7 Con blocksize (64,64), cellsize y blockstride de (4,4). ....	92
Figura 5-8 Con filtro gaussiano (5,5).....	92
Figura 5-9 Imagen del vehículo del simulador .....	92
Figura 5-10 HOG features de un árbol .....	92
Figura 5-11 HOG features de la carretera.....	92
Figura 5-12 Función computehogs, extrae los descriptores de todas las imágenes.....	93
Figura 5-13 Código de la función convert_to_ml, mismo propuesto en los ejemplos de opencv .....	94
Figura 5-14 Entrenamiento de SVM con trainauto.....	95

Figura 5-15 Se observa el impacto de establecer pesos para el valor de c, a la izquierda la clasificación normal, a la derecha con un peso de 0.9 sobre la clase de verde, fuente: (“svm bias on weights of positives and negatives - opencv q&a forum” s. F.).....	96
Figura 5-16 Imagen de la parte de entrenamiento en opencv 2.4 usando train .....	98
Figura 5-17 Imagen de la función main, la configuración inicial, hasta el inicio del ciclo .....	101
Figura 5-18 Ejemplos de falsos negativos .....	104
Figura 5-19 Fragmento del código para detectar sobre una región .....	107
Figura 5-20 measurementnoisecov de 0.1 .....	111
Figura 5-21 mesasurementnoisecov de 0.01 .....	111
Figura 5-22 measurementnoisecov de 0.001 .....	112
Figura 5-23 Con processnoisecov de 0.00005, los puntos azules reaccionan más lento a los cambios .....	112
Figura 5-24 Con processnoisecov de 0.5, los puntos azules reaccionan más rápido y como consecuencia se disparan .....	112
Figura 5-25 Código completo para crear el filtro de Kalman .....	113

Figura 5-26 Código para hacer la predicción y corrección.....	116
Figura 6-1 Curvas ROC y PRC de la primera SVM, el punto rojo es el umbral elegido ...	119
Figura 6-2 Curvas ROC y PRC de la segunda SVM .....	121
Figura 6-3 Curvas ROC y PRC del filtro de cascada .....	124
Figura 6-4 Comparación de las curvas ROC de todos los detectores .....	126
Figura 6-5 Comparación de las curvas PRC de todos los detectores .....	126
Figura 6-6 Candidatos arrojados por el filtro de cascada .....	127
Figura 6-7 Ejemplos de detecciones sobre cuadros, en la primera imagen se identifican correctamente los dos vehículos, en el segundo por el cambio de iluminación no detecta uno de los vehículos, en la tercera imagen se muestra un falso positivo.....	128
Figura 6-8 Predicción del filtro de Kalman, los recuadros blancos son las detecciones, y los negros la predicción (en esa misma ventana de la predicción es donde se realiza la detección parcial para buscar el vehículo y hacer la corrección. ....	128
Figura 6-9 Imagen del simulador desde gazebo, mapa con dos vehículos .....	130
Figura 6-10 Ejemplo de detecciones en el simulador, arriba se muestra la detección, abajo la predicción.....	131

Figura 6-11 Detecciones en el carro físico .....	133
Figura 6-12 Ejemplo de las detecciones .....	134
Figura 7-1 Corresponde a la incorporación de vehículos de la avenida Ceylán, sobre la av. Gustavo Baz.....	136
Figura 7-2 simulación 1, se experimenta el desempeño con diferente densidad de nodos... <td>139</td>	139
Figura 7-3 A la izquierda la posición inicial, a la derecha la posición final.....	142
Figura 7-4 A la izquierda se observa la posición de partida de los nodos, a la derecha la posición final.....	143
Figura 8-1 Ventana de diseño para un archivo .ned .....	149
Figura 8-2 Código del archivo .ned, son los módulos incluidos.....	149
Figura 8-3 Imagen de la simulación de AODV .....	154
Figura 8-4 Imagen del editor de una red en SUMO .....	154
Figura 8-5 Pestañas para crear construcciones (ala izquierda) y caminos (a la derecha)..	156
Figura 8-6 Ventana para crear rutas.....	157
Figura 8-7 Imagen final en SUMO .....	157
Figura 8-8 Archivo .ned para la segunda simulación .....	159

Figura 8-9 Imagen de la simulación corriendo en OMNeT++, se observa un nodo retransmitiendo a todos un mensaje.....	163
Figura 9-1 resultados del escenario 1 .....	165
Figura 9-2 a la izquierda los tiempos que demora "in progress frames" a la derecha "pending queue" .....	166
Figura 9-3 Rutas formadas por el protocolo AODV.....	166
Figura 9-4 Probabilidad de recibir 2 mensajes con AODV .....	169
Figura 9-5 Probabilidad de recibir 4 mensajes con AODv .....	170
Figura 9-6 El nodo 6 en rojo quiere transmitir .....	172
Figura 9-7 El nodo 6 transmite el mensaje, y los demás nodos al recibirla retransmiten una vez, el nodo 7 y 4 no son vistos por el nodo 6, con las retransmisiones quedan informados. ....	172
Figura 9-8 Los nodos cercanos 5, 11 y 7 bajan su velocidad y permiten que el nodo 6 se incorpore, el nodo 4 no baja su velocidad .....	173
Figura 9-9 Posición final, el nodo 6 se incorpora .....	173
Figura 9-10 Paquetes enviados (incluye retransmisiones) y paquetes recibidos por cada nodo .....	174

Figura 9-11 Tiempos de espera promedio de cada nodo .....	174
Figura 9-12 Tiempos de encolamiento .....	178
Figura 9-13 Probabilidad de recibir 2 y 4 mensajes con 10 nodos .....	179
Figura 9-14 Probabilidad de recibir 2 y 4 mensajes con 10 nodos, en un trayecto de 200 metros.....	180
Figura 9-15 Tiempos de encolamiento con 20 nodos .....	183
Figura 9-16 Probabilidad de recibir 2 y 4 mensajes con 20 nodos .....	183
Figura 9-17 Probabilidad de recibir 2 y 4 mensajes con 20 nodos, en un trayecto de 200 metros.....	184
Figura 9-18 Tiempos de encolamiento con 30 nodos .....	186
Figura 9-19 Probabilidad de recibir 2 y 4 mensajes con 30 nodos .....	187
Figura 9-20 Probabilidad de recibir 2 y 4 mensajes con 30 nodos, en un trayecto de 200 metros.....	187
Figura 9-21 Tiempos de encolamiento al aumentar el número de nodos .....	188
Figura 9-22 Porcentaje de mensajes recibidos a 50 Km/h .....	189
Figura 9-23 Intervalos de transmisión recomendados a una velocidad de 30 km/h .....	190

Figura 9-24 Intervalos de transmisión recomendados a 50 km/h .....	191
Figura 9-25 Intervalos de transmisión recomendados a 80 km/h .....	192
Figura A1-1 A la izquierda módulos y conexiones del modelo 1, a la derecha el modelo 2. Fuente: (“hardware (autonomos model v2) · automodelcar/automodelcarwiki wiki · github” s. f.) .....	205
Figura A2-1 Ejemplo de roslaunch correspondiente a manual_control .....	240
Figura A2-2 Se muestran algunos tópicos relacionados a la cámara del carro, al correr manual_control .....	242
Figura A2-3 Mensaje publicado en el tópico /scan relacionado al lidar.....	243
Figura A2-4 El tópico /app/camera/rgb/image_raw corresponde a la imagen a color publicada por el nodo de la cámara, se lee para la detección .....	243
Figura A2-5 Composición del mensaje tipo laserscan para el lidar .....	245
Figura A2-6 Conexión con transformaciones de las coordenadas de la cámara con el vehículo. Fuente (“access the tf transformation tree in ros - matlab & simulink - mathworks américa latina” s. f.) .....	253
Figura A2-7 Resultado de la conexión de los frames. Fuente (“setting up transformations — navigation 2 1.0.0 documentation” s. f.).....	257

## ÍNDICE DE TABLAS

Tabla 3-1 Comparación entre MANET, VANET y FANET .....	54
Tabla 3-2 Fuente: (Hartenstein y Laberteaux, 2010) algunas aplicaciones y su modelo sugerido .....	60
Tabla 6-1 Resultados de mover el umbral en la primera SVM .....	119
Tabla 6-2 Resultados de la segunda SVM .....	120
Tabla 6-3 Tasas de desempeño de SVM completa con la parte trasera de vehículos .....	121
Tabla 6-4 Tasas de desempeño de SVM completa con vehículos de lado .....	122
Tabla 6-5 Tasas de desempeño resultante de juntar ambas SVM completas .....	122
Tabla 6-6 Resultados del filtro de cascada .....	124
Tabla 6-7 Tasas de desempeño del filtro de cascada.....	124
Tabla 6-8 Tasas de desempeño de ambos detectores .....	125
Tabla 6-9 Tiempos medidos de los detectores en los ambientes de pruebas .....	132
Tabla 9-1 RTT y porcentaje de pérdidas de mensajes, al variar la velocidad e intervalo ..	167
Tabla 9-2 Mensajes recibidos en promedio de cada nodo, retransmisiones en promedio son el número de mensajes únicos que reciben .....	175

Tabla 9-3 Probabilidad de recibir cierto número de mensajes y los tiempos de espera promedio, al variar el tamaño del mensaje, velocidad e intervalo. en una red con 10 nodos

.....177

Tabla 9-4 Comparación de los tiempos de encolamiento y probabilidad de recibir cierto número de mensajes, en una red con 20 vehículos .....

181

Tabla 9-5 Comparación de los tiempos de encolamiento y probabilidad de recibir cierto número de mensajes, en una red con 30 vehículos .....

185

# **1 INTRODUCCIÓN**

En el presente capítulo se aborda la temática del proyecto que se desarrollará como tesis, proporcionando un contexto sobre los vehículos autónomos que nos lleva a la identificación del problema. Asimismo, se exponen los objetivos del trabajo, se describe la metodología seguida y se explica la estructura del documento.

## **1.1 Contexto**

Los vehículos autónomos han experimentado un notable avance, impulsado por los progresos tecnológicos. Entre estos avances se encuentran la diversidad de sensores disponibles, el aumento en la capacidad computacional que ha posibilitado el uso de la inteligencia artificial, así como el desarrollo de comunicaciones de alta velocidad y baja latencia, como es el caso del 5G. Estos avances han propiciado la integración de soluciones que reducen la intervención humana en la conducción de vehículos, con el objetivo de disminuir accidentes, mejorar la movilidad, aumentar la comodidad y agilizar el transporte.

Este desarrollo en vehículos autónomos se ha visualizado desde mucho tiempo atrás. En 1939 General Motors presentaba un primer concepto de vehículo autónomo. Se trataba de un coche que se conducía a sí mismo mediante campos electromagnéticos controlados por radio y operado por púas de metal incrustadas en el camino, el cual se hizo realidad en 1958. Con la carrera espacial se dio otro gran avance, ya que los científicos buscaban enviar vehículos a la Luna que incorporaran cámaras para procesar imágenes del terreno. Desde entonces,

muchos programas de investigación gubernamentales o académicos han surgido. Entre los más importantes se encuentra la competencia de vehículos autónomos, “DARPA Grand Challenge” en 2004, celebrada en el desierto de Mojave en Estados Unidos, la cual incorporaba temas como aprendizaje de máquina. Los avances han continuado y hoy en día hemos llegado a puntos donde compañías han sacado a la venta opciones comerciales como Volvo y Google con WAYMO desde 2017, o TESLA, Audi y BMW (Faisal et al. 2019).

Dentro de los vehículos autónomos se tienen en cuenta diferentes niveles de autonomía, los cuales varían según las tareas que realiza el auto y la intervención que se requiere de las personas. En general, las principales funciones que se ven involucradas son la localización, percepción, planificación, control y gestión. Se incluye también una parte importante de adquisición de información (Coppola y Morisio, 2016). Esta es resultado tanto del uso de sensores, de la comunicación y negociación con otros vehículos o infraestructura.

## **1.2 Identificación del problema**

El ITAM cuenta con un proyecto sobre vehículos autónomos, que inicio gracias a la donación de un modelo a escala desarrollado por el profesor Raúl Rojas en Freie Universität Berlin (“Home · AutoModelCar/AutoModelCarWiki Wiki” s. f.). Además, se tiene el simulador realizado por el profesor Marco Morales junto con el equipo de Eagle Knights del ITAM.

El modelo de coche autónomo, llamado AutoModelcar, contiene sensores y actuadores, como cámaras y un sensor óptico que dispara rayos de luz para medir distancias, llamado LiDAR

(del inglés “Light detection and ranging”). Cada uno de estos, es controlable de forma independiente, pero carece de nodos que los integren en su totalidad para dotarle de un funcionamiento de forma autónoma. De estos sensores, nos enfocaremos en la cámara a color, lo que significa que se requerirá realizar procesamiento de imágenes.

Lo anterior no es el único problema, el vehículo no contempla módulos para la comunicación con otros vehículos, y la documentación con que se cuenta es insuficiente. En proyectos anteriores, con este vehículo, los alumnos se enfocaron en la parte de planificación de movimiento, que le permite al vehículo identificar las líneas de los carriles y desplazarse sobre ellos. El presente trabajo pretende continuar sus avances, al abordar los problemas de percepción y comunicación. Igualmente, resolver el problema de la falta de documentación, de forma que permita que otros alumnos puedan retomarlo con mayor facilidad.

Por ende, en este trabajo se diseña un prototipo con fines educativos y experimentales sobre el tema de percepción, implementado sobre el vehículo a escala, y la simulación de una solución para la comunicación entre vehículos. Su implementación presenta restricciones importantes, como el poder de procesamiento limitado y la simplificación de los entornos de prueba. Por un lado, esta solución se puede adaptar a sistemas con recursos limitados, sobre todo en escenarios de bajo poder procesamiento que requieren soluciones en tiempo real, así como permite investigar conceptos fundamentales. Por otro lado, es importante mencionar que esta parte del trabajo no es aplicable o transferible directamente a soluciones de seguridad vial por las restricciones mencionadas. Independientemente esto, el planteamiento hipotético

desarrollado dada la naturaleza del prototipo de prueba y sus escenarios de actuación, incluyó la prevención de accidentes.

Los avances alcanzados en los vehículos autónomos pueden tener un efecto transformador en la vida de las personas, al mejorar la seguridad al volante y poder reducir accidentes viales. Estos accidentes, en la mayoría de los casos son propiciados por el conductor (otros se pueden deber a fallas del vehículo o condiciones del camino). Tan solo en México en 2021 se registraron 340,415 accidentes de tránsito en zonas urbanas. Lo que son cerca de 933 accidentes diarios, de los cuales el 17.8% resultó en lesiones, y el 1.1% fue fatal. Cifras que comparadas con el año 2020 representaron un aumento del 12.8% (“Estadísticas a propósito del día mundial en recuerdo de las víctimas de accidentes de tránsito”, 2022).

### **1.3 Objetivos**

El primer objetivo del trabajo es implementar sobre el modelo de carro autónomo, un conjunto de algoritmos que le permitan la detección y rastreo de otros autos. El uso de técnicas de seguimiento nos puede ayudar a determinar si ciertos obstáculos son o no potenciales riesgos, según la trayectoria que sigan.

Como segundo objetivo, se busca simular una red vehicular VANET (Vehicular Ad-Hoc Network). Esta red tiene la finalidad de notificar la presencia de posibles obstáculos para reducir y prevenir accidentes de tránsito. Por ejemplo, consideremos dos vehículos que se aproximan a un crucero sin ser visibles entre sí. Con la ayuda de un tercero, es posible emitir

una advertencia y tomar medidas evasivas de manera oportuna. Esta misma red en un futuro podría llegar a permitir la negociación entre vehículos para la planificación de rutas.

#### **1.4 Metodología**

Como enfoque elegido para la elaboración del proyecto se buscó seguir una metodología ágil. La elegida fue programación extrema (XP), que tiene un enfoque orientado al desarrollo de software, de una manera eficaz, flexible y con un buen control en el proyecto. Como se observa en la figura 1-1, la metodología pone especial énfasis en la comunicación con el cliente para recibir retroalimentación constante y adaptar el diseño a las necesidades. En nuestro caso, el cliente es el laboratorio de robótica del ITAM, específicamente el profesor Rafael Gamboa, quien está a cargo del proyecto del vehículo autónomo AutoModelCar.

La metodología de cierta forma favorece la adaptabilidad sobre la previsibilidad, mediante la división de un problema complejo en pequeñas etapas simples. Cada una de estas etapas es acompañada de fases de pruebas desde los inicios del proyecto. Estas iteraciones rápidas presentan ventajas claras al proyecto. Por ejemplo, adaptar los métodos utilizados según los resultados obtenidos, y poder identificar de manera temprana lo que no funciona o lo que requiere de mayor atención (Pressman, 2010).

Aquí el cliente participa de forma activa en las pruebas y con mayor control en el proyecto. Las etapas cortas toman apenas días en construirse y probar el código, lo que reduce el riesgo de tener que realizar grandes cambios. Pasadas las pruebas, estas etapas se deben integrar

con el proyecto tan pronto como sea posible, a lo que se puede añadir un paso extra de limpiar el código y hacerlo fácil de entender (Pressman, 2010). En resumen, es una metodología que permite aceptar y adaptarse fácilmente a cambios, en un proceso de desarrollo con ciclos que duran pocos días.

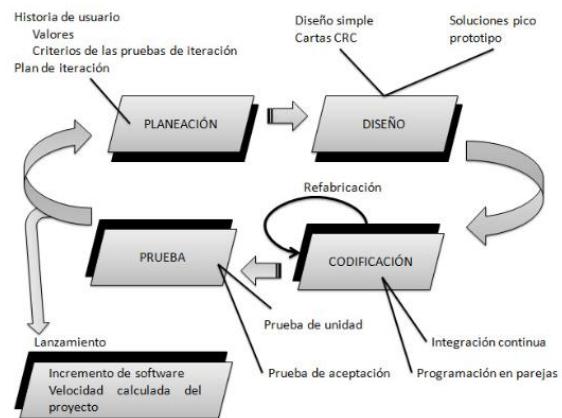


FIGURA 1-1 ETAPAS EN LA PROGRAMACIÓN EXTREMA. FUENTE: (PRESSMAN, 2010)

## **1.5 Organización del documento**

El presente trabajo se estructura en nueve capítulos que abordan los dos temas principales, la implementación sobre el modelo a escala de la detección y rastreo, y la simulación de la red VANET. En el segundo capítulo se presentan los requerimientos funcionales, no funcionales y restricciones, dadas principalmente por la plataforma de trabajo. Se detalla el alcance del trabajo, y se revisan trabajos relevantes relacionados con los sistemas de detección, junto a las investigaciones que se llevan a cabo para la creación de modelos de redes VANETS.

En el tercer capítulo se incluye el marco teórico, en el que se desarrollan los temas de detección, rastreo y redes vanet respectivamente. Se entra en mayor detalle en qué consisten estos problemas, se describen los métodos o protocolos que se trabajarán, junto a sus fundamentos matemáticos.

En el cuarto capítulo se desarrolla la primera parte del trabajo, se incluye aquí el diseño de la solución para el tema de detección y rastreo. Se definen los pasos que cubren la solución y el cómo se integrará en el carro físico. Se discuten el porqué de la elección de los métodos, junto a una comparación con la alternativa actual.

En el capítulo cinco abarca la implementación o codificación de los algoritmos para rastreo y detección. Se describen en detalle los modelos y su implementación, presentando los algoritmos tanto para uso dentro como fuera del vehículo con ROS (Robot Operating System, framework con el que está diseñado el vehículo). Este capítulo cubre la codificación de los primeros ciclos, junto con las pruebas.

El sexto capítulo cierra la primera parte del trabajo, en el que se comentan los resultados alcanzados con el programa desarrollado. En este capítulo, se analizan los tiempos de procesamiento y la precisión de los detectores implementados.

A partir del capítulo siete se desarrolla la segunda parte de este trabajo, las redes VANET. Este es otro capítulo de diseño que cubre un nuevo ciclo iterativo del proyecto. Aquí se plantea el uso para aplicaciones de seguridad y cómo se llevarán a cabo las simulaciones de los escenarios.

En el octavo capítulo se presenta la arquitectura de una red VANET, junto con la estructura y formato de los mensajes que se transmitirían, así mismo se realiza la simulación de la red. Igual que con el capítulo cinco, cubre los pasos de codificación. Mientras que los resultados se presentan en el capítulo nueve.

Como cierre del trabajo en el capítulo nueve, se repasa y comentan el trabajo desarrollado, se presentan áreas de oportunidades para mejorar. Igualmente, se incluye cómo se puede avanzar en la investigación del coche autónomo, y cómo se podría implementar físicamente en el AutoModelCar la red VANET.

Por último, en el apéndice 1 se agrega la documentación realizada del coche autónomo, junto con la configuración y cambios realizados para su funcionamiento. En el apéndice 2, se presenta una guía o manual para el uso de ROS (“es - ROS Wiki” s. f.) y OpenCV (OpenCV, 2015) enfocado en la programación del AutoModelcar.

## **2 ANÁLISIS**

En este capítulo se analizan los problemas presentados en el capítulo anterior y los requerimientos para nuestra solución. Se presentan, los requerimientos, restricciones y el alcance, dados principalmente por el mismo modelo de coche autónomo AutoModelcar. Se habla del estado del arte o trabajos hechos en el área de percepción en vehículos autónomos.

En este caso nos centramos en la detección y rastreo de otros vehículos; y en la creación de un entorno colaborativo entre vehículos, redes VANET.

### **2.1 Requerimientos**

A continuación, se detallan los requerimientos del proyecto, divididos en dos categorías: funcionales y no funcionales. Los requerimientos funcionales describen las acciones que se espera que realice el sistema propuesto, mientras que los no funcionales especifican las métricas y criterios con los que se evalúa la solución presentada.

#### **2.1.1 Requerimientos funcionales**

Para el diseño de la primera parte del trabajo, se toma en cuenta el modelo de vehículo autónomo disponible. El AutoNOMOS mini es un modelo a escala 1:10, equipado con una computadora principal Odroid XU4, la cual tiene un procesador de 8 núcleos con velocidad de 2 GHz y que ejecuta Linux y ROS Indigo. Tiene 64 GB de almacenamiento y una memoria ram de 2 GB DDR3. Adicionalmente, cuenta como auxiliar con un Arduino 3 Nano para el control de algunas de sus funciones, como el encendido del vehículo, los servomotores de las

ruedas, los leds y la unidad de medición inercial (IMU). En cambio, la Odroid controla los motores sin escobillas para las ruedas, el módulo WiFi, junto a los sensores con que cuenta. (“Hardware – AutoModelcarWiki”, 2017)

Los sensores a nuestra disposición incluyen un RPLidar 360, una cámara de ojo de pez (montada en el techo, por lo que solo sirve para navegación a través de marcas, simulando un GPS) y la cámara Intel RealSense SR300. Esta última, se compone de dos cámaras, una RGB y una de profundidad, que genera una nube de puntos (“Hardware – AutoModelcarWiki”, 2017). Por lo tanto, para la parte de percepción, se realizará procesamiento de imágenes a partir de los datos obtenidos por la cámara RGB.

El modelo de vehículo autónomo AutoModelCar está diseñado con ROS, un middleware orientado a aplicaciones para robots. En este sistema, cada funcionalidad se divide en programas llamados nodos, que se comunican entre sí mediante mensajes. ROS proporciona un nodo controlador central, lo que hace que el funcionamiento de los programas sea similar al de subprocesos. Por lo tanto, para incorporar la parte de percepción y conectarla con el resto de los programas del vehículo se requiere la programación de un nodo de ROS. Esto es, un archivo ejecutable que utiliza bibliotecas para comunicarse con otros nodos, entre ellos publican o se suscriben a mensajes que son identificados bajo un tópico. El nodo programado lleva a cabo tareas de detección y rastreo, en las que debe ser capaz de identificar correctamente otros vehículos presentes en su campo de visión.

El diseño de los modelos de detección no se centra en un tipo de vehículo en particular; sin embargo, tanto en la simulación como en físico, solo se dispone de un tipo de vehículo (y en el entorno físico, únicamente se cuenta con otro automóvil). No hay una distancia máxima a la que los vehículos deban ser identificados, pero para la ventana deslizante usada en la detección, se consideró que un tamaño mínimo de 40 píxeles sería suficiente sobre la imagen de 640x480 pixeles, de manera que no impactara mucho en el tiempo de procesamiento. Posteriormente, el sistema debe poder rastrear el movimiento de los vehículos, reconociendo entre dos fotogramas a qué vehículo se refiere, incluso cuando estos se ocultan unos detrás de otros. Además, la solución presentada debe poder integrarse con el programa seguidor de línea, desarrollado anteriormente por alumnos para el modelo a escala.

En el punto de redes VANET, se debe comprobar el funcionamiento y utilidad de protocolos como “Ad hoc On-Demand Distance Vector” (AODV) (Das, Perkins, y Belding-Royer 2003) y “Dedicated Short Range Communications” (DSRC) (“IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications” 2012) en la comunicación entre vehículos. Se deben poder identificar correctamente los requerimientos necesarios para alcanzar una comunicación fiable entre nodos (en la parte de la red VANET, por nodos nos referimos a los vehículos). Estos requerimientos, como el intervalo entre transmisión, se deben verificar en entornos cambiantes al variar la densidad de nodos y la velocidad de estos. En DSRC se prueba con diferentes densidades de nodos, cómo se comunican entre sí, al

compartir sus detecciones para la prevención de accidentes. En AODV se verifica la comunicación entre dos nodos para coordinación, una vez están enterados de la presencia del otro. Esto con el fin, de verificar la fiabilidad de ambos protocolos para la prevención de accidentes.

### **2.1.2 Requerimientos no funcionales**

Para verificar la implementación sobre el modelo a escala, la ejecución de los nodos debe ser rápida, medida por el tiempo que tarda en procesar un fotograma. Se busca lograr la mayor frecuencia posible para obtener resultados en tiempo real, con un objetivo de al menos 5 Hz debido al sistema utilizado. Además, los detectores deben ser eficientes; se evaluará su precisión y exactitud mediante un conjunto de validación. Por medio de las curvas ROC y PRC, y ajustando los umbrales, se busca alcanzar una buena relación entre verdaderos positivos y negativos, que nos permita detectar la mayoría de los vehículos. Asimismo, es fundamental que la programación desarrollada sea compatible tanto con el simulador como con el modelo a escala.

Por el lado de la red VANET, su diseño se quiere que esté basado en estándares. Dado su objetivo para aplicaciones de seguridad, la comunicación debe ser fiable. Para ello se evalúan parámetros como RTT, del que se requiere que sea menor a 200 ms, pues entre mayor sea, más distancia recorren los vehículos en ese intervalo. Otro parámetro importante, es el tiempo de encolamiento, pues deben lograr transmitir su mensaje antes de que expire y la información ya sea vieja.

## 2.2 Restricciones

Debido a que el principal objetivo es presentar la implementación sobre el modelo a escala AutoNOMOS, se presentan algunas limitaciones a la hora de elegir qué métodos usar. En primer lugar, el modelo viene con la versión de ROS Indigo para crear los nodos, el cual permite programarlos tanto en C++ como Python. En el lado del simulador, se tienen más opciones para la versión de ROS, como Noetic.

Adicionalmente, tanto el carro, como el simulador nos limitan en cuanto a los sensores que podemos usar, el formato e información de los mensajes que nos proveen, como la cámara de profundidad. Otra limitación importante se refiere a la versión de OpenCV utilizada: el vehículo cuenta con la versión 2.4 mientras que el simulador usa la versión más nueva, la 4. Estas son bibliotecas de visión por computadora que se utilizarán para el procesamiento de imágenes y la aplicación de técnicas de aprendizaje de máquina.

Una restricción adicional en los nodos se relaciona con la medición de los tiempos y la selección de los métodos, lo cual depende tanto de la capacidad de procesamiento del vehículo como de la computadora. Esta limitación afecta la cantidad de pruebas que se pueden realizar y la posibilidad de probar otros enfoques o alternativas. En el contexto de las redes VANET, si bien la elección del simulador es libre, estamos restringidos a las implementaciones de los protocolos que están incorporados en él.

## 2.3 Alcance

Aunque el problema es muy amplio, el enfoque elegido busca mantenerse en un caso más particular. Sin abordar la detección en entornos con problemas de iluminación, pero sí en presencia de múltiples vehículos. Considerando las capacidades del modelo a escala AutoModelCar, la solución debe ser un sistema de detección que funcione en tiempo real y sea ligero computacionalmente. Por otro lado, el sistema de rastreo debe ser capaz de identificar que un objeto es el mismo entre dos observaciones distintas, además de manejar detecciones incorrectas u observaciones faltantes, ya que estas pueden generar errores en la predicción.

Del lado de la red VANET, por razones de tiempo, únicamente nos limitamos a experimentar con AODV e IEEE 802.11p. Sobre estos nos enfocamos en tareas relacionadas a la prevención de accidentes. Con estos protocolos, nos limitamos a la comunicación entre vehículos (V2V), sin considerar el uso de infraestructura adicional. Esto busca sentar las bases para el modelo del carro, más no llega a implementarse.

Para las simulaciones, solamente se estudian un par de escenarios. En ellos los nodos recorren una misma distancia a diferentes velocidades (se comprueba su fiabilidad para comunicarse a 100 y 200 metros antes de un cruce).

## **2.4 Trabajos relacionados**

En la investigación sobre vehículos autónomos, una de las áreas de gran relevancia y que ha experimentado avances significativos en los últimos años es la visión por computadora. En este campo, existen dos enfoques principales: el enfoque modular y el enfoque monolítico de extremo a extremo.

El enfoque modular, ampliamente respaldado, consiste en partir de un modelo complejo de altas dimensiones y reducirlo a variables controlables de bajas dimensiones. Cada uno de ellos, separables en módulos que pueden entrenarse y probarse de forma independiente (por "módulos", nos referimos a fragmentos de programas que funcionan de manera autónoma y realizan tareas específicas, los cuales se interconectan con otros mediante entradas y salidas). Estos módulos suelen dividirse en percepción, análisis del entorno, planificación de trayectorias y control del vehículo (Janai et al. 2021).

Para la percepción y el análisis, se utilizan comúnmente técnicas de aprendizaje automático como redes neuronales, mientras que para la planificación y el control se recurre a enfoques tradicionales como máquinas de estado, algoritmos de búsqueda y modelos de control. Si bien, este enfoque modular permite un resultado más fácil e interpretable para las personas, no necesariamente es más eficiente o rápido que el enfoque monolítico (Janai et al. 2021).

En el ámbito de la percepción y análisis del entorno, desde el enfoque de aprendizaje de máquina tradicional, la detección de objetos se reduce a tareas de clasificación y el uso de

ventanas deslizantes. Para la clasificación, se propone la combinación de un conjunto de clasificadores de cascada, e histograma de gradientes orientados con máquinas de soporte vectorial (HOG+SVM), propuestas encontradas en artículos como (Bougharriou, et al, 2017) y (Feng Han et al, 2006). Trabajos en los que se propone el uso de HOG con SVM, principalmente como un algoritmo ligero y rápido. Los cuales funcionan relativamente bien con pequeños conjuntos de datos, son baratos computacionalmente y fáciles de interpretar.

Aun así, el enfoque dominante actualmente son las redes neuronales convolucionales, que se utilizan para la detección completa o por partes de objetos. Esto es gracias a que son más adaptables, transferibles, de mayor precisión y escalan con los datos. Aunque son muy efectivas, aún pueden ser insuficientes, especialmente en entornos nuevos o con condiciones climáticas variables.

Pensando específicamente en el vehículo, se han utilizado distintas técnicas para las tareas de reconocimiento y evasión de obstáculos. Una de ellas realizada en el ITAM que usa en su totalidad redes neuronales, pero solo se implementó en el simulador, no en el modelo a escala (Zetina, 2019). Otros ejemplos suelen confiar esta tarea exclusivamente al uso del LiDAR, pero no llevan pasos para reconocer el objeto, ni rastrearlo (Ayala, 2019). Por último, orientado más al reconocimiento de señales (no de obstáculos), se suelen usar métodos como filtros de cascada (Pérez, 2017) o árboles kd (Pérez, 2023).

Con el fin de mejorar la seguridad tras el volante, el campo de las comunicaciones entre vehículos también ha experimentado un notable avance. Se han introducido nuevos

protocolos, tecnologías y técnicas para lograr tasas de transmisión más altas, menor latencia y mayor confiabilidad. Esto ha sido posible en parte gracias a tecnologías como 5G, fog computing y, anteriormente, 802.11p. Estos avances han abierto la puerta a nuevas aplicaciones, como el uso de estas tecnologías en redes VANET. En este contexto, se observan dos tendencias: la comunicación vehículo a infraestructura y la comunicación vehículo a vehículo.

En vehículo a vehículo se ha explorado desde la tercera generación de telefonía celular. A manera de servicios que permiten a dispositivos comunicarse entre sí cuando están en proximidad (Tariq Islam y Cheolhyeon Kwon, 2022). El debate aquí viene desde usar bandas licenciadas (asignar nuevos espacios o reusarlas), o no licenciadas; qué técnicas de acceso y asignación de recursos usar. Así como el uso de enfoques tradicionales o aquellos basados en aprendizaje de máquina.

Entre los avances se ha logrado la estandarización de algunos protocolos como DSRC. Sobre él se han reservado inicialmente 75 MHz en la banda de 5.9 GHz para usuarios con licencia en aplicaciones de sistemas de transporte inteligente. Pero debido a la falta de adopción, se reacomodaron 45 de los 75 MHz en la banda de 5.8 GHz para usuarios sin licencia.

Las investigaciones actuales se centran en diversos tipos de tecnología. Por un lado, se consideran las tecnologías celulares, que ofrecen amplia cobertura y seguridad, pero tienen un alto costo y latencia. Por otro lado, se estudia el estándar IEEE 802.11p, que opera en la banda de frecuencia 5.8-5.9 GHz y ha sido promovido por la industria automotriz de Europa

y Estados Unidos. Adicionalmente, se discuten diferentes técnicas de broadcasting, de localización y agrupamiento para hacer frente a desafíos como los cambios en topología y escalabilidad de la red. Por último, pero no menos importante, se investiga la seguridad y privacidad de estas redes, ya que la transmisión falsa de mensajes puede representar un gran peligro (Jakubiak y Koucheryavy, 2008).

### **3 MARCO TEÓRICO**

En este capítulo se aborda el tema de la detección de objetos, en que consiste y sus pasos. Se presentan y expande sobre las técnicas de detección usadas en el trabajo, así como las redes neuronales. Además, se cubre el tema de rastreo de objetos con el filtro de Kalman. Para el tema de las redes Vehicular Ad-hoc Network (VANET), se analizan las problemáticas asociadas a la comunicación entre vehículos. Se revisan los protocolos de comunicación, incluido el Ad-hoc On-demand Distance Vector (AODV). Se discuten los modelos de comunicación Vehicle-to-Everything (V2X), Vehicle-to-Vehicle (V2V) y Vehicle-to-Infrastructure (V2I), lo que permite introducir el protocolo Dedicated Short-Range Communications (DSRC), del que se analiza el contenido del protocolo en función de cada capa del modelo OSI.

#### **3.1 Detección de objetos**

En el ámbito de la detección de objetos, el enfoque clásico sigue una serie de pasos que consiste primero en una etapa de preprocessamiento de la imagen, la extracción de una región de interés, la clasificación del objeto y una etapa de verificación o refinamiento (Janai et al. 2021).

Para el preprocessamiento, se realizan tareas de ajuste de ganancias, rectificación de imagen o calibración de las cámaras (Janai et al. 2021). Típicamente, se añaden filtros, se escalan las imágenes y se convierten a otros formatos. Dependiendo del problema, también pueden añadirse filtros de bordes o emplearse técnicas de compresión de imagen.

Adicionalmente, se emplean filtros para introducir imperfecciones en los conjuntos de datos, lo que ayuda a que los modelos sean más robustos ante condiciones del mundo real. El uso de filtros de difuminado, en particular, degrada la calidad de la imagen y disminuye la dependencia de contar con conjuntos de datos más grandes. El difuminado en imágenes consiste en tomar píxeles vecinos y sacar un promedio de ellos (Nelson J. 2020). En OpenCV se presentan distintas técnicas para aplicar difuminado, algunas mejores que otros según sus aplicaciones.

- Desenfoque Gaussiano

Se usa un kernel Gaussiano en el que se especifican el ancho y altura de la matriz, junto a la desviación estándar. En él, el píxel central recibe el mayor peso, mientras los píxeles vecinos reciben pesos más pequeños al alejarse del centro. Para la creación de la matriz de convolución se usa la fórmula: donde  $x, y$  son la distancia del origen a su respectivo eje, y la desviación estándar  $\sigma$  influye que tan relevante son los píxeles vecinos sobre el pixel central.

$$G_{2D}xy\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Alternativamente se puede implementar aplicando dos veces el filtro gaussiano de una dimensión. Pero con resultado distintos a la fórmula mostrada, al no considerar los bits de las esquinas de una celda (“OpenCV: Smoothing Images” s. f.).

Después del preprocesamiento se deben extraer las regiones de interés. El método más común es el de deslizar una ventana sobre la imagen e ir escalando el filtro o la imagen. Esto puede

resultar en un proceso muy caro, según qué tan exhaustiva sea la búsqueda. Otras alternativas son búsquedas selectivas en las cuales, en lugar de recorrer toda la imagen, se seleccionan solo algunas localizaciones aproximadas que se cree contienen al objeto (Janai et al. 2021).

Una vez que se ha extraído la región de interés, se procede a la etapa de clasificación, para la cual existen diversos métodos, algunos de los cuales se describen a continuación. Los filtros en cascada son considerados clasificadores débiles debido a su baja capacidad y complejidad, lo que se refleja en una menor precisión. Sin embargo, mediante el uso de Adaboost (Freund y Schapire, 1995), es posible combinar una serie de clasificadores débiles de manera iterativa, logrando así la precisión de un clasificador fuerte. En contraste, un clasificador fuerte, como las máquinas de soporte vectorial (SVM), es capaz de realizar predicciones correctas por sí solo, ofreciendo un alto nivel de precisión.

### **3.1.1 Clasificador de cascada**

Este método utiliza una función en cascada, que consiste en un grupo de funciones que, en conjunto, forman el clasificador y se evalúan secuencialmente para cumplir ciertas condiciones. Estas funciones analizan características específicas de la imagen (píxeles) para determinar si un objeto puede pertenecer a una clase determinada. Cada filtro evalúa solo una clase, y requiere ser entrenado por un conjunto extenso de imágenes positivas y negativas (Viola y Jones 2001).

El proceso comienza con la preparación de un conjunto de imágenes positivas (que representan la clase u objeto a detectar) y negativas. A continuación, se extraen las

características de la imagen, utilizando métodos como HAAR o LBP, que permiten obtener información relevante de la imagen, como los bordes, reduciendo así la complejidad del problema en lugar de trabajar con la imagen completa.

LBP, presentado por Ojala et al. en 1996, es un método en el que se considera una celda alrededor de cada píxel, compuesta por 9 píxeles (3x3). Se compara la intensidad de estos píxeles con respecto al centro, asignando un valor de 1 a los vecinos cuya intensidad sea mayor o igual. En sentido de las manecillas del reloj, cada valor se multiplica por una potencia de 2. Esto nos permite promediar los alrededores asignando un valor entre 0 y 255 (Pietikäinen 2010),

$$LBP(xc, yc) = \sum_{p=0}^{p-1} 2^p (ip - ic) * s$$

$xc, yc$  es el píxel del centro,  $ic$  es el brillo del centro,  $ip$  el brillo de un píxel vecino,  $s$  función signo. Para calcular el valor del centro, como se observa en la figura 3-1, se puede iniciar desde cualquier píxel y movernos en sentido de las manecillas del reloj, a cada bloque asignándole un valor binario y calculando la suma. Aquí el problema se reduce en comparar dos medidas: el contraste en escala de grises y patrones espaciales locales (Pietikäinen 2010) (Rosebrock 2015).

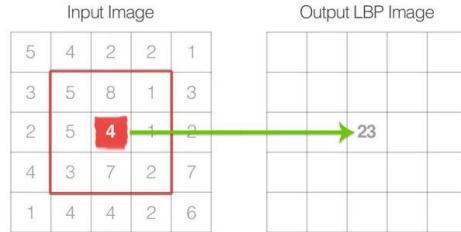


FIGURA 3-1 EJEMPLO DE LA OPERACIÓN REALIZADA EN LBP, PRIMERO SI EL VALOR DEL PIXEL ES MENOR AL CENTRO SE TOMA 1, A CONTINUACIÓN, PARTIENDO DE LA ESQUINA SUPERIOR DERECHA SE LE ASIGNA COMO  $2^0$ , DE AHÍ SE DA UNA VUELTA EN SENTIDO DE LAS MANECILLAS DEL RELOJ, AUMENTA. FUENTE: (ROSEBROCK 2015)

Calculadas las características, la mayoría resultan ser irrelevantes, por lo que para elegir las mejores se usa Adaboost. El procedimiento es aplicar todas las características obtenidas sobre el set de entrenamiento completo, y para cada característica encontrar el mejor umbral que las clasifique de forma correcta. Como no todas las características son relevantes, solo se escogen las que tengan la menor tasa de errores. En un principio todas las imágenes tienen el mismo peso, después de cada clasificación, se incrementa el peso de las imágenes que son erróneamente clasificadas y se calcula la nueva tasa. El proceso se repite hasta alcanzar la precisión indicada, o juntar cierto número de características (“OpenCV: Cascade Classifier” s. f.).

Finalmente, cada característica evaluada corresponde a un clasificador débil, que por sí solo no pueden decidir sobre la imagen completa, lo que se hace es realizar la suma ponderada de los clasificadores. Esto presenta la ventaja de que, si la imagen es negativa, la puede descartar sin evaluarla por completo, de modo que funciona por etapas (“OpenCV: Cascade Classifier” s. f.).

### 3.1.2 HOG+SVM

Se trata de la combinación de dos técnicas, máquinas de soporte vectorial o SVM que es un algoritmo de aprendizaje supervisado, e histograma de gradientes orientados HOG que permite extraer las características de la imagen.

#### HOG

Para empezar, el histograma de gradientes orientados es una técnica que cuenta las ocurrencias de vectores gradiente en una porción de una imagen, estos se componen de magnitud y ángulo. Para esto, primero se toman bloques de 3x3 píxeles, y se calculan los gradientes  $G_x$  y  $G_y$  para cada píxel, con la fórmula:

$$G_x(r, c) = I(r, c + 1) - I(r, c - 1) \quad G_y(r, c) = I(r - 1, c) - I(r + 1, c)$$

Por  $r$  se refiere a la fila, y  $c$  a la columna, lo que hace es comparar las intensidades  $I$  de los píxeles anteriores y posteriores. Estos gradientes, horizontal ( $G_x$ ) y vertical ( $G_y$ ) equivales a multiplicar por el kernel:

$$\begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline \end{array} \quad \begin{array}{|c|} \hline -1 \\ \hline 0 \\ \hline 1 \\ \hline \end{array}$$

El siguiente paso es formar los histogramas, para eso se calcula la magnitud y ángulo (dirección) de los gradientes, es decir evalúa y se mueve hacia donde hay mayor cambio en los píxeles:

$$\text{Magnitud} = \sqrt{G_x^2 + G_y^2} \quad \text{Ángulo} = \tan^{-1} \frac{Gy}{Gx}$$

La imagen, dividida en celdas, se agrupa para formar un bloque al que se le asocia un histograma con n valores diferentes llamados clases. Cada punto del histograma toma valores entre 0 a 180 grados, con 9 clases se tiene un salto de 20 grados (toma valores de 0, 20, 40 ... 360). Esta aproximación en celdas y clases permite tener una representación más compacta de la imagen. La contribución individual de las celdas que forman un bloque crea el histograma (Tyagi 2021).

## SVM

En el caso de máquinas de soporte vectorial son clasificadores que, a partir de un set de entrenamiento, se encuentra un hiperplano óptimo (figura 3-2) que separe los datos en dos clases representadas como -1 y 1. Este método es fácil de entrenar y permite una clasificación relativamente rápida.

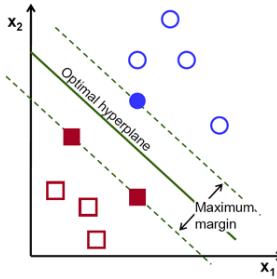


FIGURA 3-2 EJEMPLO DE LA SEPARACIÓN DE DOS VARIABLES POR UN MARGEN DE SEPARACIÓN, SE MUESTRA LA SEPARACIÓN ÓPTIMA QUE LOGRA EL MAYOR MARGEN ENTRE LAS DOS VARIABLES, FUENTE: (“OPENCV: INTRODUCTION TO SUPPORT VECTOR MACHINES” S. F.)

$$f(x) = ax^T + b$$

En este caso,  $a$  son los pesos del vector,  $b$  el sesgo, y  $x$  los datos. Para clasificar se aplica la función signo:

$$f(x) = \text{sign}(ax^T + b)$$

La idea es tener el mayor margen a la frontera posible, de tal manera que se reduzca la probabilidad de error. Para la definición del margen de separación se emplean los vectores de soporte, los cuales consisten en los ejemplos de entrenamiento más cercanos al hiperplano.

En el entrenamiento se usa una función de pérdida o hinge loss. En ella, si la predicción es incorrecta la penalización es grande cuanto más lejos del margen está. Si la predicción es correcta y está cerca del margen, recibe una pequeña penalización. Si es lejos el hinge es de 0. Para hallar los parámetros  $a$  y  $b$  se tienen dos funciones una de error de entrenamiento y un término de regularización para evitar grandes escalas. La función de costo es:

$$S(a, b, \lambda) = \frac{1}{N} \sum_{i=1}^N [\max(0, 1 - y_i * (ax_i^T + b))] + \lambda \frac{1}{2} a^T a = g(u)$$

El primer término es la función hinge loss  $\max(0, 1 - y_i * (ax_i^T + b))$  que penaliza la distancia de los datos al margen de separación,  $N$  es el tamaño del set de entrenamiento. El segundo término  $\lambda \frac{1}{2} a^T a$  es la regularización que previene que los pesos de  $a$  sean grandes, siendo  $\lambda$  una constante. Esta función de costo se minimiza por el método de descenso por gradiente para hallar los valores de  $a$  y  $b$ . En este método se define un tamaño de paso  $\eta$ , una dirección y valores iniciales aleatorios para los pesos, posteriormente se procede a iterar hasta que converja:

$$p^{(n+1)} = p^n - \eta \nabla g((u^{(n)}))$$

Las máquinas de soporte vectorial usan un hiperplano lineal para separar las clases. Cuando los objetos no pueden separarse de manera lineal, se realiza una transformación para colocar las características en un espacio de mayor dimensión. Esta transformación puede lograrse mediante varios mapeos no lineales mostrados en la figura 3-3, como la Función de Base Radial (RBF), utilizada en este trabajo (“OpenCV: Introduction to Support Vector Machines” s. f.; Kecman 2005).

Kernel Functions	Type of Classifier
$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^T \mathbf{x}_i)$	Linear, dot product, kernel, CPD
$K(\mathbf{x}, \mathbf{x}_i) = [(x^T \mathbf{x}_i) + 1]^d$	Complete polynomial of degree $d$ , PD
$K(\mathbf{x}, \mathbf{x}_i) = e^{\frac{1}{2}[(\mathbf{x}-\mathbf{x}_i)^T \Sigma^{-1}(\mathbf{x}-\mathbf{x}_i)]}$	Gaussian RBF, PD
$K(\mathbf{x}, \mathbf{x}_i) = \tanh[(\mathbf{x}^T \mathbf{x}_i) + b]^*$	Multilayer perceptron, CPD
$K(\mathbf{x}, \mathbf{x}_i) = \frac{1}{\sqrt{\ \mathbf{x}-\mathbf{x}_i\ ^2 + \beta}}$	Inverse multiquadric function, PD

\* only for certain values of  $b$ . (C)PD = (conditionally) positive definite

FIGURA 3-3 TABLA DE DIFERENTES KERNELS PARA LINEALIZAR UN PROBLEMA. FUENTE:(KECMAN 2005)

### 3.1.3 Redes neuronales convolucionales

Las redes neuronales convolucionales se especializan en el tratamiento de imágenes y audio. Son compuestas de 3 capas principales: capa convolucional que aplica una serie de filtros sobre las imágenes, cuyos pesos debe aprender y que le permiten detectar la presencia de características; capa de *pool* que reduce el tamaño y preserva solo las características importantes; alguna capa de activación como Sigmoid o ReLU para devolver valores entre 0 o 1, lo que introduce no linearidades al modelo y permite aprender características más complejas; y finalmente la capa “Fully Connected” para conectar con todas las salidas. En términos generales, al principio, las redes CNN tienden a identificar características pequeñas, pero a medida que se agregan más capas, la red puede reconocer características más complejas, llegando finalmente a identificar el objeto completo (“What Are Convolutional Neural Networks? | IBM” s. f.).

En redes neuronales se puede usar otro método para extraer la región de interés, en lugar de ventanas deslizantes, llamado *region proposal network* (Ren et al. 2015). Este permite clasificar regiones y aplicarla dentro de alguna categoría. *Region proposal* toma una imagen y como salida devuelve sets de propuestas de objetos sobre los que buscar. Esto ha permitido que ganen un impulso en su velocidad, con algoritmos rápidos como YOLO, SSD y Faster RCNN (Ren et al. 2015).

Se diferencia de los enfoques clásicos, los cuales se basan en estos fundamentos:

- Los datos se pueden modelar con un set de funciones de parámetros lineales,
- En la mayoría de los problemas de la vida real, los datos siguen un comportamiento de distribución de probabilidad normal,
- Debido al punto anterior, la estimación de parámetros se hace mediante el método de estimación de máxima verosimilitud, que normalmente se reduce a minimizar la suma de errores al cuadrado como función de costo (Kecman 2005).

Los puntos anteriores, puede resultar en suposiciones inapropiadas. Por ejemplo, en problemas de altas dimensiones donde el mapeo a una forma lineal no es adecuado (maldición de la dimensionalidad). O cuando no sigue una distribución normal, por lo que se requiere construir de otro algoritmo más efectivo (Kecman 2005).

### **3.2 Rastreo de objetos**

Dentro del problema del rastreo de objetos, el enfoque principal ha sido tomarlo como problemas de inferencia bayesiana. En él se estima una función de densidad de probabilidad del estado siguiente usando el estado actual y observaciones pasadas. Posteriormente se usa un paso de corrección o actualización, el cual, mediante retropropagación ajusta los pesos (Janai et al. 2021).

Algunos de estos métodos son el filtro de Kalman y filtro de partículas. Los cuales toman un acercamiento de espacio estado, donde se estima el estado oculto no medible de un sistema dinámico lineal. En ellos se crean sistemas clon similares al original, pero medibles. Para

esto se diseña un observador y, a medida que se agregan más muestras, el sistema clon convergerá al original. (Janai et al. 2021)

Para el caso del filtro de Kalman, nuestro sistema dinámico es expresado en espacio estado mediante matrices A, B, C, D, que en tiempo continuo se ve de la forma:

$$\dot{x}(t) = Ax(t) + Bu(t) + w(t)$$

$$y(t) = Cx(t) + v(t)$$

Donde w y v son ruido.

La formulación del problema es similar al observador de Luenberger (Luenberger D. G., 1971), pero considera además la varianza del ruido para encontrar los valores k del controlador. En el caso de nuestro problema, el objetivo es predecir el movimiento de los vehículos sobre una imagen 2D, considerando la posición xy y las velocidades. Con este problema, el filtro de Kalman debería ser suficiente. Pero en casos de sistemas dinámicos no lineales se puede usar el filtro de Kalman extendido, que realiza un paso extra para linealizar el problema.

### 3.2.1 Filtro de Kalman

Como se ha mencionado el filtro de Kalman parte de un modelo dinámico, que para este problema se deduce de las ecuaciones de movimiento:

$$x = x_0 + v_0\Delta t + \frac{1}{2}a\Delta t^2$$

Donde  $x$  es la posición del objeto,  $x_0$  la posición inicial,  $v_0$  la velocidad inicial,  $a$  la aceleración y  $\Delta t$  el intervalo de tiempo. Esta ecuación se repite para el caso de 3 dimensiones, lo que nos da un sistema de ecuaciones para  $x, y, z$ .

Los pasos del filtro de Kalman se presentan en la figura 3-4 e implican la inicialización, que en nuestro caso se refiere a la posición; luego se realiza la predicción y, finalmente, cuando se produce una nueva detección, se lleva a cabo la actualización.

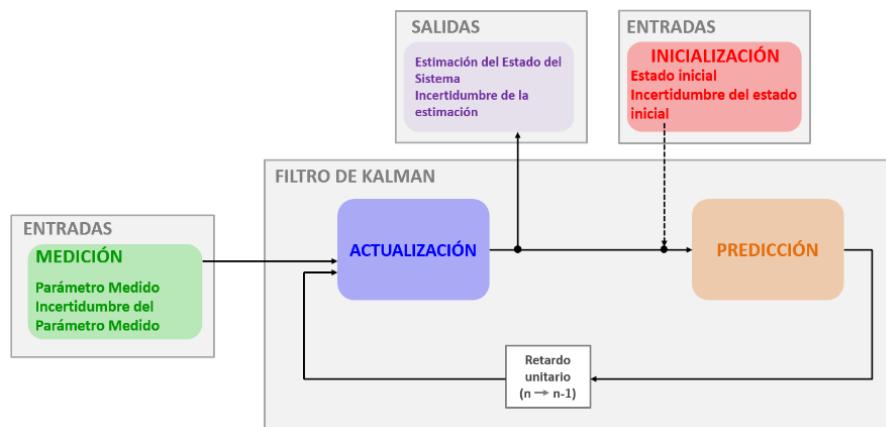


FIGURA 3-4 PASOS DEL FILTRO DE KALMAN, HAY UNA INICIALIZACIÓN A PARTIR DE LA QUE SE REALIZA LA PREDICCIÓN, EN EL SIGUIENTE PASO, SÍ SE TIENEN LAS MEDIDAS SE REALIZA LA ACTUALIZACIÓN, ADEMÁS, SE PUEDE AGREGAR UN PASO DE CONTROL. FUENTE: (BECKER S. F.)

Tomando el caso en una dimensión (para 3 dimensiones se repiten las mismas ecuaciones para cada eje de coordenadas), en el caso con aceleración constante las ecuaciones son:

$$x_{n+1,n} = x_{n,n} + \dot{x}_{n,n}\Delta t + \ddot{x}_{n,n}\frac{\Delta t^2}{2}$$

$$\dot{x}_{n+1,n} = \dot{x}_{n,n} + \ddot{x}_{n,n}\Delta t$$

$$\ddot{x}_{n+1,n} = \ddot{x}_{n,n}$$

Caso con velocidad constante

$$x_{n+1,n} = x_{n,n} + \dot{x}_{n,n}\Delta t$$

$$\dot{x}_{n+1,n} = \dot{x}_{n,n}$$

(Becker s. f.)

El filtro de Kalman parte de 5 ecuaciones:

- 1) Ecuaciones dinámicas, que son la posición, velocidad y aceleración
- 2) Ecuaciones de actualización de estado

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n(z_n - \hat{x}_{n,n-1})$$

Donde  $K_n$  es la ganancia de Kalman.  $Z_n$  es la medición.

- 3) Ecuación de ganancia del filtro

$$K_n = \frac{\text{Incertidumbre de la estimación}}{\text{Incertidumbre de la estimación} + \text{Incertidumbre de la medición}} = \frac{p_{n,n-1}}{p_{n,n-1} + r_n}$$

- 4) La actualización de incertidumbre estimada

$$p_{n,n} = (1 - K_n)p_{n,n-1}$$

- 5) Extrapolación de la incertidumbre estimada para sistemas estáticos

$$p_{n+1,n} = p_{n,n}$$

Además, se incluye para la inicialización el estado y la incertidumbre inicial del sistema  
(Becker s. f.).

### 3.2.2 Caso multivariable

Tenemos un vector de estado con las variables medidas, en el caso de 3 dimensiones con posición y velocidad se tienen 6 variables:

$$[x \ y \ z \ \dot{x} \ \dot{y} \ \dot{z}]^T$$

El cual, en notación de espacio estado, el valor del estado siguiente se obtiene como:

$$x_{n+1,n} = Fx_{n,n} + Gu_n + w_n$$

$X_{n+1}$  es el valor siguiente,  $x_n$  es el valor estimado en el tiempo n, u la variable de entrada, y  $w_n$  es el ruido. Nuestras ecuaciones de movimiento vienen representadas dentro de la matriz de transición F. Como en nuestro caso solo tenemos la predicción y no hay entradas de control, no se usa la matriz G ni la entrada u. Entonces retomando las ecuaciones de movimiento para el caso de 3 dimensiones, midiendo la posición y velocidad:

$$x_{n+1} = x_n + \dot{x}_n \Delta t$$

$$y_{n+1} = y_n + \dot{y}_n \Delta t$$

$$z_{n+1} = z_n + \dot{z}_n \Delta t$$

$$\dot{x}_{n+1} = \dot{x}_n$$

$$\dot{y}_{n+1} = \dot{y}_n$$

$$\dot{z}_{n+1} = \dot{z}_n$$

Se traducen en la matriz de transición F como:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \\ \dot{x}_{n+1} \\ \dot{y}_{n+1} \\ \dot{z}_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \\ \dot{x}_n \\ \dot{y}_n \\ \dot{z}_n \end{bmatrix}$$

(Becker s. f.)

### 3.3 Comunicación entre vehículos

Dentro del ámbito de las redes móviles ad hoc (MANET), se destaca un tipo específico conocido como redes vehiculares ad hoc (VANET), en las cuales los nodos son vehículos equipados con módulos de comunicación inalámbrica. A pesar de la amplia investigación y experimentación realizada en este campo, aún no se ha definido un modelo único o preferido, ya que las tecnologías requeridas pueden variar según los requisitos específicos de comunicación. En la tabla 3-1 se presenta una comparativa con redes móviles y de vehículos aéreos no tripulados (FANET), lo cual permite introducir la problemática inicial en una red VANET.

TABLA 3-1 COMPARACIÓN ENTRE MANET, VANET Y FANET

	MANET	VANET	FANET
Movilidad	Baja	Alta	Muy alta
Modelo de movilidad	Aleatorio	Regular	Regular para algunos caminos, principalmente con multiUAV

Densidad de nodos	Baja	Alta	Muy baja
Cambios en topología	Lento	Rápido	Rápido
Modelo de propagación	A nivel de suelo, por lo general no hay LOS (Line of sight)	A nivel de suelo, No hay LOS	En el aire, LOS
Consumo energético	Consumo eficiente	Suministro ilimitado, es proporcionado por la batería del vehículo	Consumo eficiente para mini UAV's
Poder computacional	Limitado	Alto	Alto
Localización	GPS	GPS, A-GPS, DGPS	GPS, A-GPS, DGPS

En el contexto de las redes VANET, se enfrenta el desafío de gestionar una cantidad variable de nodos que pueden desplazarse a alta velocidad según las condiciones del tráfico, y que en muchas ocasiones no cuentan con línea de visión directa. No obstante, una ventaja significativa es que el movimiento de los vehículos es predecible y está restringido a las vías existentes, lo que facilitaría la implementación de modelos de comunicación vehículo a infraestructura.

Con el fin de prevenir accidentes, a través de sus detecciones un vehículo puede alertar a otros en caso de observar algún comportamiento anormal. Por ejemplo, que otro vehículo dentro de su rango de visión se detenga, desacelere, cambie de dirección o de carril.

### **3.3.1 Patrones de comunicación**

Para compartir información entre vehículos se consideran distintos esquemas o patrones, que dependen según qué información se quiere compartir y a qué participantes les puede interesar. Aquí la información se puede compartir por beaconing, geocasting, unicasting o diseminación de información.

Se considera broadcasting como solo un salto, donde un vehículo comparte con todos los nodos que estén dentro del rango la información. A partir de este punto, los nodos pueden reenviar la información, utilizando múltiples saltos (multihop), con el objetivo de cubrir un área más amplia. En este proceso, se puede recurrir a la infraestructura de la carretera para aumentar la robustez de la comunicación. El broadcasting se puede llevar a cabo mediante distintos métodos:

- Inundación: Todos los nodos que reciben la información la reenvían, lo que puede resultar en la duplicación de mensajes. Para mitigar este efecto, se puede implementar un contador (Time to Live - TTL) que indique cuántas veces un mensaje puede ser reenviado antes de ser descartado.
- Beaconing: Se considera como el envío periódico de información, lo que ayuda a limitar la densidad de información en la red.
- Geocasting: Los nodos reenvían la información solo dentro de un área geográfica.

Si bien todos estos métodos son considerados de inundación, presentan mecanismos de control para evitar problemas de congestión. Otras alternativas para limitar la cantidad de mensajes que son retransmitidos dentro de la red, por ejemplo, se propone realizar broadcast según la densidad de tráfico, de forma que, con mayor tráfico, menos autos deben retransmitir. También se sugiere utilizar una estrategia en la que la probabilidad de retransmitir un mensaje disminuya a medida que un nodo se aleje del nodo original (Hartenstein y Laberteaux 2010).

Entre los métodos derivados de inundación, se encuentran los protocolos gossip donde un nodo se empareja con otros nodos de forma aleatoria (en lugar de retransmitir a todos dentro de su rango), estos nodos que reciben la información a su vez están emparejados con otros a los que les comparten la información. Lo que nos lleva a otra alternativa que se explora, la idea de formar grupos (agrupamiento). Aquí hay nodos especiales que se dedican a retransmitir la información dentro de su grupo (cluster), y nodos repetidores (relay) que pertenecen a varios grupos. En general estos grupos se hacen de forma geográfica, y los repetidores son solo los nodos donde se produce superposición entre ellos.

Por último, hay propuestas basadas en unicast que pueden ser efectivas para ciertas aplicaciones que requieren que dos nodos se conecten directamente, como coordinarse, como se plantea con el uso de AODV. Sin embargo, para la prevención de accidentes, por sí solo no es ideal ya que es muy lento y presenta dificultades de escalabilidad.

El protocolo AODV o “Ad-hoc on demand distance vector routing” (estándar de la IETF: RFC 3561), es un protocolo bajo demanda que soporta tanto unicast como multicast. Cuando se requiere transmitir un mensaje, se establece una ruta a la que, como los nodos están en movimiento, periódicamente se le debe dar mantenimiento. Es reactivo porque la ruta solo se crea cuando se necesita. AODV usa mensajes de control que incluyen:

- RREQ para solicitar ruta a un nodo, cada nodo lo retransmitirá hasta llegar al objetivo o algún nodo que conozca una ruta y esté fresca.
- RREP es la respuesta de algún nodo con la ruta, ya sea el nodo objetivo o un intermediario.
- RERR mensajes de error para avisar que una ruta ya no es válida porque un nodo ya no está activo (Aswathy 2012).

En su aplicación como multicast, usa los mismos mensajes para descubrir un grupo y crear la topología con los nodos pertenecientes. Periódicamente se mandan también mensajes de actualización RREP para mantener las rutas frescas y reparar links rotos. Por último, para el enrutamiento se pueden usar diferentes parámetros como número de saltos, o probabilidad de propagación (Janne Salmi 2000).

A pesar de ser ampliamente utilizado, el protocolo AODV presenta ciertos problemas de escalabilidad en redes extensas, lo que puede resultar costoso en términos de almacenamiento para los nodos, ya que deben mantener tablas de enrutamiento. Además, puede ocurrir

congestión, debido a que se envían de forma constante mensajes de búsqueda, respuesta y de errores.

Una solución extra que se propone es combinarlo con lo que se llama protocolos de agregación. En lugar de retransmitir la información como se recibe, los nodos de la red reciben y procesan la información para enviar un resumen de ésta a los demás nodos conectados a la red.

### **3.3.2 Modelos V2X, V2I y V2V**

En el ámbito de la comunicación vehicular, se encuentran diversos modelos con distintas aplicaciones. El primero de ellos es “Vehicle-to-Infrastructure” (V2I), que se refiere a la comunicación entre vehículos y la infraestructura vial. En este modelo, la infraestructura desempeña el rol de coordinador, y las principales preocupaciones son la latencia y la cobertura de la comunicación, dado que requiere la instalación de antenas en el camino y que actúan como intermediarias entre los nodos.

Por otro lado, “Vehicle-to-Vehicle” (V2V) hace referencia a la comunicación directa entre vehículos (peer to peer), sin necesidad de un nodo central. Para realizar broadcasting, los nodos deben notificar al medio sus intenciones de transmitir.

Finalmente, “Vehicle-to-Everything” (V2X) se refiere a la comunicación entre vehículos y cualquier otro elemento, abarcando no solo los modelos anteriores sino también la comunicación con los dispositivos móviles de los peatones. Por ejemplo, en la Tabla 3-2 se

presentan algunas aplicaciones que pueden utilizar estos modelos y el tipo de comunicación que requieren.

**TABLA 3-1 FUENTE: (HARTENSTEIN Y LABERTEAUX, 2010) ALGUNAS APLICACIONES Y SU MODELO SUGERIDO**

Ocho aplicaciones cooperativas de seguridad vehicular de alta prioridad, elegidas por la “National Highway Traffic Safety Administration” y la “Crash Avoidance Metrics Partnership”

Aplicación	Tipo comunicación	Frecuencia (Hz)	Latencia máxima (ms)	Datos transmitidos	Rango (m)
Violación de señal de tráfico	V2I	10	100	Señal, momento, ubicación, dirección, geometría de la carretera	250
Advertencia de velocidad en curva	V2I	1	1000	Ubicación de la curva, curvatura, pendiente, límite de velocidad, superficie	200
Luces de frenado de emergencia	V2V	10	100	Ubicación, rumbo, velocidad, aceleración	200
Detección previa a un impacto	V2V	50	20	Tipo de vehículo, ubicación, rumbo, velocidad, aceleración, velocidad de rotación	50
Colisión en frente	V2V	10	100	Tipo de vehículo, ubicación, rumbo, velocidad, aceleración, velocidad de rotación	150
Asistencia de giro	V2I o V2V	10	100	Señal, momento, ubicación, dirección, geometría de la carretera.	300
Advertencia de cambio de carril	V2V	10	100	Ubicación, velocidad, rumbo, aceleración, estado de las luces de giro	150
Asistencia de señal de pare	V2I o V2V	10	100	Ubicación, velocidad, rumbo, advertencia	300

Aquí nos centraremos en el modelo V2V donde hay dos tecnologías dominantes: IEEE 802.11p basado en DSRC (contempla capas física y MAC), y C-V2X de la 3GPP que usa tecnología celular basada en 4G LTE y 5G.

El elegido para experimentar fue DSRC, debido a que es una tecnología con más tiempo de adopción, sobre la que se han creado estándares y regulaciones en países. Comparado con C-V2X, posee mayor rango (hasta 1km), soporta altas velocidades de hasta 500 km/h, presenta menor latencia (0.4 ms en promedio vs 1ms). Además, es asíncrono, no requiere un paso previo de sincronización. Por otro lado, hay desventajas como el número de canales limitados y que el volumen de datos que puede transmitir es menor. Igualmente, C-V2X podría tener ventajas frente a obstáculos, ya que usa otra modulación (sc-dfm), y es más fácil aprovechar infraestructura celular (Kyle Barratt, 2021).

### 3.3.3 DSRC

En la figura 3-5 se observa que las capas que conforma el protocolo DSRC o IEEE 802.11p se compone de las capas físicas y MAC. Su funcionamiento se puede extender a otras capas superiores a través de otros protocolos, como lo es WAVE.

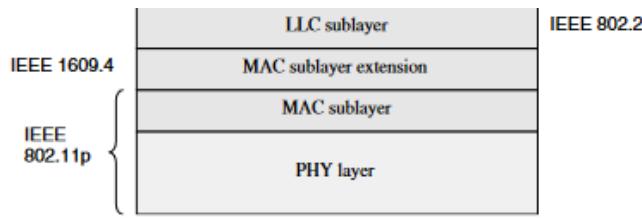


FIGURA 3-5 FUENTE: (HARTENSTEIN Y LABERTEAUX 2010) CAPAS DEL ESTÁNDAR DSRC, SE COMPONE DE DOS PARTES PRINCIPALES, EL ESTÁNDAR IEEE 802.11P E IEEE 1609 O WAVE

### Capa física

DSRC o direct short range communications, es un estándar del IEEE publicado en 2002 denominado IEEE 802.11p Acceso Inalámbrico en Ambientes Vehiculares (WAVE). Parte del estándar de Wi-Fi 802.11 para aplicaciones de transporte inteligente, como la comunicación entre vehículos e infraestructura. Para el uso del estándar, el IFT en concordancia con la UIT designó la banda de frecuencias 5850-5925 MHz para servicios fijos, fijos por satélite y móvil. DSRC es un servicio que engloba comunicaciones V2V y V2I, con el objetivo de prevenir accidentes.

Como se observa en la figura 3-6, la banda considera el uso de canales de 10 MHz con motivos de robustez, con modulación QAM o OFDM, y CSMA/CA como método de control

Parámetro	ETSI (Europa)	IEEE (Estados Unidos)	TTA (Corea del Sur)
Rango de frecuencia de operación	5855-5925 MHz	5850-5925 MHz	5855- 5925 MHz (Sistema piloto)
Ancho de banda de canal de RF	10 MHz	10 o 20 MHz	Menos de 10 MHz 23 dBm
Potencia de transmisión / PIRE RF	Máx. 33 dBm (PIRE)	-	
Modulación	BPSK OFDM, QPSK OFDM, 16QAM OFDM, 64QAM OFDM	64-QAM-OFDM 16-QAM-OFDM QPSK-OFDM BPSK-OFDM	BPSK OFDM, QPSK OFDM, 16QAM OFDM. Opción: QAM-64
Velocidades de transmisión	3 Mbit/s. 4.5 Mbit/s.	3, 4.5, 6, 9, 12, 18,	3, 4.5, 6, 9, 12,

FIGURA 3-6 CARACTERÍSTICAS DE LA BANDA 5850-5925 MHZ. FUENTE: (“IDENTIFICACIÓN DE NECESIDADES DE ESPECTRO PARA SISTEMAS DE TRANSPORTE INTELIGENTE EN LA BANDA 5850-5925 MHZ”, 2021)

de acceso al medio (“identificación de necesidades de espectro para sistemas de transporte inteligente en la banda 5850-5925 MHz”, 2021).

### **Capa MAC**

El estándar IEEE 802.11p abarca también la segunda capa o de control de acceso al medio (MAC), en la que se define el uso del medio de transmisión y de organización. Aquí al ser un medio inalámbrico sin un coordinador, se define el uso de CSMA/CA o acceso al medio por detección de portadora y prevención de colisiones.

DSRC hace una simplificación eliminando pasos como la sincronización e iniciación de la conexión, lo que reduce encabezados, Aun así, son pasos que serían útiles para dar seguridad a las conexiones y que se investigan en forma de lograr una configuración rápida del enlace.

El encabezado MAC, como se ve en la figura 3-7, es el mismo que en Wi-Fi y consiste en encabezados de control de la trama, direcciones MAC, duración, el cuerpo del mensaje, y FCS que es la secuencia de verificación de trama (Kenney 2011).

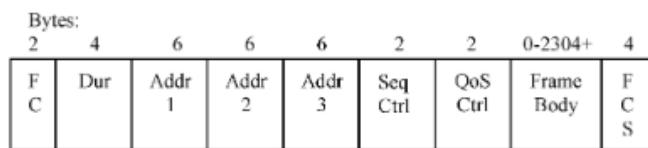


FIGURA 3-7 FUENTE: (KENNEY 2011) ENCABEZADOS DE CAPA 2

En el contexto de CSMA/CA, cuando un vehículo desea iniciar una transmisión, primero escucha el canal durante un intervalo de tiempo denominado "Arbitration Inter-Frame

Spacing" (AIFS). Este intervalo define un período durante el cual el vehículo debe monitorear el canal según la prioridad del mensaje. Si el canal se ocupa mientras se encuentra en el AIFS, el vehículo espera un tiempo determinado (random backoff value con una distribución exponencial) antes de intentar nuevamente monitorear el canal. Si el canal está libre, el vehículo procede a transmitir el mensaje. La implementación de CSMA/CA puede o no incluir el uso de retransmisiones para garantizar la entrega del mensaje en caso de colisiones u otros problemas de transmisión.

No obstante, aún hace falta encontrar un mejor método de acceso al medio, pues CSMA/CA en redes Ad-hoc no resulta idóneo para aplicaciones críticas como la prevención de accidentes. Por un lado, se encuentra el problema de los retrasos cuando el canal está ocupado, lo que significa que no garantiza la transmisión de un paquete antes de expirar. Por otro lado, la forma en que está implementado acepta la posibilidad de colisiones cuando dos nodos intentan iniciar una transmisión de manera simultánea, y no aborda el problema de la estación oculta (Khairnar y Kotecha 2013).

### **Capa lógica**

DSRC hace uso del estándar IEEE 802.2 control de enlace lógico, y se apoya del protocolo “subnetwork access protocol” (SNAP). Como se observa en la figura 3-8 se compone de LLC y SNAP, se trata de una subcapa que sirve de interfaz para el usuario en la capa de red, configurada para permitir conexiones desconocidas, en el campo EtherType se define 0x88DC para WAVE o 0x86DD para ipv6 (Kenney 2011).

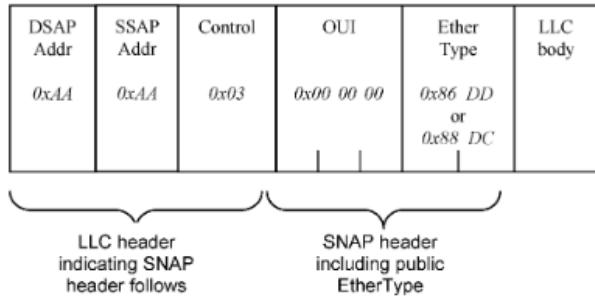


FIGURA 3-8 FUENTE: (KENNEY 2011) ENCABEZADOS DE LA SUBCAPA LLC Y SNAP

### Capas superiores

El estándar “wireless access in vehicular environments” (WAVE) o IEEE 1609 es un conjunto de protocolos de acceso inalámbrico en entornos vehiculares, que sirven como recomendación para el estándar DSRC:

- IEEE 1609.4: para operación multicanal.
- IEEE 1609.3: para servicios de red en entornos WAVE, el protocolo busca reducir la sobrecarga de UDP/IPv6, con mensajes cortos llamados WSMP que permitan transmisiones de un solo salto eficientes, el contenido se observa en la figura 3-9. Contiene la versión de WSMP usada, PSID el identificador para el tipo de servicio que se usa, en la extensión se incluye el número del canal, la tasa de transmisión y potencia de transmisión, WAVE ID indica el fin de la extensión, y la longitud del segmento de datos.

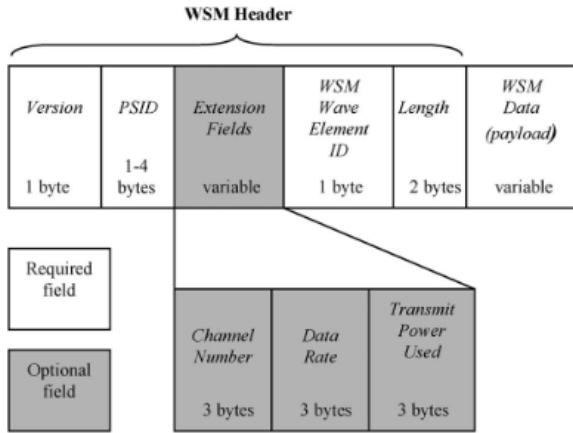


FIGURA 3-9 FUENTE: (KENNEY 2011) FORMATO DE WAVE SHORT MESSAGE

- IEEE 1609.2: para servicios de seguridad, el transmisor debe tener una firma digital que garantice que tiene la autoridad para enviar mensajes y el contenido no ha sido alterado. El estándar usa el algoritmo ECDSA para firmas digitales de curva elíptica (Kenney 2011).

## **4 DISEÑO**

En este capítulo se aborda el tema de la detección de objetos, junto con algunos pasos previos para el preprocesamiento de imágenes. Se presentan las técnicas de detección usadas en el trabajo y se comparan contra las redes neuronales. Además, se cubre el tema de rastreo de objetos, visto desde técnicas tradicionales. Principalmente se trata el filtro de Kalman, el cual se puede resumir como la predicción del estado siguiente, a partir de la medición de los estados anteriores, con una etapa de corrección al recibir una nueva lectura.

### **4.1 Arquitectura**

La primera parte del trabajo se centra en el desarrollo de un nodo de ROS encargado de las tareas de percepción para el carro. Problema que se separa en dos partes:

- Detección de objetos, en este caso vehículos,
- Rastreo de los objetos, los vehículos detectados en el paso anterior.

Por lo tanto, se procede a programar un nodo de ROS para el vehículo a escala AutoModelCar. Si bien podría separarse en dos nodos (archivos), esto no es conveniente por lo interconectados que están, pues la parte de rastreo se usa también en las tareas de detección. Asimismo, dadas las restricciones de alcanzar un resultado rápido en el carro, nos limitamos a programarlo en C++. La alternativa, Python, al ser un lenguaje interpretado en lugar de compilado, conlleva a una pérdida notable de rendimiento.

Para el nodo, se desarrolla un único programa con tres versiones distintas: uno para pruebas utilizando video o imágenes, otro para el simulador y el último para el modelo a escala. La diferencia entre las dos últimas versiones radica en los cambios significativos entre el carro y el simulador. El primero utiliza ROS Indigo y OpenCV 2.4, mientras que el segundo utiliza ROS Noetic y OpenCV 4. Esta disparidad se debe a la falta de estandarización en ROS, el cual se organiza en distribuciones, cada una con diferentes versiones de paquetes y requisitos de compatibilidad. De manera similar, OpenCV es una biblioteca de visión por computadora de código abierto que se maneja mediante versiones. Por la falta de estandarización, algunos elementos como modelos entrenados o funciones no son compatibles entre versiones. En específico, los cambios se centran en el entrenamiento de las máquinas de soporte vectorial y el cómo se exporta el modelo.

Además, se necesitan otros programas complementarios para el entrenamiento de los detectores, la realización de pruebas y ajustes en el rastreo. Por último, se desarrolla un nodo simple para el vehículo que permite la evasión de obstáculos, el cual se integra con el nodo de conducción previamente desarrollado por Edgar Alejandro y otros miembros del laboratorio Eagle Knights, denominado "lane\_navigation simulador\_gary.launch" (Granados Osegueda, 2019). Dicho nodo, se encarga de la navegación a través de carriles, a partir de información obtenida por la cámara.

Los nodos programados para el carro se observan en la figura 4-1. En ella, la cámara comunica su imagen mediante un mensaje con tópico: /app/camera/rgb/image\_raw, al que se

suscribe el nodo programado de detección y rastreo. Por tópico se refiere al canal de comunicación entre nodos, equivale a un identificador para los mensajes que corresponden a un mismo contexto. Este nodo, publica un arreglo con las detecciones que es leído por changeLane, que se encarga de ver si debe cambiar de carril. En general, la conducción sobre los carriles está controlado por los nodos lane\_navigation. Por lo que, en el cambio de carril se debe detener el funcionamiento del seguidor de línea. Esto se hace a través del nodo translate\_sim\_real. Tanto translate como change\_lane se comunican con manual\_control, el cual controla los motores a partir de los mensajes de velocidad y steering.

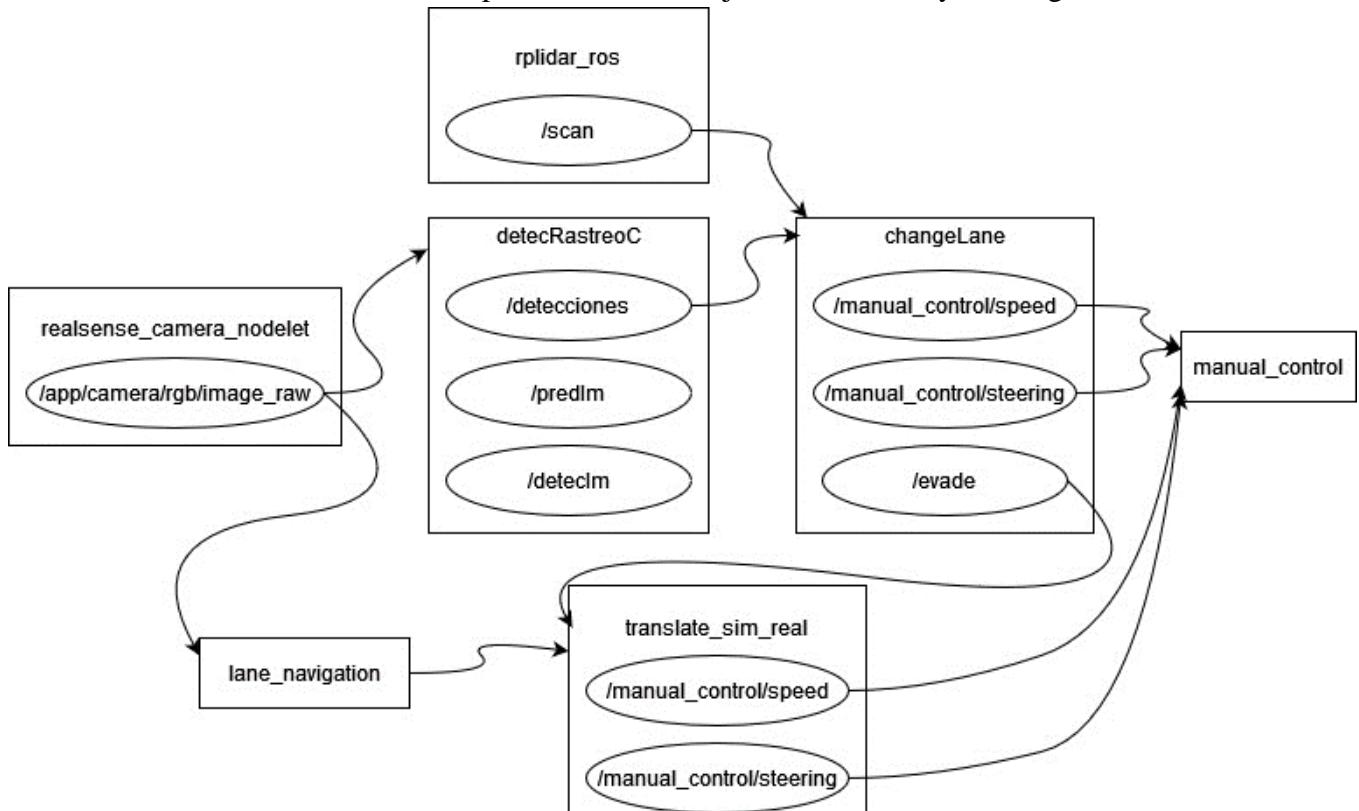


FIGURA 4-1 EN EL RECUADRO ES EL NOMBRE DEL NODO, EN LOS CÍRCULOS LOS TÓPICOS QUE PUBLICAN. SON LOS TÓPICOS LOS QUE INTERCONECTAN CON OTROS NODOS. EL NODO MANUAL\_CONTROL, A PARTIR DE LA VELOCIDAD Y STEERING SE COMUNICA CON LOS MOTORES. LANE\_NAVIGATION EN REALIDAD SON UN CONJUNTO DE NODOS QUE SE ENCARGAN DEL SEGUIMIENTO DE LÍNEA.

El vehículo autónomo está equipado con una cámara de profundidad Intel Realsense versión r200, que cuenta con una resolución de 640x480 píxeles y una frecuencia de 30 fps. Esta cámara publica un mensaje de tipo sensor\_msgs/Image, que contiene los siguientes campos:

std_msgs/msg/Header header	uint8 is_bigendian
uint32 height	uint32 step
uint32 width	uint8[] data
string encoding	

Mensajes como éste, forman parte de paquetes que contienen formas estandarizadas para la representación de los datos, en el fondo todos usan std\_msgs compuesto por tipos de datos comunes, como enteros, flotantes y strings.

Los formatos (“encodings”) usados para las imágenes no son estándares, sino formas típicas para describir cómo se representan los datos en un píxel. ROS mediante los mensajes de sensor\_msgs/Image soporta varios tipos de formatos de OpenCV, Bayer y otros comunes como MONO8, BGR8, BGRA8 y BGR16. En los casos de los formatos para imágenes a color, como los anteriormente mencionados, ROS no publica una matriz diferente para cada canal, por el contrario, los empaqueta en un solo valor. Para su uso, se debe acceder a ellos bit por bit (con ayuda del valor step), o en su lugar hacer uso de bibliotecas.

La imagen entregada por la cámara es en formato RGB8, y para usarla en OpenCV nos apoyamos de cv\_bridge. Se trata de una biblioteca de ROS que desempaquetá los datos de un mensaje imagen y los convierte a un formato de OpenCV. Por ejemplo, uno de los usados

es CV\_8UC1, el cual se refiere a que cada píxel se representa con un valor de 8 bits sin signo, con un solo canal (“Converting between ROS images and OpenCV images (C++)”, 2017).

El nodo programado publica tres mensajes. Los primeros dos para visualización, de tipo sensor\_msgs/Image, uno publica la imagen con la detección marcada, y el otro con la siguiente posición predicha. El tercer tipo de mensaje informa de las posiciones de los otros vehículos, y es el que se conecta con la evasión de obstáculos, se compone de:

detec.msg	detecTiempos.msg	detecArray.msg
uint8 x	detec posA	uint8 numDetec
uint8 y	detec[] possig	detecTiempos[] array
uint8 width		
uint8 height		

El mensaje consta de tres niveles. Primero, detecArray indica el número de detecciones y un arreglo donde se almacenan las detecciones, cada una correspondiente a un vehículo detectado. Luego, en el mensaje detecTiempos, se proporciona, para cada detección, su posición actual y las siguientes 10 posiciones predichas. Estas posiciones corresponden al mensaje detec y siguen el mismo formato que un objeto de tipo rectángulo en OpenCV: coordenadas x, y, ancho y altura. Para manipular este mensaje dentro del programa, se requiere el uso de la biblioteca cv\_bridge que contiene funciones para transformar el mensaje de la cámara a una matriz de OpenCV, así como las bibliotecas de OpenCV para el procesamiento de imágenes y aprendizaje de máquina. Además, se importan los mensajes sensor\_msgs y la biblioteca de ROS.

Se opta por utilizar una combinación de filtros de cascada con máquinas de soporte vectorial. Mediante los filtros de cascada se realiza una detección por ventana deslizante, con la que localizar candidatos. La ventaja de los filtros de cascada sobre otras técnicas se debe a su gran velocidad, con la desventaja de ser propenso a detectar falsos positivos. Para la extracción de características de la imagen en este método, se elige utilizar Local Binary Patterns (LBP) (Ojala, Pietikäinen, y Harwood 1996) debido a la rapidez en su entrenamiento. El uso de máquinas de soporte vectorial con histograma de gradientes orientados (SVM+HOG) sirve como paso de refinamiento, para descartar los falsos positivos arrojados por el filtro de cascada. Esta combinación de métodos se selecciona para maximizar la precisión y velocidad de detección con un bajo requerimiento de poder de procesamiento (Bougharriou, et al, 2017; Feng Han et al, 2006). Dado que se implementa en el vehículo con ROS Indigo, que oficialmente soporta la versión 2.4 de OpenCV lanzada en 2012, nos limitamos al uso de técnicas tradicionales de aprendizaje de máquina, pues no cuenta con modelos de deep learning. Una alternativa sería usar en el carro las versiones de OpenCV 3.0 (lanzada en 2015) junto a la 2.4.

Los pasos se detallan en la figura 4-2. El nodo recibe la imagen y realiza algunos pasos de preprocessamiento, como la conversión al formato BGR8, antes de ingresar a la detección con el filtro en cascada. Para cada detección candidata, se recorta la imagen, se reescalas y se convierten al formato MONO8. Luego, se extraen las características HOG y se realiza la predicción con SVM, lo que arrojará un resultado positivo o negativo. Dependiendo del

número de fotograma, se puede realizar la detección con la imagen completa o solo en regiones específicas.

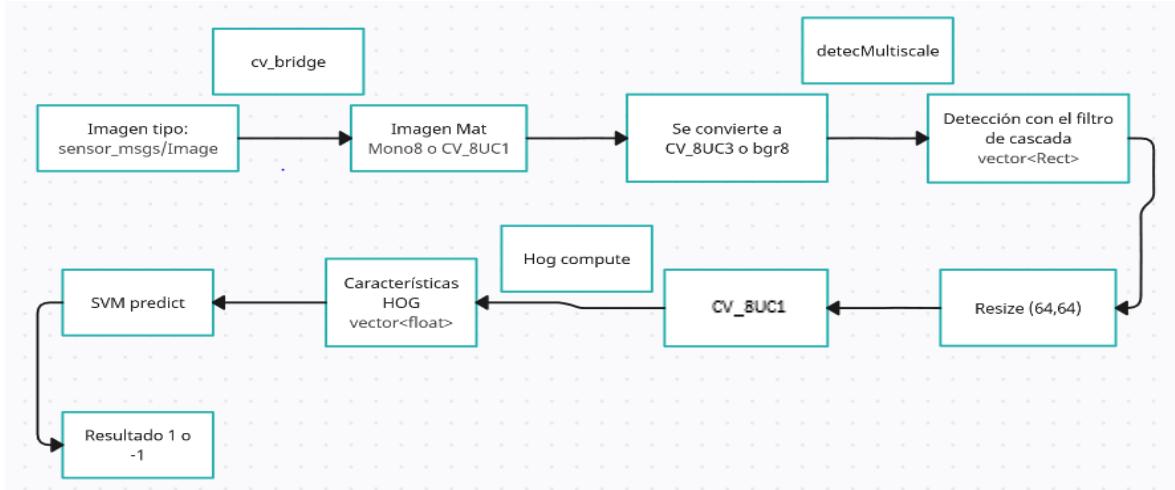


FIGURA 4-2 PASOS PARA LA DETECCIÓN

Del lado del rastreo, la elección fue el uso del filtro de Kalman sobre una imagen 2D. Al tener como paso previo la detección, no se requiere de técnicas de aprendizaje de máquina. Igualmente, otros métodos como filtro de Kalman extendido son innecesarios y costosos, pues el rastreo del movimiento se realiza sobre una imagen 2D y se puede tomar como un problema lineal.

Por lo tanto, usando el mismo nodo, una vez hecha la detección, se pasa al rastreo. En esta etapa se crea un nuevo filtro de Kalman, o se actualiza el existente (figura 4-3). Para los tres métodos, nos limitamos a los algoritmos implementados en OpenCV.

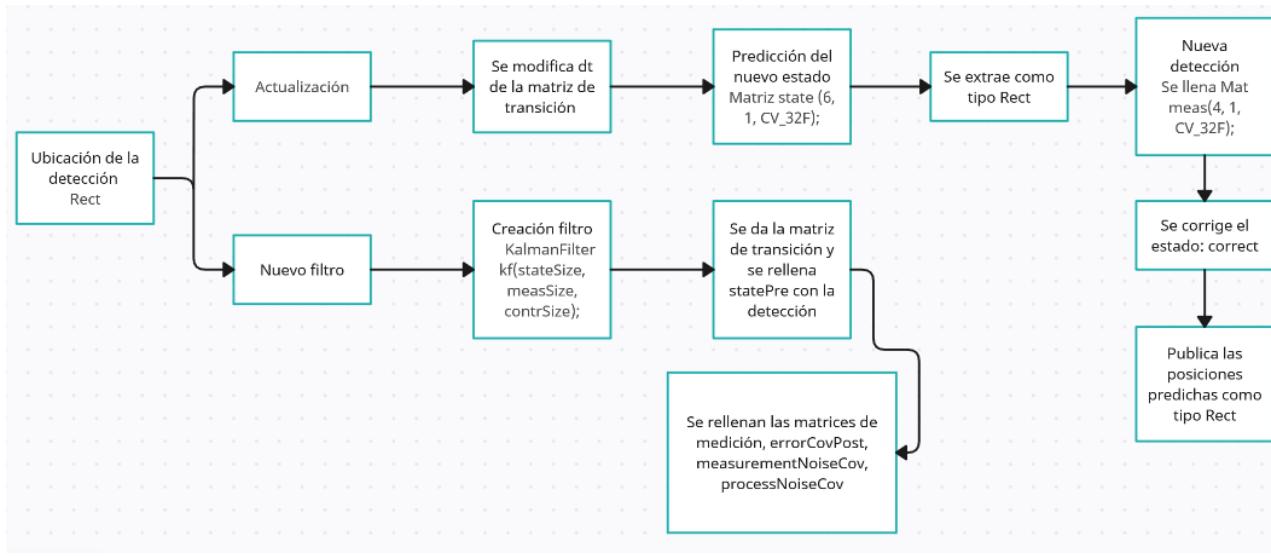


FIGURA 4-3 PASOS PARA EL RASTREO

Dentro del filtro de Kalman se tienen 3 matrices principales: la de medición, transición y control, esta última no se usa. En el caso de transición se tienen 6 variables: la posición y velocidad en (x, y), el ancho y alto de la detección. Para la de medición se tienen 4 variables: la posición en (x, y), el ancho y alto.

El primer paso para entrenar estos detectores es preparar un conjunto de datos que contenga imágenes positivas de vehículos y negativas como imágenes de árboles, de la carretera y señales. Para el entrenamiento, se desarrolla un programa que lee las imágenes, las somete a una fase de preprocesamiento (escalado y aplicación de filtros), extrae sus características y las envía al proceso de entrenamiento. Inicialmente, se prueba con un conjunto de imágenes pequeño para realizar un "grid search" (en ella se definen rangos de valores y se prueban diferentes combinaciones durante el entrenamiento) para encontrar los mejores parámetros.

Luego, se procede a entrenar con el conjunto completo de los datos. Este proceso se ilustra en la figura 4-4.

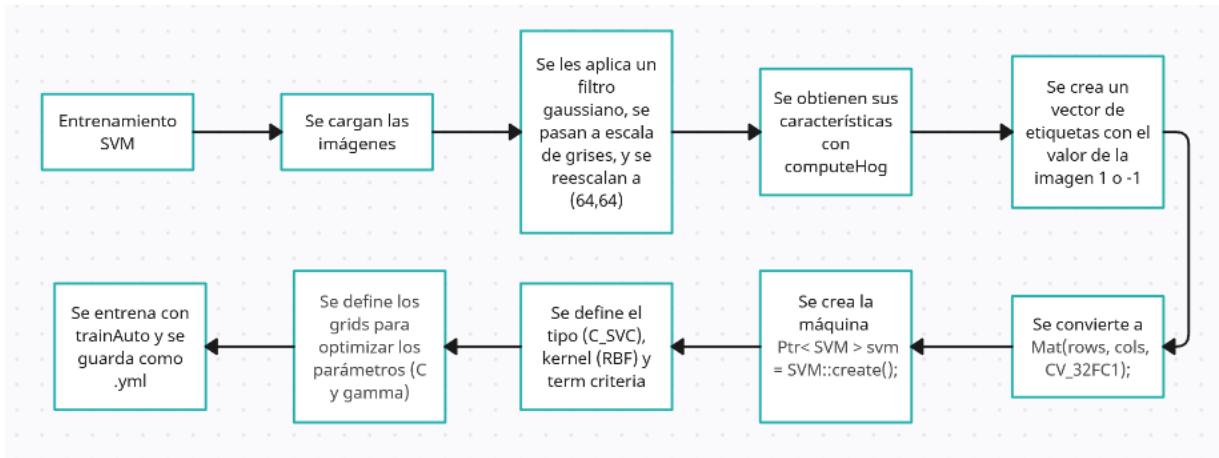


FIGURA 4-4 PASOS PARA EL ENTRENAMIENTO DE MÁQUINAS DE SOPORTE VECTORIAL

En el entrenamiento de los detectores, aunque no tengan un estándar, normalmente se sigue una serie de pasos como buenas prácticas. Desde los que involucran la colección de datos (por ejemplo, para evitar sesgos), optimización de parámetros, preprocesamiento de las imágenes (normalizar y reducir el ruido), y buenas prácticas para evaluar los resultados (como separar el set de entrenamiento del de validación). Se busca que los modelos sean replicables y con resultados interpretables.

Pasando al último nodo, para la interconexión con el seguidor de línea, se realiza un programa que cambie de carril según la detección. Este programa lee los mensajes con las detecciones definidas anteriormente y, en función de la predicción, realiza maniobras para esquivar obstáculos. Los mensajes que manda son de tipo sensor\_msgs/speed y sensor\_msgs/steering,

ambos de tipo int16. Asimismo, se manda otro mensaje también de tipo int16 para interrumpir el programa de conducción.

En las siguientes secciones se desarrollan las dos tareas principales que abarca el nodo para la percepción: detección y rastreo. En cada una de ellas se comentan los métodos usados, las razones de su elección, y otras alternativas. Sobre cada una de ellas se desarrolla con mayor profundidad en qué consisten.

## **4.2 Detección de objetos**

La detección de objetos es un componente crucial de la conducción autónoma, ya que nuestro vehículo operará en entornos con diversos elementos del tráfico, incluidos vehículos, personas y animales. En este trabajo, nos enfocamos exclusivamente en la detección de vehículos, pero las mismas técnicas pueden aplicarse a otras clases de objetos.

Ante las dificultades para la detección de objetos móviles con formas complejas desde una cámara en movimiento, el enfoque predominante se ha orientado hacia técnicas de aprendizaje de máquina. De entre ellas, la que más impulso ha recibido se trata de aprendizaje profundo con redes neuronales. Aun así, se tienen otros enfoques como el presentado, que combina máquinas de soporte vectorial y clasificadores de cascada. Estas técnicas tienen que ser lo suficientemente precisas y robustas, pues errores en la clasificación pueden presentar un gran riesgo en la seguridad de las personas. Estos errores pueden manifestarse tanto en forma de falsos positivos, donde un objeto se identifica erróneamente como un vehículo,

como de falsos negativos, donde un vehículo no es detectado. Aunque ambos tipos de errores son importantes, nuestra prioridad es la reducción de los falsos negativos, que servirá de guía para mover el margen de los detectores. Esto se debe a que falsos positivos puede llevar a frenados o maniobras evasivas innecesarias, lo que reduce la eficiencia y comodidad de los pasajeros. Sin embargo, los falsos positivos son en general menos peligrosos que los falsos negativos. Pues estos últimos, pueden resultar en colisiones o fallar al tomar medidas evasivas cuando se necesita.

En este trabajo, nos centramos en entrenar clasificadores para la detección de vehículos durante el día, con el objetivo de ser lo más generales posible para abordar problemas de oclusión, como las sombras, sin separar explícitamente este problema. Además, en situaciones nocturnas, se pueden mejorar los resultados utilizando otros tipos de cámaras.

Como se revisó en el capítulo anterior, la detección de objetos incluye una serie de pasos: el preprocessamiento de la imagen, la extracción de una región de interés, la clasificación del objeto y una etapa de verificación o refinamiento.

Para el preprocessamiento, en nuestro caso, la imagen se reescaló inicialmente para adaptarla al clasificador. Luego, se convierte a escala de grises, lo que simplifica los algoritmos y reduce la complejidad al eliminar la necesidad de aprender colores específicos. Este proceso, en cierto sentido, comprime la imagen al reducir los 3 canales de color a 1.

Además, se emplean filtros para introducir imperfecciones en los conjuntos de datos, esto únicamente se realiza sobre el entrenamiento para ayudar a que los modelos sean más

robustos. Para el trabajo, se prueba exclusivamente con el filtro gaussiano, que acelera el entrenamiento de los clasificadores al aprender de manera más rápida y general la forma de los vehículos, debido a que suprime el ruido y elimina detalles innecesarios. Aunque no se incluye en este trabajo, el uso de desenfoque de movimiento podría ser beneficioso para la detección de vehículos a altas velocidades, aunque no para aprender su forma.

El segundo paso es la extracción de las regiones de interés, que para el caso de algoritmos de aprendizaje de máquina tradicional el método más común es de ventana deslizante. Al ser un proceso que puede ser caro, para este trabajo, se deben asumir algunas cosas, como el tamaño de la ventana, la posición de los candidatos y la proporción con la cual se incrementa el tamaño de la ventana, lo que reduce el número de comparaciones.

Otra alternativa, que también se implementa en el trabajo, son búsquedas selectivas, en lugar de recorrer toda la imagen, se seleccionan solo algunas localizaciones aproximadas (Janai et al. 2021). Esta técnica utiliza al procesar algunos frames en los cuales, en lugar de realizar la detección completa, buscamos en las ubicaciones predichas por el filtro de Kalman. Adicionalmente, este enfoque se puede explotar más, pues permitiría dividir la imagen en regiones y aplicar un clasificador específico a cada una. Otra forma de implementarlo podría ser combinando el uso de LiDAR y seleccionando como regiones los obstáculos detectados.

#### **4.2.1 Técnicas de detección**

Para el trabajo se experimentó con métodos clásicos: clasificadores de cascada y máquinas de soporte vectorial (SVM) con histograma de gradientes orientados (HOG). En ellas, se ha

observado que SVM con HOG trabaja mejor a más altas resoluciones, mientras presenta un mayor tiempo de procesamiento. Del lado de filtros de cascada funciona mejor a bajas resoluciones, alcanzando resultados prácticamente en tiempo real. Otros métodos son “Scale-Invariant Feature Transform” (Lindeberg 2012) y “Speeded Up Robust Feature” (Bay, Tuytelaars, y Van Gool 2006). Todos ellos dependen de extraer manualmente la información de las imágenes, son difíciles de diseñar y limitados en sus capacidades de representación (Janai et al. 2021). La alternativa actualmente es el uso de redes neuronales convolucionales como YOLO (Redmon et al. 2016), MobileNet (Howard et al. 2017), Fast R-CNN (Girshick 2015). En las redes neuronales de preferencia se incluye también un paso de preprocessamiento de la imagen como el realizado en algoritmos tradicionales, pero también aceptan el uso de imágenes sin procesar. Usar las imágenes preprocesadas puede ayudar a reducir sesgos y mejorar el rendimiento. Hay algunas redes que ya están diseñadas para aplicar algunos pasos, como normalización.

Para ambos métodos es necesario extraer las características de la imagen. Para el método de cascada, se utiliza Local Binary Patterns (LBP), que captura características locales para describir la imagen en su totalidad, centrándose en patrones repetitivos. En cambio, para SVM se emplea el Histograma de Gradientes Orientados (HOG), que analiza la imagen de manera global, centrándose en la captura de formas y contornos.

Una alternativa es utilizar características HAAR (Viola y Jones 2001) para los filtros de cascada, las cuales aproximan las características mediante rectángulos en lugar de crear

histogramas. Este método ofrece una mayor precisión, pero debido a limitaciones de tiempo en el entrenamiento, no se utiliza en este caso. OpenCV proporciona un módulo para el entrenamiento de este tipo de clasificadores. En el nodo detector, a partir de la imagen preprocesada, se ingresa al clasificador, que utiliza la detección por ventana deslizante. La idea es que, debido a su velocidad, pueda detectar candidatos, y luego se utilice SVM como paso de refinamiento.

Para la máquina de soporte vectorial, el entrenamiento de este detector requiere hacerse manualmente, ya sea ingresando los parámetros o buscando los mejores a través de definir mallas (grids) para probar una serie de combinaciones. Previo a la implementación en el nodo, requiere probarse variando tanto el kernel usado, como la configuración de HOG (por ejemplo, en cuántas celdas dividir la imagen) para buscar los mejores resultados. Igualmente, por la complejidad del problema, se requiere de un kernel como la Función de Base Radial (RBF). Completado el entrenamiento, en el nodo se incluye únicamente como clasificador de los candidatos arrojados por cascada, para esta tarea las regiones se deben reescalar nuevamente e ingresarse para predecir la clase. Por su lentitud, no se usa para buscar dentro de la imagen.

Con los modelos entrenados, se pueden mejorar los resultados ajustando algunos parámetros. En el caso de las máquinas de soporte vectorial, es posible ajustar los resultados mediante la distancia al margen. Es importante señalar que esta distancia no se corresponde directamente con la probabilidad, aunque puede ser transformada mediante métodos como la calibración

de Platt, que se encuentra fuera del alcance de este trabajo. Por lo tanto, al ajustar el umbral de la distancia al margen, es posible aceptar un mayor número de resultados positivos a lo largo de la frontera de decisión, aunque esto implica la introducción de más errores. Para determinar el mejor valor de umbral, se pueden utilizar curvas como receiver operating characteristic curve (ROC) o precision recall curve (PRC).

Para el filtro de cascada se ajusta el parámetro minNeighbors, el cuál define cuántas detecciones superpuestas, al escalar la ventana deslizante, deben encontrarse en la misma región para que se considere una detección válida. Establecer minNeighbors en 0 implica que se acepta cualquier detección, incluso si es aislada, lo que resulta en una mayor cantidad de candidatos. Con un valor de 1 o superior, se mejora significativamente la tasa de verdaderos negativos (TN), al requerir que cada detección esté acompañada por al menos un vecino o una detección adicional en la misma área. En el caso de utilizar únicamente el clasificador de cascada, sería recomendable exigir al menos un vecino para mejorar la precisión. Pero como se usa la máquina de soporte vectorial para refinar los resultados, se aceptan todas las detecciones.

#### **4.2.2 Comparación con redes neuronales convolucionales**

El uso de redes neuronales convolucionales ha experimentado un notable avance en eficiencia, aproximándose a la capacidad de procesamiento en tiempo real, en parte gracias a la capacidad de paralelización en la GPU. Por ejemplo, el modelo Fast R-CNN ha logrado una frecuencia de detección de 0.5 Hz, Faster R-CNN con región proposal alcanza 17 Hz, y

YOLO9000 alcanza 90 Hz en imágenes de 288 x 288 píxeles y 40 Hz en imágenes de 544 x 544 (Janai et al., 2021). La capacidad de alcanzar altas frecuencias de detección en imágenes es fundamental, especialmente en el contexto de un vehículo en movimiento, donde la distancia recorrida entre dos detecciones consecutivas de la cámara puede ser considerable, lo que puede representar serios problemas de seguridad.

Por consiguiente, los métodos tradicionales, a pesar de su capacidad para lograr niveles de precisión similares a los de las redes neuronales, como SVM, se verán limitados únicamente a situaciones con recursos computacionales limitados, especialmente en aplicaciones cercanas a tiempo real. En términos generales, se puede decir que, bajo condiciones equiparables, los métodos convencionales tienden a ser más rápidos que las redes neuronales, aunque menos precisos cuando se trabaja con conjuntos de datos pequeños o moderados. Al comparar SVM con las redes neuronales, estas últimas pueden volverse más rápidas al aumentar el tamaño del conjunto de datos y al aprovechar la aceleración por hardware.

### **4.3 Rastreo de objetos**

Dentro del problema de rastreo, una vez localizado el objeto en la imagen, lo que equivale a decir que tenemos su posición estimada. El siguiente paso es predecir su movimiento y verificarlo con el siguiente fotograma, esto es asociar dos imágenes en el tiempo. Como dice en el libro de Joel Janai, se diferencia de la detección de objetos, donde los fotogramas se procesan de forma independiente. Esto permite, por ejemplo, evitar posibles colisiones al

poder mantener una distancia de frenado y adaptar la velocidad, o reconocer si otro vehículo intenta cambiar de carril.

Nuevamente aquí se presentan múltiples problemas y dificultades, que van desde errores en la detección y oclusión (un vehículo se puede ocultar detrás de otro). O en el caso de múltiples detecciones para poder asociar que una detección corresponde al mismo objeto entre dos fotogramas cuando se trata de la misma clase (Janai et al. 2021).

La opción elegida es el uso del filtro de Kalman. En nuestra implementación, a partir de las detecciones se obtiene un rectángulo que describe la posición ( $x$ ,  $y$ ) y el tamaño (ancho y altura) del objeto detectado en una imagen 2D. Se utiliza el filtro de Kalman de OpenCV para realizar el seguimiento del movimiento de los vehículos. Se estima también el tamaño del objeto al agregar dos dimensiones adicionales para predecir el ancho y la altura del rectángulo que los enmarca. Este enfoque es eficiente y resuelve un problema bien estudiado, de forma que el uso de métodos más complejos, como el filtro de Kalman extendido o el filtro de partículas, se considera innecesario, ya que añadirían complejidad al problema y aumentarían el tiempo de procesamiento.

La implementación del nodo entonces debe recibir la imagen de la cámara en formato sensor\_msgs/Image, es una imagen de 640x480 pixeles en RGB. A continuación, según el número de fotograma, se realiza el preprocesamiento y la detección de la imagen completa o solo de regiones. Las detecciones se marcan en la imagen original y se debe publicar. Posteriormente, con la ubicación de los objetos se crea un filtro de Kalman para darle

seguimiento, o se realiza la corrección de uno ya existente. Del filtro de Kalman, se publica la imagen con la predicción, y para su uso en otros nodos, un mensaje personalizado. Este mensaje incluye en un arreglo todas las detecciones, y para cada una de ellas, su predicción en los 10 siguientes pasos.

Como se implementa en el proyecto, también mediante la predicción se puede facilitar la detección de objetos. Al tomar regiones de interés según la posición del objeto estimado. Aun así, cada determinado tiempo se debe realizar la detección de un fotograma completo, no solo para detectar nuevos objetos, también porque se van acumulando errores.

Una ventaja del filtro de Kalman es su resistencia a la falta de información. Puede ayudar a compensar los falsos negativos de la detección al permitir el seguimiento de un vehículo incluso cuando se pierden algunas muestras o al proporcionar regiones en las que buscar.

A pesar de ser un sistema dinámico bien comprendido, se realizan varias simplificaciones debido a la limitada información disponible en la imagen y a los errores acarreados por la detección. Sería interesante comparar este enfoque con una solución completamente basada en redes neuronales.

El método alternativo es el filtro de partículas, donde el rastreo se hace mediante un método secuencial de Monte Carlo. Mediante la simulación se generan modelos, a los que, al introducir una perturbación, calculan la probabilidad del siguiente estado. Mientras el filtro de Kalman parte de asumir que un problema es lineal y normal (gaussiano), el filtro extendido

sirve para problemas no lineales. El filtro de partículas se aplica en problemas no lineales y no gaussianos. En lugar de derivar las ecuaciones analíticas, se usan simulaciones para generar el estado estimado, pero por esta razón es menos eficiente (Fernández Villaverde, s. f.). Estos esquemas, también reciben el nombre de rastreo a partir de la detección. En ellos el problema se presenta en la parte de la detección, cuando es incorrecta o tiene datos faltantes. Otros métodos incluyen el uso de redes neuronales.

## 5 CODIFICACIÓN

En este capítulo se presentan con detalle los pasos para implementar los métodos de detección y rastreo en el carro. Primero se comenta el entrenamiento de los modelos y luego la programación de los nodos.

### 5.1 Entrenamiento del filtro de cascada

El primer paso es el entrenamiento de los detectores, para ello se requiere tener *datasets* de imágenes que correspondan a vehículos y otras que no. Primero se comenzó por preparar un conjunto de imágenes que fueran representativas para la clase, un pequeño fragmento se muestra en la figura 5-1. Tomando muestras de las bases de KITTI (Andreas Geiger et al. 2013) y GTI (Arróspide et al. 2012), se partieron en dos sets de aproximadamente 7000 vehículos, y 8000 de no vehículos. Estos últimos correspondían a imágenes como pedazos de la carretera, sombras, árboles y señalizaciones como se ve en la figura 5-2.

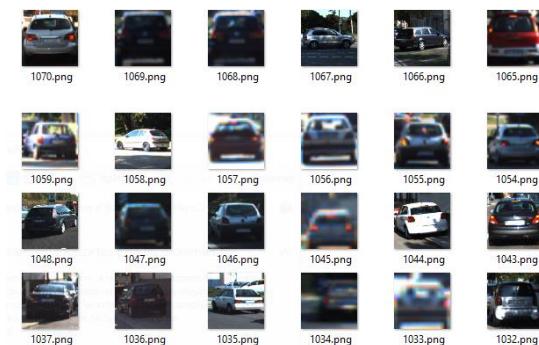


FIGURA 5-1 EJEMPLOS DE VEHÍCULOS

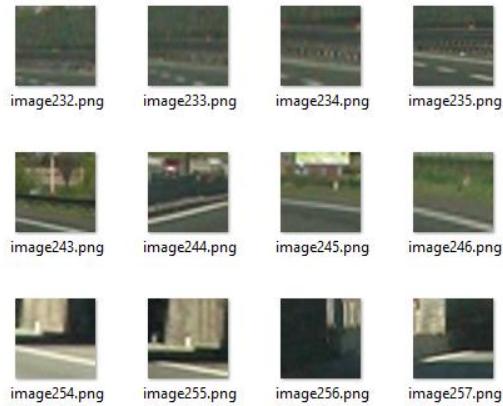


FIGURA 5-2 EJEMPLOS DE IMÁGENES NEGATIVAS PARA EL ENTRENAMIENTO

Entrenar un clasificador de cascada usando OpenCV es bastante directo. Siguiendo ("OpenCV: Cascade Classifier Training" s. f.) para las imágenes negativas se crea a un archivo .txt que contiene únicamente su nombre. Para las imágenes positivas, se requiere la creación de un archivo .dat que funcione como descripción. En él se incluyen el nombre de la imagen, número de objetos presentes, coordenadas, y dimensiones del objeto. Sobre los últimos dos puntos, solo se asignó un valor constante aproximado para todas las imágenes, suponiendo que el tamaño del vehículo era de 50 x 50 píxeles en una imagen de 64 x 64.

```
1070.png 1 7 7 50 50
```

El módulo “traincascade” se trata de un ejecutable incluido solo hasta la versión de OpenCV  
3.4. En ella, por medio de la consola se ejecutan los siguientes comandos:

```
.\opencv_createsamples.exe -info cars.info -num 7000 -w 50 -h 50 -
```

```
vec cars.vec
```

Crea un archivo para las imágenes positivas `.vec`, en el comando se define el tamaño de los sets y la dimensión de los objetos.

```
.\opencv_traincascade -data data3 -vec cars.vec -bg nocar.txt -  
    numPos 4000 -numNeg -6000 -numberStages 20 -w 50 -h 50 -  
    acceptanceRatioBreakValue 10e-05 -minHitRate 0.995 -  
    maxFalseAlarmRate 0.25 -featureType LBP
```

- data corresponde al folder destino

- vec archivo de imágenes positivas

- bg imágenes negativas

- numPos y numNeg tamaño del batch de imágenes que se usan en cada etapa. Tanto los métodos de boosting como bagging implican el entrenamiento de múltiples clasificadores, para los que es recomendable usar diferentes conjuntos de datos. Por lo que, del total de muestras, el método toma aleatoriamente solo un conjunto de datos, para cada clasificador débil.

- numberStage número máximo de etapas

- w y h tamaño de los objetos

- acceptanceRatioBreakValue: Indica cuándo se debe dejar de entrenar el modelo, se usó el ratio sugerido de 10e-05 para evitar sobreajuste de los datos

- maxFalseAlarmRate: Máxima tasa de errores aceptada, normalmente se mantiene alta con valores de hasta 0.5, se tomó 0.25%.
- minHitRate: tasa mínima de aciertos, se eligió 0.995%
- featureType: HAAR o LBP, se usó LBP para entrenar la máquina de forma más rápida. Con HAAR a costa de un mayor tiempo de entrenamiento se podrían obtener ligeramente mejores resultados.

Al final regresa el modelo como un archivo .xml.

## 5.2 Entrenamiento de SVM

En el filtro de cascada se decidió solo usar un set de autos, con el fin de únicamente identificar posibles candidatos para hacer la predicción. Esto es gracias a la velocidad de cascada comparado con SVM, al hacer la detección con ventana deslizante. De forma, que sobre las detecciones de cascada solo se realiza una predicción, que separa las imágenes en dos clases -1 o 1, si son o no autos.

Para descartar los falsos positivos, se usan máquinas de soporte vectorial. Aquí con el fin de lograr una mejor precisión, reducir el tiempo de entrenamiento y el número de muestras requeridas, se parte el problema en dos máquinas. Una para identificar vehículos desde la parte trasera y otro lateral, figuras 5-3 y 5-4 respectivamente. Es posible y recomendable separar en más clases el problema.



FIGURA 5-3 IMÁGENES DE LA PARTE TRASERA DE VEHÍCULOS



FIGURA 5-4 IMÁGENES DE VEHÍCULOS DE LADO

Para el entrenamiento, dentro de un programa en C++, se deben cargar las imágenes, extraer las características con HOG de cada una y preparar un vector con sus etiquetas. Debido a las diferencias entre las versiones de OpenCV 2.4 (utilizada en el vehículo) y OpenCV 4 (utilizada en el simulador), los modelos generados en el entrenamiento no son compatibles entre sí. Por lo tanto, es necesario entrenar el modelo por separado para cada versión.

Primero se usa un método *load\_images* (figura 5-5), en el que se realiza un pequeño preprocessamiento para las imágenes. Se pasan a blanco y negro, y se les aplica un filtro gaussiano, con el fin de remover el ruido y otros detalles, lo que ayuda a mejorar la precisión. Otros tipos de filtros que se pueden usar son por ejemplo el desenfoque de movimiento, para un efecto de velocidad, o un filtro de *sobel* para resaltar solo los bordes.

```
void load_images(const String& dirname, vector< Mat >& img_lst)
{
    Mat imgGray, imgBlur, imgRe;
    vector< String > files;
    glob(dirname, files);
    for (size_t i = 0; i < files.size(); ++i)
    {
        Mat img = imread(files[i]);
        cvtColor(img, imgGray, COLOR_BGR2GRAY);
        GaussianBlur(imgGray, imgBlur, Size(5, 5), 0);
        resize(imgRe, imgBlur, Size(64, 64), 0, 0, cv::INTER_AREA);
        img_lst.push_back(imgRe);
    }
}
```

FIGURA 5-3 FUNCIÓN LOAD\_IMAGES

A continuación, con la función computeHOGs, se extraen las características de las imágenes tanto positivas como negativas.

Las características que se definen son:

- |               |                 |
|---------------|-----------------|
| - winSize     | - nbins         |
| - blockSize   | - derivAperture |
| - cellSize    | - winSigma      |
| - blockStride |                 |

De ellas, las más importantes son:

- BlockSize define el tamaño de las imágenes que se le proporcionan, normalmente se suelen usar de 64x64.
- Cellsize es el tamaño de la celda o sobre cuantos pixeles se calcula el histograma. Mientras que blockstride es el paso para deslizar un bloque sobre la imagen. Cell y blockStride deben ser potencias de 2, y el residuo de blocksize entre cellsize debe ser 0.
- Nbins es el número de orientaciones o en cuántos intervalos se puede partir una celda, a mayor número mejor precisión.

Usando la implementación de HOGImage (Zhou, 2019) nos permite ver las características extraídas por HOG. Varios ejemplos se observan en las figuras de la 5-6 a 5-11, variando parámetros de HOG, como incluyendo filtros.

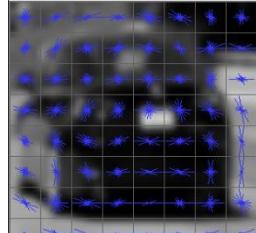


FIGURA 5-6 CON BLOCKSIZE (64,64), CELLSIZE Y BLOCKSTRIDE (8,8), CON SOLO 8X8 CELDAS NO ES SUFICIENTE PARA EXTRAER LAS CARACTERÍSTICAS DE LA IMAGEN, POR LO QUE EL RESULTADO ES MUY POBRE.

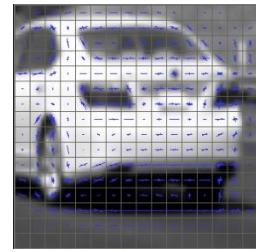


FIGURA 5-7 CON BLOCKSIZE (64,64), CELLSIZE Y BLOCKSTRIDE DE (4,4).

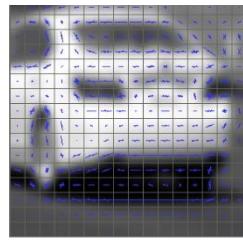


FIGURA 5-8 CON FILTRO GAUSSIANO (5,5)

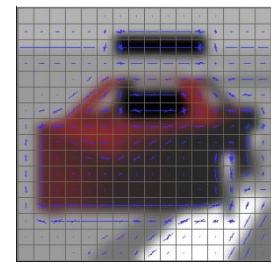


FIGURA 5-9 IMAGEN DEL VEHÍCULO DEL SIMULADOR

Ejemplo de imágenes negativas

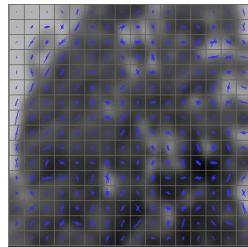


FIGURA 5-10 HOG FEATURES DE UN ÁRBOL

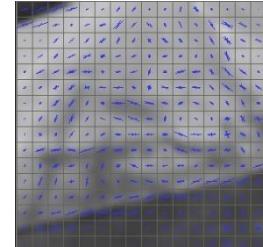


FIGURA 5-4 HOG FEATURES DE LA CARRETERA

Aumentar el número de características tiene un impacto muy alto en el tamaño de la máquina de soporte vectorial, junto al tiempo de procesamiento. Por ello se consideró suficiente con un tamaño de celdas y de paso de 4 pixeles, lo que equivale a dividir la imagen en 256 bloques.

```

void computeHOGs(const vector< Mat >& img_lst, vector< Mat >& gradient_lst)
{
    HOGDescriptor hog;
    hog.winSize = Size(64, 64);
    hog.blockSize = Size(16, 16);
    hog.blockStride = Size(4, 4);
    hog.cellSize = Size(4, 4);
    hog.nbins = 9;
    hog.derivAperture = 1;
    hog.winSigma = -1;
    //hog.histogramNormType = ;Usa L2Hys por defecto
    hog.L2HysThreshold = 0.2;
    hog.gammaCorrection = 1;
    hog.nlevels = 64;
    hog.signedGradient = false;

    vector< float > descriptors;
    for (size_t i = 0; i < img_lst.size(); i++)
    {
        hog.compute(img_lst[i], descriptors, Size(0, 0), Size(0, 0)); //winStride y padding
        gradient_lst.push_back(Mat(descriptors).clone());
    }
}

```

FIGURA 5-5 FUNCIÓN COMPUTEHOGS, EXTRAÉ LOS DESCRIPTORES DE TODAS LAS IMÁGENES

Para extraer HOG se usa la función computeHOGs de la figura 5-12, la instrucción principal es:

```
hog.compute(img_lst[i], descriptors, Size(0, 0), Size(0, 0));
```

Una vez extraídas todas las características y guardadas, se crea otro vector labels donde se guardan las etiquetas. +1 si son positivas, -1 si son negativas.

```
size_t positive_countC = gradient_lstC.size();

labelsC.assign(positive_countC, +1)
```

Con el cambio de la versión a OpenCV 4, Se requiere convertir los datos extraídos por HOG a una matriz Mat de nx1, para eso se utiliza la función de la figura 5-13:

```
convert_to_ml(gradient_lstC, train_dataC);
```

```

void convert_to_ml(const vector< Mat >& train_samples, Mat& trainData)
{
    const int rows = (int)train_samples.size();
    const int cols = (int)std::max(train_samples[0].cols, train_samples[0].rows);
    Mat tmp(1, cols, CV_32FC1);
    trainData = Mat(rows, cols, CV_32FC1);
    for (size_t i = 0; i < train_samples.size(); ++i)
    {
        CV_Assert(train_samples[i].cols == 1 || train_samples[i].rows == 1);
        if (train_samples[i].cols == 1)
        {
            transpose(train_samples[i], tmp);
            tmp.copyTo(trainData.row((int)i));
        }
        else if (train_samples[i].rows == 1)
        {
            train_samples[i].copyTo(trainData.row((int)i));
        }
    }
}

```

FIGURA 5-6 CÓDIGO DE LA FUNCIÓN CONVERT\_TO\_ML, MISMO PROPUESTO EN LOS EJEMPLOS DE OPENCV

Además, con OpenCV2.4 se agrega la siguiente línea para las etiquetas:

```

Mat labels2 = Mat(1, labelsC.size(), CV_32SC1,
                  labelsC.data()).clone();

```

Una vez extraídas las muestras, lo siguiente es pasar a entrenar la máquina de soporte vectorial, figura 5-14:

```

Ptr< SVM > svmC = SVM::create();
Ptr< SVM > svml = SVM::create();

svmC->setType(SVM::C_SVC);
svmC->setKernel(SVM::RBF);
svmC->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, (int)1e8, 1e-6));
svml->setType(SVM::C_SVC);
svml->setKernel(SVM::RBF);
svml->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, (int)1e8, 1e-6));

cv::Mat1f weights(1, 2);
weights(0, 0) = 0.2;
weights(0, 1) = 0.8;

ParamGrid CvParamGrid_C(pow(2.0, -2), pow(2.0, 5), pow(2.0, 1));
ParamGrid CvParamGrid_gamma(pow(2.0, -5), pow(2.0, 3), pow(2.0, 1));
ParamGrid CvParamGrid_p(0.1, 1.0, 0);
ParamGrid CvParamGrid_nu(pow(2.0, -2), pow(2.0, 5), 0);
ParamGrid CvParamGrid_coeff(pow(2.0, -2), pow(2.0, 5), 0);
ParamGrid CvParamGrid_deg(pow(2.0, -2), pow(2.0, 5), 0);

svmC->setClassWeights(weights);
svml->setClassWeights(weights);

svmC->trainAuto(ml::TrainData::create(train_dataC, ml::ROW_SAMPLE, labelsC), 10, svmC->getDefaultValueGrid(SVM::C),
    svmC->getDefaultValueGrid(SVM::GAMMA), CvParamGrid_p, CvParamGrid_nu, CvParamGrid_coeff, CvParamGrid_deg, false);

svml->trainAuto(ml::TrainData::create(train_dataL, ml::ROW_SAMPLE, labelsL), 10, svml->getDefaultValueGrid(SVM::C),
    svml->getDefaultValueGrid(SVM::GAMMA), CvParamGrid_p, CvParamGrid_nu, CvParamGrid_coeff, CvParamGrid_deg, false);

```

FIGURA 5-7 ENTRENAMIENTO DE SVM CON TRAINAUTO

Para crear la máquina se utiliza:

```
Ptr< SVM > svmC = SVM::create();
```

En OpenCV2.4

```
CvSVM *svm = new CvSVM;
```

Hay dos formas de entrenar la máquina. Con train dándole los parámetros o trainAuto para buscar los mejores parámetros a partir de una malla dada. Como trainAuto es lento, es mejor reducir la búsqueda limitando el rango y definiendo qué parámetros debe optimizar.

```

svmC->setType (SVM::C_SVC) ;
svmC->setKernel (SVM::RBF) ;

```

En primer lugar, se define el kernel y tipo, de los cuales se lograron mejores resultados con RBF y C\_SVC. A continuación, se definen los grids o valores sobre los que buscar, OpenCV ya tiene default grid. Para el tipo RBF + C\_SVC únicamente nos interesan optimizar los parámetros de gamma y c, para los que se usaron los valores por defecto. Para que no optimice los demás parámetros se utiliza:

```
ParamGrid CvParamGrid_coeff(pow(2.0, -2), pow(2.0, 5), 0);
```

Se crea una malla en la que se da el primer valor, el valor máximo y el incremento, al darle 0 de incremento no lo optimizará.

Se define también el termCriteria que indica cuando parar.

```
svmC->setTermCriteria(TermCriteria::MAX_ITER,
(int)1e8, 1e-6));
```

Opcionalmente se pueden poner class weights, los cuales son auxiliares a C para penalizar el error como se observa en la figura 5-15.

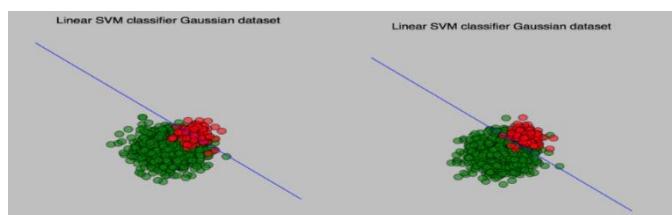


FIGURA 5-8 SE OBSERVA EL IMPACTO DE ESTABLECER PESOS PARA EL VALOR DE C, A LA IZQUIERDA LA CLASIFICACIÓN NORMAL, A LA DERECHA CON UN PESO DE 0.9 SOBRE LA CLASE DE VERDE, FUENTE: (“SVM BIAS ON WEIGHTS OF POSITIVES AND NEGATIVES - OPENCV Q&A FORUM” S. F.)

```

cv::::Mat1f weights(1, 2);

weights(0, 0) = 0.2; //para la clase de -1
weights(0, 1) = 0.8; //para la clase de 1

```

En el ejemplo, se puede interpretar como que, para la solución, en la ubicación del corte tenga a lo más un 20% falsos positivos y un 80% verdaderos positivos.

Entonces la función para entrenar la máquina queda como:

```

svmL->trainAuto(ml::TrainData::create(train_dataL,
                                         ml::ROW_SAMPLE, labelsL), 10, svmL->
                                         getDefaultGrid(SVM::C), svmL->
                                         getDefaultGrid(SVM::GAMMA), CvParamGrid_p,
                                         CvParamGrid_nu, CvParamGrid_coeff, CvParamGrid_deg,
                                         false);

```

En primer lugar, los parámetros se optimizaron con `trainAuto`, mismo que se usaron al incrementar el tamaño de los *datasets* y para el entrenamiento en OpenCV 2.4 (figura 5-16).

Sin embargo, puede que haya otros óptimos.

```

svmL->train(ml::TrainData::create(train_dataL,
                                         ml::ROW_SAMPLE, labelsL));

```

```

CvSVM* svmC = new CvSVM;
CvSVM* svmL = new CvSVM;

CvSVMParams params;
params.svm_type = CvSVM::C_SVC;
params.kernel_type = CvSVM::RBF;
params.C = 62.5;
params.gamma = 0.00225;
params.term_crit = cvTermCriteria(CV_TERMCRIT_ITER, (int)1e8, 1e-6);

CvSVMParams params2;
params2.svm_type = CvSVM::C_SVC;
params2.kernel_type = CvSVM::RBF;
params2.C = 12.5;
params2.gamma = 0.00225;
params2.term_crit = cvTermCriteria(CV_TERMCRIT_ITER, (int)1e8, 1e-6);

ParamGrid CvParamGrid_C(pow(2.0, -2), pow(2.0, 5), pow(2.0, 1));
ParamGrid CvParamGrid_gamma(pow(2.0, -5), pow(2.0, 3), pow(2.0, 1));
ParamGrid CvParamGrid_p(0.1, 1.0, 0);
ParamGrid CvParamGrid_nu(pow(2.0, -2), pow(2.0, 5), 0);
ParamGrid CvParamGrid_coeff(pow(2.0, -2), pow(2.0, 5), 0);
ParamGrid CvParamGrid_deg(pow(2.0, -2), pow(2.0, 5), 0);

Mat labels2 = Mat(1, labelsC.size(), CV_32SC1, labelsC.data()).clone();
svmC->train(train_dataC, labels2, Mat(), Mat(), params);

labels2 = Mat(1, labelsL.size(), CV_32SC1, labelsL.data()).clone();
svmL->train(train_dataL, labels2, Mat(), Mat(), params2);

```

FIGURA 5-9 IMAGEN DE LA PARTE DE ENTRENAMIENTO EN OPENCV 2.4 USANDO TRAIN

Para los vehículos de lado, los valores obtenidos fueron de  $2.25 \times 10^{-3}$  para gamma y 12.5 para C. Para los vehículos desde atrás, tuvo la misma gamma, pero 62.5 en C.

En el caso de OpenCV2.4 la máquina se guarda en formato .xml, mientras que OpenCV4 en .yml. Se mencionan ambas, porque inicialmente se realizó sobre la versión 4 y es la que se usó para el simulador, posteriormente se tradujo a la versión 2.4 para el vehículo.

### 5.3 Detección y nodo para el vehículo

El primer paso en ambos casos es cargar los descriptores desde la función de main:

SVM:

```
Ptr<SVM> svm;  
  
svm = StatModel::load<SVM>(svmFile);
```

Cascada:

```
CascadeClassifier carC;  
  
carC.load(obj_det_filename);
```

En 2.4, para la máquina de soporte vectorial se cambia a:

```
CvSVM *svm = new CvSVM;  
  
svm->load(svmFile.c_str());
```

Después se preparan vectores que contendrán todos los objetos detectados siempre que no rebasen un determinado número de fotogramas sin aparecer. En el caso del vehículo se usaron 15 fotogramas, que, usando una frecuencia de 5 Hz, corresponde a 3 segundos. Se trata de un vector de rectángulos que guardan la posición, el último fotograma en que apareció y de filtros de Kalman. Con ello nos permite seguir múltiples objetos sin que tengan que aparecer en el fotograma actual.

Una vez todo inicializado, dentro de un ciclo se recibe la imagen. En las pruebas iniciales se leyeron imágenes extraídas de un video, pero para el vehículo es necesario un paso extra para extraer la imagen de la cámara:

Al inicio del main, figura 5-17 se define:

```
ros::NodeHandle nh("~");  
  
ros::Subscriber camara_sub =
```

```
nh.subscribe("/app/camera/rgb/image_raw", 1,  
camaraRGBCallback);
```

Se suscribe al tópico de la cámara que nos manda un mensaje tipo sensor\_msgs::Image

Esta función de callback se entra cada que se complete un ciclo (while(nh.ok())), lee el tópico que se publica y lo guarda en una variable local. La imagen debe convertirse al formato Mat con ayuda de cv\_bridge:

```
cv_bridge::CvImagePtr cv_ptr;  
  
cv_ptr = cv_bridge::toCvCopy(msg,  
sensor_msgs::image_encodings::MONO8);  
  
img0 = cv_ptr->image.clone();
```

Con MONO8 ya recibimos la imagen en escala de grises

La imagen es de 640x480, pero podemos simplificar el problema tomando una ROI. Para este caso se recortó la altura, tomando ahora una imagen de 640x400, que corresponde a quitar una porción de la parte delantera del auto que se alcanza a ver en la cámara.

Igualmente se define una frecuencia para el ciclo con:

```

    ros::Rate loop_rate(RATE_HZ);

int main(int argc, char** argv)
{
    ros::init(argc, argv, "detecc_vehiculos");
    ros::NodeHandle nh("~");
    ros::Subscriber camara_sub = nh.subscribe("/app/camera/rgb/image_raw", 10, camaraRGBCallback);

    ros::Time current_time, last_time;
    current_time = ros::Time::now();
    last_time = ros::Time::now();

    ros::Rate loop_rate(RATE_HZ);

    String cascadaFile = "C:/Users/ifons/source/repos/cascada/cascada/data/cascade.xml";
    String svmfile = "C:/Users/ifons/source/repos/kalman/kalman/my_svML3C.yml"; //my_svML3C.yml
    String svmFile2 = "C:/Users/ifons/source/repos/kalman/kalman/my_svML4L.yml"; //my_svML3L.yml

    cout << "Iniciando detector..." << endl;

    std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();

    //Se carga la máquina de soporte vectorial
    Ptr<SVM> svm;
    svm = StatModel::load<SVM>(svmFile);
    Ptr<SVM> svm2;
    svm2 = StatModel::load<SVM>(svmFile2);

    std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();

    std::cout << "\nCargar SVM Time: " << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count() << "[ms]" << std::endl;

    begin = std::chrono::steady_clock::now();

    //Se carga el filtro de cascada
    CascadeClassifier carC;
    carC.load(cascadaFile);
    end = std::chrono::steady_clock::now();

    std::cout << "\nCargar cascada: " << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count() << "[ms]" << std::endl;

    //Vectores para guardar los objetos detectados, cuando fue el último frame en que aparecio, y sus filtros de kalman asociados
    vector<Rect> objetos;
    vector<int> ultimoFrame;
    vector<KalmanFilter> filtrosK;
    cout << "\n svm cargada";

    int cont = 5;
    int i=0;
    ...

    ros::Duration(1).sleep();

    while(nh.ok()){
        ros::spinOnce();
        current_time = ros::Time::now();
    }
}

```

FIGURA 5-10 IMAGEN DE LA FUNCIÓN MAIN, LA CONFIGURACIÓN INICIAL, HASTA EL INICIO DEL CICLO

Dentro del ciclo se hace primero la detección y después el rastreo. En el caso de la detección puede hacerse tomando el fotograma completo o solo en regiones cercanas a las detecciones.

Es por ello tenemos dos funciones:

- Detección completa

```
detecVentana(svm, svm2, carC, img, i, &img2,  
&numberD, &detections);
```

Función que cada 5 fotogramas realiza la detección completa del fotograma. Dentro, como no se usa la detección multiescala de HOG, si no que solo la predicción SVM, se deben cargar los mismos parámetros para extraer las características con que fue entrenada originalmente:

```
HOGDescriptor hog;  
  
hog.winSize = Size(64, 64);  
  
hog.blockSize = Size(16, 16);  
  
hog.cellSize = Size(4, 4);  
  
hog.blockStride = Size(4, 4);  
  
hog.nbins = 9;  
  
hog.derivAperture = 1;  
  
hog.winSigma = 4;  
  
//hog.histogramNormType = 0;  
  
hog.L2HysThreshold = 2.000000000000001e-01;  
  
hog.gammaCorrection = 1;  
  
hog.nlevels = 64;
```

El primer paso es usar el filtro de cascada para darnos una lista de candidatos en los que buscar. Usa el método de ventana deslizante y es bastante rápido, pero con muchos falsos positivos:

```
carC.detectMultiScale(imgGRAY, detectionsCascada, 1.1, 2, 0,  
Size(40, 40), Size(200, 200));
```

detections2 regresa una lista de rectángulos con las detecciones,

- 1.1 es el factor por el que se va incrementando la ventana
- 2 son el número de coincidencias que deben aparecer
- Size(40,40) tamaño inicial de la ventana
- Size(200,200) tamaño máximo

Posteriormente se recorre cada candidato, tratando de tomar una pequeña área más grande del 20%.

```
int width = detectionsCascada[j].width * 0.2;  
int height = detectionsCascada[j].height * 0.2;
```

Se cambia de tamaño a 64x64 (mismo de HOG) y se extraen sus características:

```
resize(imgEsquina, imgSVM, Size(64, 64), 0, 0, cv::INTER_AREA);  
vector< float > descriptorsSVM;  
imgSVM.convertTo(imgSVMF, CV_8UC3);  
hog.compute(imgSVMF, descriptorsSVM, Size(0, 0), Size(0, 0));
```

En compute el primer size(0,0) significan que se toma la imagen completa para hacer la predicción, el segundo corresponde al padding.

Con OpenCV2.4 se agrega para usar los descriptores:

```
Mat fm = Mat(1, descriptorsSVM.size(), CV_32FC1,  
descriptorsSVM.data()).clone();
```

Para la predicción, OpenCV normalmente devuelve solo la clase a la que pertenece, según del lado de la frontera donde se encuentra. Para recibir la distancia se usa:

```
float result1 = svm->CvSVM::predict(const CvMat* sample, bool  
returnDFVal=true)
```

Algunas de las dificultades observadas pueden apreciarse en la figura 5-18, donde se evidencia que, al intentar detectar vehículos que van apareciendo en las esquinas, ninguno de los dos detectores logra identificarlos hasta que el vehículo esté completamente visible en la imagen.

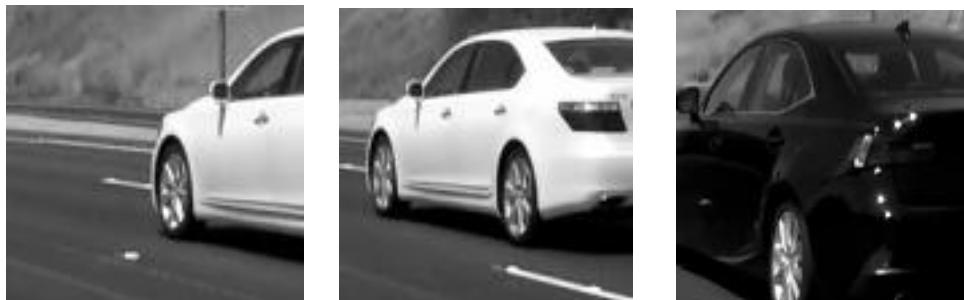


FIGURA 5-11 EJEMPLOS DE FALSOS NEGATIVOS

Hecha la predicción, si el resultado es positivo (1), se guarda dos veces para poder usar groupRectangles. La función toma el promedio de los rectángulos sobrepuertos, pero solo los conserva si hay más de uno.

```
if (result > 0.0) {  
  
    detectionsFinal.push_back(detectionsCascada[j]);  
  
    detectionsFinal.push_back(detectionsCascada[j]);  
  
}  
  
groupRectangles(detectionsFinal, 1, 0.6);
```

Los rectángulos detectados se dibujan sobre la imagen y esta se publica con el tópico /detec, agregando al inicio:

```
ros::NodeHandle nh("~");  
  
detec_publisher =  
  
nh.advertise<sensor_msgs::Image>("/detec", 1);
```

Al final de la detección se publica:

```
sensor_msgs::Image img_msg;  
  
std_msgs::Header header;  
  
header.stamp = ros::Time::now();  
  
img_bridge = cv_bridge::CvImage(header,  
  
sensor_msgs::image_encodings::TYPE_8UC1, img1);  
  
img_bridge.toImageMsg(img_msg);  
  
detec_publisher.publish(img_msg);
```

Se crea un nuevo mensaje imagen, cuyo formato incluye un encabezado, al que se le pone la hora. Con CV\_bridge se convierte la imagen a un formato de ROS, y se publica. Las detecciones se pueden visualizar en RVIZ.

- Detección sobre solo una región:

En este caso se llama a la función de la figura 5-19:

```
deteceRegion(carC, img, i, predRect.x, predRect.y,  
predRect.width, predRect.height, &img2, &encontro,  
&detec);
```

Solo entra cuando hay detecciones previas, busca en los lugares indicados por la predicción de Kalman, que se realiza desde el main:

```
Mat state(6, 1, CV_32F);  
  
state = kalmans[j].predict();  
  
Rect predRect;  
  
predRect.width = state.at<float>(4);  
predRect.height = state.at<float>(5);  
predRect.x = state.at<float>(0);  
predRect.y = state.at<float>(1);
```

En caso de no encontrarlo, busca en la posición anterior, en ambos casos se toma una ventana un 80% más grande. De forma que nuestro acercamiento es asumir que el mismo objeto debe estar cerca de donde fue detectado la primera vez. En este método únicamente se realiza la

detección por cascada, si hay coincidencias se usa group rectangle para dar el promedio de la nueva posición.

```

carC.detectMultiScale(img2, detections2, 1.1, 2, 0,
                      Size(width*0.4,height*0.4), Size(width1,height1));

imgGRAY(Rect(xl,yl,width1,height1)).copyTo(imgROI);

std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
carC.detectMultiScale(imgROI, detectionsCascada, 1.1, 2, 0, Size(width*0.4, height*0.4), Size(width1, height1)); //Dado tamano segun prediccion
std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();

vector<Rect> detectionDup;
detectionDup = detectionsCascada;
detectionsCascada.insert(detections2.end(), detectionDup.begin(), detectionDup.end());

cout << "\n number of detections: " << (int)detectionsCascada.size();
if ((int)detections2.size() > 0) {
    *encontro = true;

    groupRectangles(detectionsCascada, 1, 0.60);
    rectangle(imgGRAY, Rect(xl + detectionsCascada[0].x, yl + detectionsCascada[0].y, detectionsCascada[0].width, detectionsCascada[0].height), 200, imgGRAY.cols / 400 + 1);
    rectangle(imgGRAY, Rect(xl+detectionsCascada[0].x + detectionsCascada[0].width / 2-2, yl+detectionsCascada[0].y + detectionsCascada[0].height / 2-2, 4, 4), 250,
              | imgGRAY.cols / 400 + 1);

    *imgF = imgGRAY; ///
    detections = Rect(xl + detectionsCascada[0].x, yl + detectionsCascada[0].y, detectionsCascada[0].width, detectionsCascada[0].height);

    sensor_msgs::Image img_msg;
    std_msgs::Header header;
    header.stamp = ros::Time::now();
    img_bridge = cv_bridge::CvImage(header, sensor_msgs::image_encodings::TYPE_8UC1, img1);
    img_bridge.toImageMsg(img_msg);
    detec_publisher.publish(img_msg);
}
else {
    *imgF = imgGRAY;
}

```

FIGURA 5-12 FRAGMENTO DEL CÓDIGO PARA DETECTAR SOBRE UNA REGIÓN

## 5.4 Rastreo

El siguiente paso es el rastreo, para cada detección se revisa si es el mismo vehículo entre dos fotogramas, basándose en que se halle en una posición cercana a la anterior. Si se cumple, se hace la actualización, en caso contrario se debe crear un nuevo filtro de Kalman. El código completo se ve en la figura 5-25.

Para crear el filtro se cuenta con la función:

```
void preparacionKalman(Rect detections, KalmanFilter* kalman)
```

Función que recibe un rectángulo con la detección y regresa el filtro.

Para el rastreo se consideran 6 variables la posición y velocidad en x y y, el tamaño de la ventana detectada width y height. De las 6 variables, 4 son las que se miden, la posición y la ventana

```
int stateSize = 6; // [x, y, v_x, v_y, w, h]
int measSize = 4; // [z_x, z_y, z_w, z_h]
KalmanFilter kf(stateSize, measSize, contrSize);
//Matriz estados
Mat state(stateSize, 1, CV_32F);
//Matriz mediciones
Mat meas(measSize, 1, CV_32F);
```

Después de crear el filtro, se agrega la primera medición del objeto en los estados statePre y statePost.

```
kf.statePre.setTo(0);
kf.statePre.at<float>(0, 0) = detections.x;
kf.statePre.at<float>(1, 0) = detections.y;
kf.statePre.at<float>(2, 0) = 0;
kf.statePre.at<float>(3, 0) = 0;
kf.statePre.at<float>(4, 0) = detections.width;
kf.statePre.at<float>(5, 0) = detections.height;
```

```

kf.statePost.setTo(0);

kf.statePost.at<float>(0, 0) = detections.x;

kf.statePost.at<float>(1, 0) = detections.y;

kf.statePost.at<float>(2, 0) = 0;

kf.statePost.at<float>(3, 0) = 0;

kf.statePost.at<float>(4, 0) = detections.width;

kf.statePost.at<float>(5, 0) = detections.height;

```

La matriz de transición entonces queda de la siguiente forma:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Las ecuaciones son las siguientes:

$$x = x_0 + v_{x0} * \Delta t$$

$$y = y_0 + v_{y0} * \Delta t$$

La posición en x depende de la posición y velocidad en x en el paso anterior. No se está tomando en cuenta la aceleración, y se considera x, y independientes. Mientras que las ventanas solo dependen de su valor anterior, podría agregarse que dependieran de la posición.

```

kf.transitionMatrix = (Mat<float>(6, 6) << 1, 0, 1, 0, 0, 0,
0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1);

```

La matriz de medición queda como:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```
kf.measurementMatrix = cv::Mat::zeros(measSize, stateSize,  
CV_32F);  
  
kf.measurementMatrix.at<float>(0) = 1.0f;  
  
kf.measurementMatrix.at<float>(7) = 1.0f;  
  
kf.measurementMatrix.at<float>(16) = 1.0f;  
  
kf.measurementMatrix.at<float>(23) = 1.0f;
```

Por último, se define la covarianza y el ruido.

```
setIdentity(kf.processNoiseCov, Scalar::all(.00005));
```

Para processNoiseCov se encontró adecuado un valor de  $5*10e^{-5}$ . Con un valor más grande se tiene una respuesta más rápida del filtro, pero es más propenso a errores cuando no hay detecciones.

```
setIdentity(kf.measurementNoiseCov, Scalar(1e-1));  
  
setIdentity(kf.errorCovPost, Scalar::all(.1));
```

Podemos comparar como afecta modificar las covarianzas. En el caso de `measurementNoiseCov` se puede interpretar en qué tanto se confía en las mediciones y es un error que se propaga hacia atrás. Representa el ruido que puede haber en las mediciones o para nuestro caso el detector. En el ejemplo mostrado entre las figuras 5-20 y 5-22, los puntos azules deben seguir al verde y hacia el final no se obtienen nuevas mediciones, por lo que ya no se realiza la corrección.

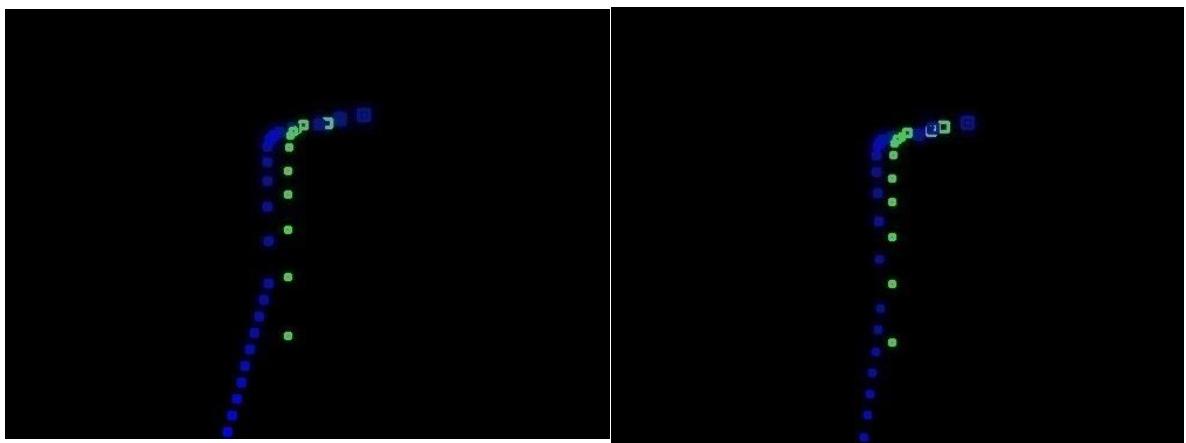


FIGURA 5-140  
MEASUREMENTNOISECOV DE 0.1

FIGURA 5-13  
MEASUREMENTNOISECOV DE 0.01

Como se ve con mayor ruido “`measurementNoiseCov`” sigue mucho peor la señal cuando deja de tener nuevas mediciones. Además, requiere de más valores para un mejor seguimiento. En el ejemplo deja de moverse en el eje x, pero con ruido alto la predicción preserva todavía movimiento en ese eje.

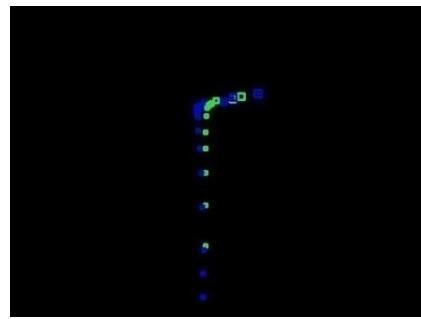


FIGURA 5-15 MEASUREMENTNOISECOV DE 0.001

En cambio, como se ve en las figuras 5-23 y 5-24, el efecto de processNoiseCov se observa en la corrección. Con un valor más grande permite una convergencia más rápida a la señal, pero acarrea más ruido que se hace presente cuando no se tienen nuevas mediciones.

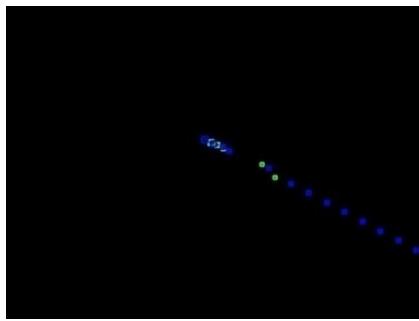


FIGURA 5-16 CON  
PROCESSNOISECOV DE  
0.00005, LOS PUNTOS  
AZULES REACCIONAN  
MÁS LENTO A LOS  
CAMBIOS

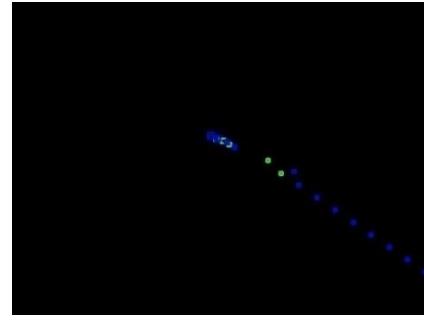


FIGURA 5-17 CON  
PROCESSNOISECOV DE 0.5,  
LOS PUNTOS AZULES  
REACCIONAN MÁS RÁPIDO  
Y COMO CONSECUENCIA  
SE DISPARAN

FIGURA 5-18 CÓDIGO COMPLETO PARA CREAR EL FILTRO DE KALMAN

## Actualización del filtro

Para la actualización, figura 5-26, se tiene la siguiente función que recibe el filtro y lo devuelve actualizado. Recibe la imagen, si hubo una detección en el fotograma actual y dicha posición, así mismo como los valores del reloj para obtener el intervalo de tiempo entre las dos predicciones.

```

kalman(KalmanFilter kf, Rect detections, Mat img,int i,bool
found, std::chrono::monotonic_clock::time_point lastP,
std::chrono::monotonic_clock::time_point Pactual,
KalmanFilter* act)

```

Igual que para la detección, se publicará la imagen con el rectángulo negro de la predicción y un mensaje con las detecciones.

```

ros::NodeHandle nh("~");
detec_publisherk =
nh.advertise<sensor_msgs::Image>("/deteck",1);

```

Al final de la predicción se publica:

```

sensor_msgs::Image img_msg;
std_msgs::Header header;
header.stamp = ros::Time::now();
img_bridge = cv_bridge::CvImage(header,
sensor_msgs::image_encodings::TYPE_8UC1, img);
img_bridge.toImageMsg(img_msg);
detec_publisherk.publish(img_msg);

```

Dentro se realiza primero la predicción para actualizar al siguiente estado y dibujar sobre la imagen. Además, se actualiza la matriz de transición modificando el valor de dt que corresponde el intervalo de tiempo transcurrido entre dos fotogramas.

```

kf.transitionMatrix = (Mat_<float>(6, 6) << 1, 0, dt, 0, 0, 0,
0, 1, 0, dt, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 1);

Mat meas(4, 1, CV_32F);
Mat state(6, 1, CV_32F);
state = kf.predict();

Rect predRect;
predRect.width = state.at<float>(4);
predRect.height = state.at<float>(5);
predRect.x = state.at<float>(0);
predRect.y = state.at<float>(1);

```

Si se encontró hace la corrección:

```

if (found) {
    meas.at<float>(0) = detections.x;
    meas.at<float>(1) = detections.y;
    meas.at<float>(2) = detections.width;
    meas.at<float>(3) = detections.height;
    kf.correct(meas);
}

```

Como se observa, la operación que realiza es: statePre = TransitionMatrix \* statePost

```

Mat meas(4, 1, CV_32F);
Mat state(6, 1, CV_32F);

state = kf.predict();

Rect predRect;
predRect.width = state.at<float>(4);
predRect.height = state.at<float>(5);
predRect.x = state.at<float>(0);
predRect.y = state.at<float>(1);

if (found) {
    meas.at<float>(0) = detections.x;
    meas.at<float>(1) = detections.y;
    meas.at<float>(2) = detections.width;
    meas.at<float>(3) = detections.height;

    kf.correct(meas);
}

*act = kf;

```

FIGURA 5-19 CÓDIGO PARA HACER LA PREDICCIÓN Y CORRECCIÓN

## **6 RESULTADOS EN LA DETECCIÓN Y RASTREO**

Tras el entrenamiento del filtro de cascada y de las máquinas de soporte vectorial, ambos detectores fueron evaluados con un conjunto de prueba de 1600 muestras. En el filtro de cascada, los conjuntos de entrenamiento fueron de 7000 muestras positivas y 8000 negativas. Por su parte, para las SVM se utilizó inicialmente un conjunto de entrenamiento más reducido con el fin de realizar una búsqueda exhaustiva de los mejores parámetros dentro de una malla y experimentar con los hiperparámetros. Este conjunto inicial estuvo compuesto por 400 muestras de vehículos capturados desde la parte trasera, 400 muestras de vehículos laterales, y 600 muestras negativas. Posteriormente, con los datos encontrados, se incrementó el tamaño del conjunto de entrenamiento a 1500 imágenes positivas cada tipo y 2000 negativas, con los que se entrenaron nuevamente las máquinas. Este aumento en el tamaño del conjunto de entrenamiento mejoró los resultados, aunque a costa de un incremento significativo tanto en el tiempo de entrenamiento como en el tiempo necesario para cargar los detectores. Asimismo, se observó un ligero aumento en el tiempo de detección. Aunque se realizaron pruebas con conjuntos de datos de entrenamiento aún más grandes para las SVM, se decidió detener el proceso en este punto. El aumento en el tamaño del conjunto llevaba a un crecimiento en el tamaño de los modelos resultantes que, junto con la limitada capacidad de memoria RAM disponible en el carro físico, impedía cargar completamente el modelo, lo que ocasionaba un incremento considerable en los tiempos de detección.

Con las máquinas entrenadas, se refinaron los resultados ajustando la distancia al margen, utilizando como referencia las curvas PRC y ROC. A continuación, se presentan los resultados obtenidos al modificar la frontera de decisión. Es importante destacar que el conjunto de evaluación de imágenes positivas está compuesto por vehículos en buenas a moderadas condiciones de iluminación (es decir, con la presencia de algunas sombras) e incluye tanto vehículos de lado como capturados desde la parte trasera. Los resultados se verían considerablemente afectados en entornos nocturnos o cuando el vehículo está parcialmente oculto.

La tabla 6-1, muestra los resultados de la máquina de soporte vectorial para la detección de vehículos desde la parte trasera. Con un umbral de 0, se obtiene el resultado predeterminado al calcular la frontera de decisión. Desplazar esta frontera hacia la izquierda permite aumentar el número de verdaderos positivos, mientras que moverla hacia la derecha reduce los falsos positivos, en un comportamiento similar al observado en la Figura 5-15. Las curvas ROC y PRC se presentan en la figura 6-1, donde se ha marcado en rojo el punto de corte seleccionado, que corresponde a un margen de -0.8. Esta elección se hizo con el objetivo de maximizar la tasa de verdaderos positivos antes de que el incremento en la tasa de falsos positivos se vuelva significativo, como ocurre con un margen de -1.

TABLA 6-1 RESULTADOS DE MOVER EL UMBRAL EN LA PRIMERA SVM

Threshold	% TP / Recall	% TN	% FN	% FP	Precision
-2.0	94.942	48.428	5.058	51.572	65.507
-1.8	94.856	59.774	5.144	40.226	70.868
-1.6	94.726	70.420	5.274	29.580	76.765
-1.4	94.481	79.963	5.519	20.037	82.950
-1.2	93.891	87.369	6.109	12.631	88.467
-1.0	93.314	94.231	6.686	5.769	94.349
<b>-0.8</b>	<b>92.666</b>	<b>98.016</b>	<b>7.334</b>	<b>1.984</b>	<b>97.968</b>
-0.6	91.557	98.731	8.443	1.269	98.675
-0.4	90.404	98.886	9.596	1.114	98.821
-0.2	88.906	98.956	11.094	1.044	98.877
0.0	86.874	98.995	13.126	1.005	98.894
0.2	84.699	99.034	15.301	0.966	98.910
0.4	81.875	99.096	18.125	0.904	98.946
0.6	78.403	99.197	21.597	0.803	99.023
0.8	73.692	99.244	26.308	0.756	99.023
1.0	67.021	99.337	32.979	0.663	99.062

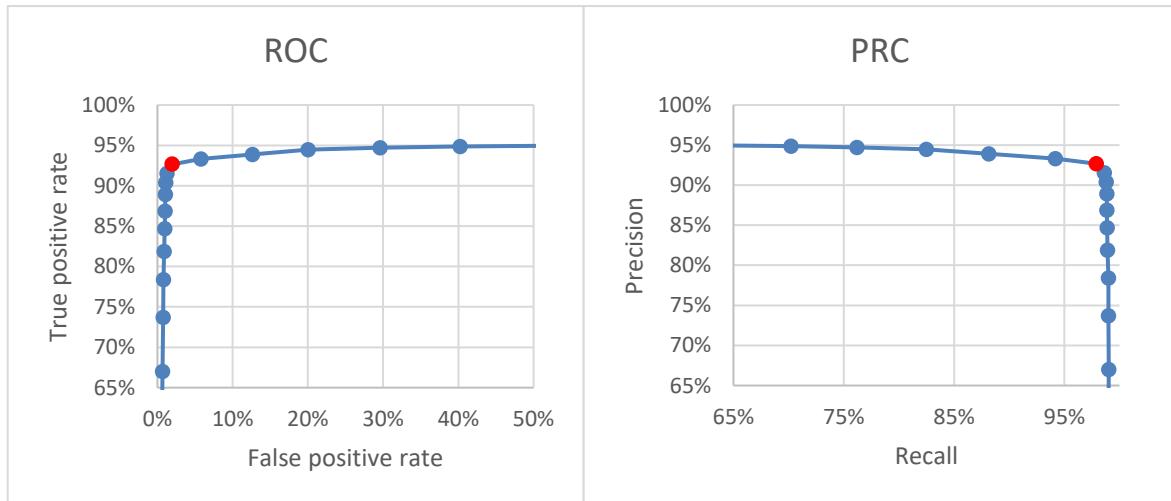


FIGURA 6-1 CURVAS ROC Y PRC DE LA PRIMERA SVM, EL PUNTO ROJO ES EL UMBRAL ELEGIDO

La tabla 6-2 y la figura 6-2 muestran los resultados para la segunda máquina de soporte vectorial, entrenada para detectar vehículos de lado. El margen se eligió también -0.8, recordando que en el set de prueba las imágenes de vehículos vistos desde la parte trasera y de lado están combinados. Igualmente, las SVM realizan un paso de refinamiento, el objetivo es que filtren los falsos positivos arrojados por el filtro de cascada.

TABLA 6-2 RESULTADOS DE LA SEGUNDA SVM

Threshold	% TP / Recall	% TN	% FN	% FP	Precision
-2.0	99.389	47.894	0.611	52.106	65.605
-1.8	98.180	58.199	1.820	41.801	70.138
-1.6	96.659	68.495	3.341	31.505	75.419
-1.4	93.915	78.101	6.085	21.899	81.091
-1.2	89.251	86.571	10.749	13.429	86.922
-1.0	84.691	93.767	15.309	6.233	93.145
<b>-0.8</b>	<b>77.571</b>	<b>97.311</b>	<b>22.429</b>	<b>2.689</b>	<b>96.650</b>
-0.6	69.832	98.749	30.168	1.251	98.240
-0.4	61.877	99.277	38.123	0.723	98.845
-0.2	55.982	99.619	44.018	0.381	99.324
0.0	51.277	99.775	48.723	0.225	99.562
0.2	47.665	99.860	52.335	0.140	99.707
0.4	44.700	99.899	55.300	0.101	99.774
0.6	42.090	99.938	57.910	0.062	99.853
0.8	39.637	99.992	60.363	0.008	99.980
1.0	36.011	100.000	63.989	0.000	100.000

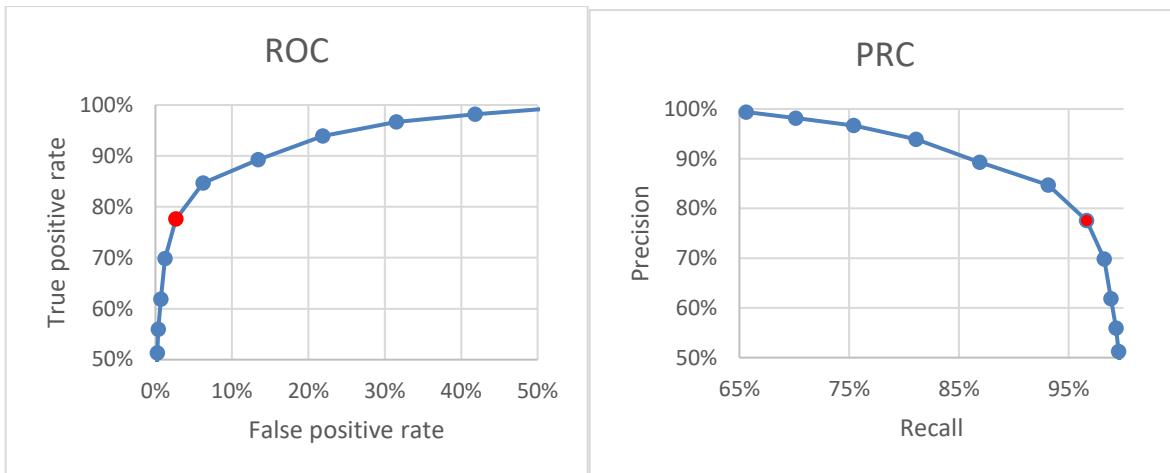


FIGURA 6-2 CURVAS ROC Y PRC DE LA SEGUNDA SVM

En las tablas 6-3 y 6-4 se presentan las tasas de matrices de confusión, junto las métricas de precisión y exactitud en los puntos elegidos. La tabla 6-5 muestra los resultados de juntar ambas máquinas de estado trabajando en conjunto, realizan una operación OR, es decir, basta con que alguna de las dos máquinas lo detecte como vehículo, para tomarlo como positivo, claro que esto, incrementa también el número de falsos positivos.

TABLA 6-3 TASAS DE DESEMPEÑO DE SVM COMPLETA CON LA PARTE TRASERA DE VEHÍCULOS

%TN Especificidad	98.02	%FN	7.33
%FP	1.98	%TP Sensibilidad	92.66

Precisión:  $TP/(TP+FP)$ : 97.904%

Exactitud:  $(TP+TN)/(TP+TN+FP+FN)$ : 95.34%

Máquina 2 (vista de lado de los vehículos):

TABLA 6-4 TASAS DE DESEMPEÑO DE SVM COMPLETA CON VEHÍCULOS DE LADO

%TN Especificidad	97.31	%FN Sensibilidad	22.43
%FP	2.69	%TP	77.57

Precisión: 96.65 %

Exactitud: 87.44%

Ambas máquinas:

TABLA 6-5 TASAS DE DESEMPEÑO RESULTANTE DE JUNTAR AMBAS SVM COMPLETAS

%TN Especificidad	96.84	%FN Sensibilidad	1.55
%FP	3.15	%TP	98.44

Precisión: 96.89%

Exactitud: 97.64%

De la combinación de ambos detectores logra una sensibilidad del 98.44% de los vehículos correctamente. Permitiendo hasta un 3.15% de falsos positivos. Para mejorarlo, se puede continuar con la idea de separar en múltiples detectores que consideren diferentes posiciones del vehículo, incluido cuando aparece incompleto.

De los resultados, se observan las mejoras significativas de incrementar el tamaño del set de entrenamiento, como de usar varios detectores según la posición del vehículo. Aun así, en las pruebas en un escenario real los resultados empeoran, siendo el mayor problema la oclusión ambiental. Debido a ella, cuando se presentan pequeños cambios en la iluminación o sombras, una detección puede no aparecer en el siguiente fotograma. A esto se le suma que la detección no siempre es igual. El área que detecta como vehículo o el centro puede variar ligeramente entre fotogramas, lo que entorpece principalmente al filtro de Kalman.

En el caso del filtro de cascada, este componente representa una de las partes más sensibles del proceso de detección. Entrenado con un conjunto de 7000 imágenes positivas y 8000 negativas, su desempeño en la detección de verdaderos positivos es limitado. Para mejorar un poco los resultados, se ajusta el parámetro minNeighbors para definir cuántas detecciones superpuestas se requieren, como se observa en la tabla 6-6 y la figura 6-3. Dado que en este escenario se empleará una máquina de soporte vectorial para filtrar los resultados, se optó por el valor que maximiza la tasa de verdaderos positivos (TP), como se muestra en la tabla 6-7.

TABLA 6-6 RESULTADOS DEL FILTRO DE CASCADA

minNeighbors	% TP / Recall	% TN	% FN	% FP	Precision
0	90.007	92.143	9.993	7.857	91.972
1	80.000	98.026	20.000	1.974	97.592
2	74.200	98.998	25.800	1.002	98.667
3	69.700	99.347	30.300	0.653	99.072
4	67.019	99.580	32.981	0.420	99.378

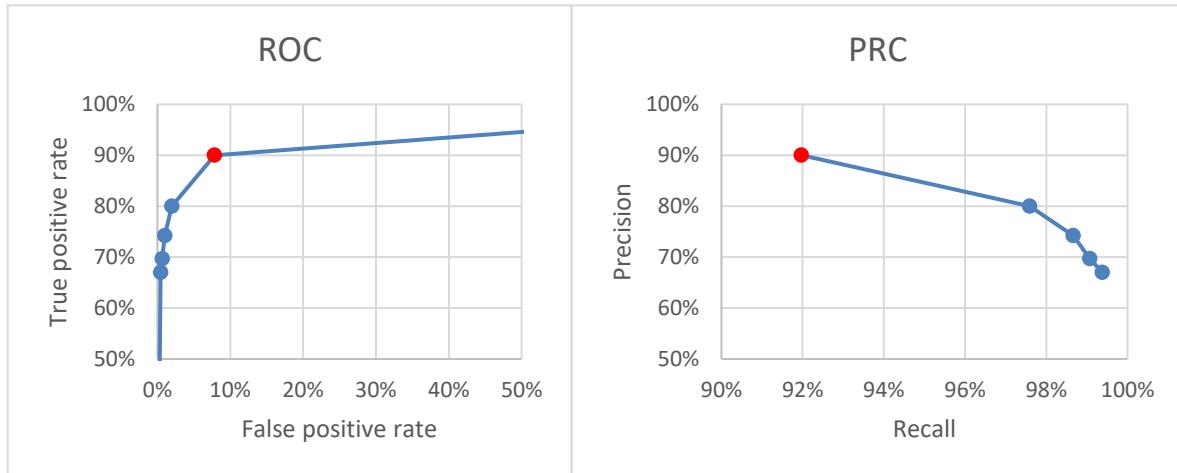


FIGURA 6-3 CURVAS ROC Y PRC DEL FILTRO DE CASCADA

TABLA 6-7 TASAS DE DESEMPEÑO DEL FILTRO DE CASCADA

%TN Especificidad	92.14 %	%FN	9.99 %
%FP	7.86 %	%TP Especificidad	90.01 %

Precisión: 91.97 %

Exactitud: 91.07 %

Por lo tanto, para obtener el resultado final de la detección se juntan los detectores. Primeramente, el filtro de cascada realiza un barrido sobre la imagen para obtener candidatos, y sobre ellos se realiza la predicción con las máquinas de soporte vectorial. Si alguna de las dos máquinas lo clasifica como positivo lo mantenemos como una detección válida. Como se muestra en la tabla 6-8, lleva a una ligera reducción de la tasa de verdaderos positivos obtenida por el filtro de cascada, a cambio de elevar la tasa de verdaderos negativos del 92.14% al 98.83%.

TABLA 6-8 TASAS DE DESEMPEÑO DE AMBOS DETECTORES

%TN Especificidad	98.83 %	%FN	10.2 %
%FP	1.16 %	%TP Especificidad	89.79 %

Precisión: 98.71%

Exactitud: 94.31%

En las figuras 6-4 y 6-5 se incluye la comparación de todos los detectores. Los puntos en solitario son el resultado de juntar los detectores, el punto azul claro corresponde a juntar ambas máquinas de soporte vectorial, y el punto rojo de agregar además el filtro de cascada. Por cuestiones de tiempo, no se probaron varias combinaciones, para estos solo se eligieron los mejores resultados de cada detector en solitario.

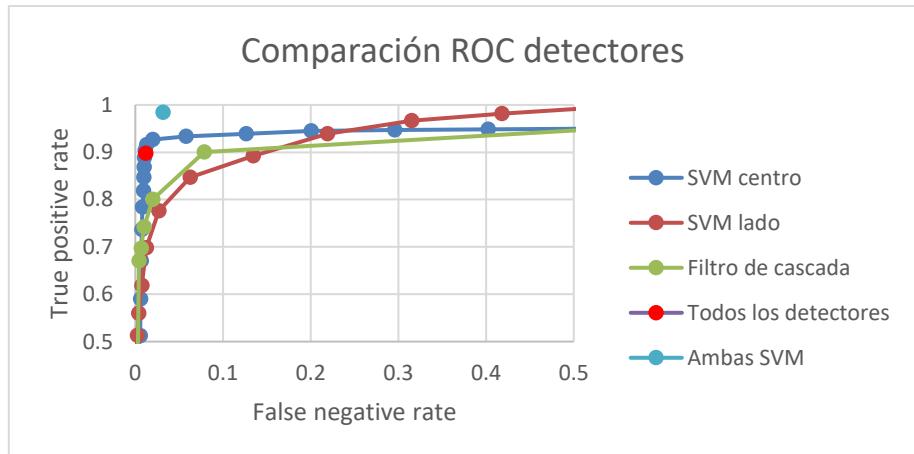


FIGURA 6-4 COMPARACIÓN DE LAS CURVAS ROC DE TODOS LOS DETECTORES

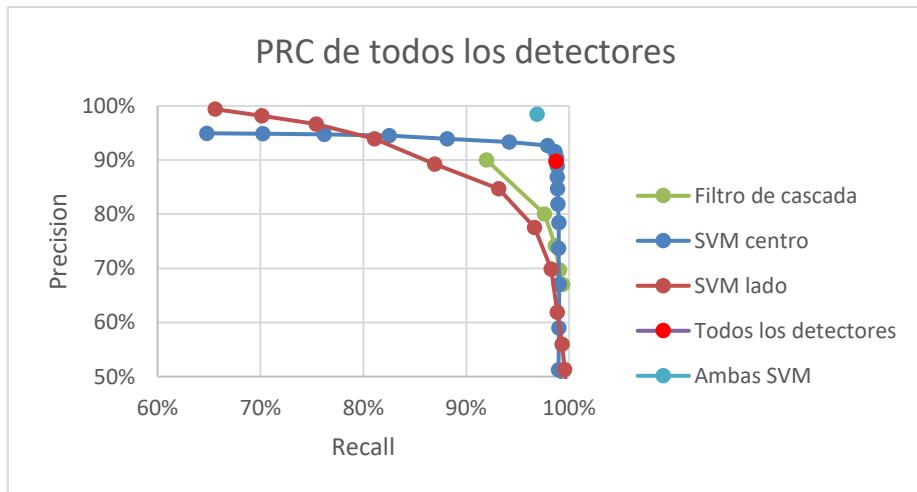


FIGURA 6-5 COMPARACIÓN DE LAS CURVAS PRC DE TODOS LOS DETECTORES

Al realizar prueba de un caso real con un video. El filtro de cascada arroja por fotograma entre 7 y 11 detecciones o candidatos para SVM, como se ven en la figura 6-6, pese a tener una tasa muy alta de falsos positivos, la idea es que encuentre la mayoría de los vehículos. En la figura 6-7 se observa el resultado de juntar las máquinas de soporte vectorial para filtrar los candidatos. El mayor problema es la presencia de falsos positivos. Pues en el caso de falsos negativos, se tiene la ventaja de que se sigue realizando el seguimiento entre fotogramas con el filtro de Kalman. Siempre y cuando se hayan tenido suficientes muestras para hacer el seguimiento de forma correcta. El siguiente paso, en la figura 6-8 se observa la predicción Kalman.



FIGURA 6-6 CANDIDATOS ARROJADOS POR EL FILTRO DE CASCADA



FIGURA 6-7 EJEMPLOS DE DETECCIONES SOBRE CUADROS, EN LA PRIMERA IMAGEN SE IDENTIFICAN CORRECTAMENTE LOS DOS VEHÍCULOS, EN EL SEGUNDO POR EL CAMBIO DE ILUMINACIÓN NO DETECTA UNO DE LOS VEHÍCULOS, EN LA TERCERA IMAGEN SE MUESTRA UN FALSO POSITIVO.



FIGURA 6-8 PREDICCIÓN DEL FILTRO DE KALMAN, LOS RECUADROS BLANCOS SON LAS DETECCIONES, Y LOS NEGROS LA PREDICCIÓN (EN ESA MISMA VENTANA DE LA PREDICCIÓN ES DONDE SE REALIZA LA DETECCIÓN PARCIAL PARA BUSCAR EL VEHÍCULO Y HACER LA CORRECCIÓN.

La solución presentada aún tiene margen para mejorarse, y es importante mencionar que requiere de modificaciones antes de trasladarse a otro problema. Según las especificaciones se requiere de optimizar algunos puntos, entre ellos:

- Tamaño mínimo y máximo para las ventanas en detección con cascada.  
Depende de la cámara, la distancia focal de esta, junto a su posición. Se

pueden tener distintos zooms según el ángulo de visión, con los que las imágenes pueden ser más grandes o pequeñas a una misma distancia.

- Tamaño de la región de interés, el problema se realizó tomando en cuenta que la cámara del vehículo es de 640x480 pixeles, y parte del área de la imagen es ocupada por la parte frontal del vehículo.
- El número máximo de fotogramas sin que aparezca una detección para seguir haciendo el seguimiento. Depende en parte de la potencia computacional del vehículo, con la cual se tendrá una tasa u otra para hacer el procesamiento de las imágenes. Por lo que entre fotogramas se pueden esperar mayores o menores cambios según la frecuencia.
- Mismo razonamiento, para el máximo movimiento esperado de un vehículo entre dos fotogramas, para poder asociar una detección como correspondiente al filtro de Kalman
- Según la tasa, se puede limitar el problema si se considera el entorno sobre el cual el vehículo se mueve, junto a su velocidad, se pueden ajustar mejor los parámetros del filtro, el processNoiseCov, measurementNoiseCov y errorCovPost.
- Igualmente se puede mejorar la precisión de los detectores. Limitando el problema a algún ambiente, o dividiendo la imagen y aplicar según la región distintos clasificadores.

Con el simulador se ejecuta uno de los mapas como curved\_road.launch (figura 6-9). El cual se ha modificado para que aparezcan dos vehículos. Por medio de 3 terminales se corre:

- roscore
- roslaunch autonomous\_gazebo\_simulation curved\_road.launch
- rosrun prueba detecRastreoSim

Para pruebas le damos velocidades a los vehículos, con los cuales se ajustaron los valores de la covarianza para el filtro de Kalman, algunos resultados se observan en la figura 6-10, en la figura 6-9 se observa el entorno de pruebas. Para el carro físico, las figuras 6-11 y 6-12 muestran algunos resultados.

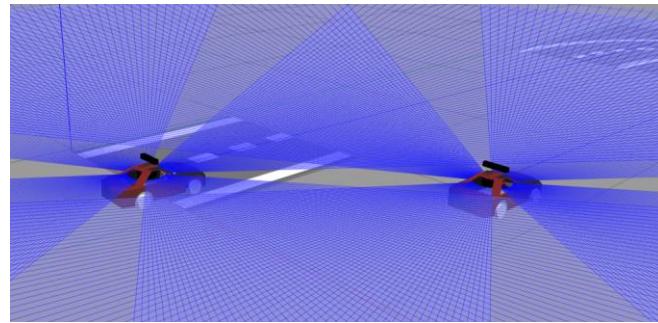


FIGURA 6-9 IMAGEN DEL SIMULADOR DESDE GAZEBO, MAPA CON DOS VEHÍCULOS

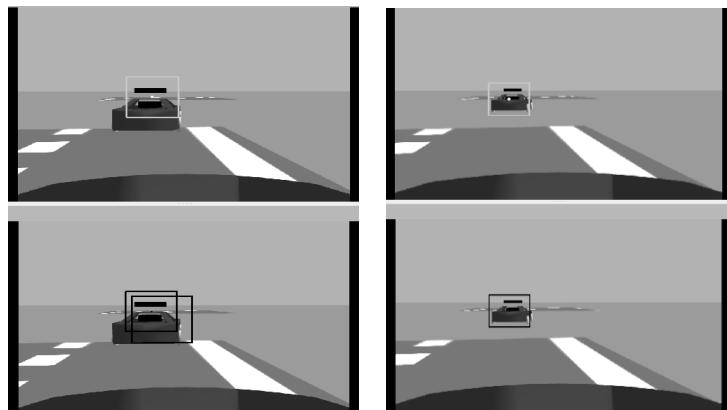


FIGURA 6-10 EJEMPLO DE DETECCIONES EN EL SIMULADOR, ARRIBA SE MUESTRA LA DETECCIÓN, ABAJO LA PREDICCIÓN

El procedimiento es similar para ejecutarlo en el vehículo, se ejecutan los comandos:

- rosrun manual\_control manual\_odroid.launch
- rosrun vision\_camara detecRastreoC

En la tabla 6-9 se presentan los tiempos promedios medidos tanto en el simulador de ROS, como el carro físico.

TABLA 6-9 TIEMPOS MEDIDOS DE LOS DETECTORES EN LOS AMBIENTES DE PRUEBAS

	En el simulador	En el vehículo
Tiempo para cargar las dos máquinas de soporte vectorial	10.156 segundos	63.829 segundos
Tiempo para la ventana deslizante	100-110 ms	380-390 ms
Tiempo para la detección sobre una región	50-70 ms	60-70 ms
Tiempo para la predicción	40-60 ms (predicción y corrección)	30-40 ms
Tiempo total en detección de un fotograma completo	140-180 ms	430-450 ms
Tiempo total en detección solo de regiones en un fotograma	110-140 ms	90-100 ms

Tanto el simulador como el vehículo pueden llevar a cabo la ejecución sin problemas, siendo el primero hasta 10 veces más rápido. De acuerdo con los tiempos que le toma procesar cada fotograma, en el simulador se pudo haber tomado una tasa ligeramente más alta que 5 Hz. Pero en el caso del carro parece ser la tasa más alta que puede alcanzar, considerando que la detección del frame completo le toma cerca de medio segundo, mientras que la detección en regiones es más rápida.

En el caso para cargar el filtro de cascada, al ser muy ligero, prácticamente no demora tiempo, apenas milisegundos para el simulador. Por un lado, en la detección, hacer el barrido con

ventana deslizante con SVM sería muy tardado, aun haciéndose con el filtro de cascada, es la parte de la detección que más consume tiempo. Por otro lado, en el rastreo, la mayor parte del tiempo lo consume la predicción y corrección de los filtros, crear uno nuevo es instantáneo. De lo observado en el carro físico, es que demora más en la predicción con las máquinas de soporte vectorial.

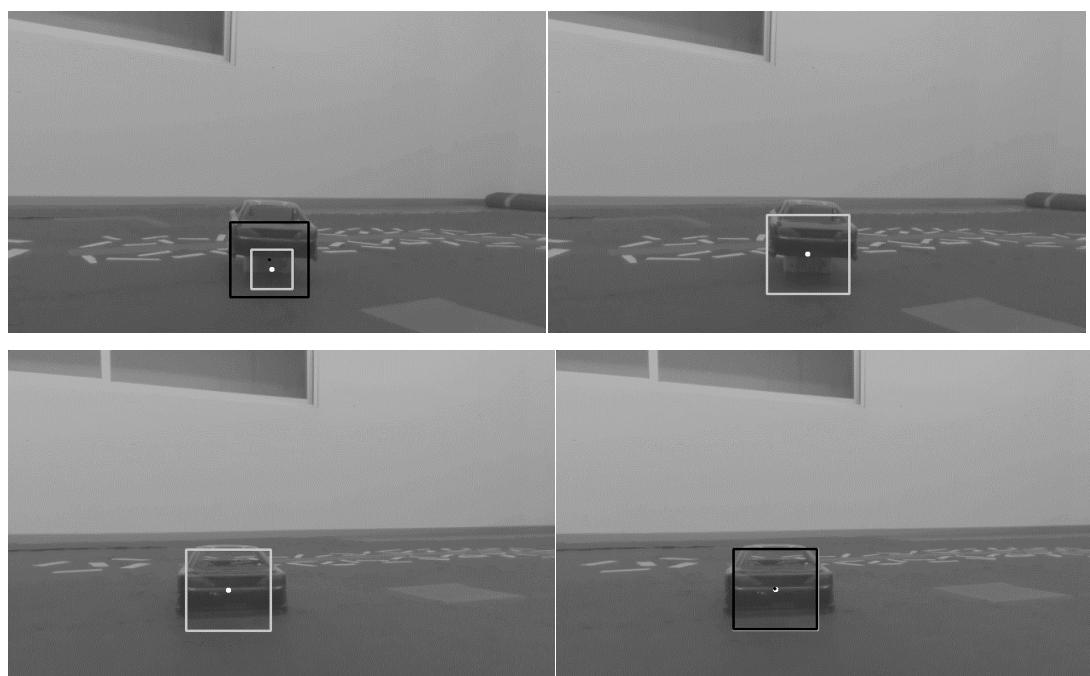


FIGURA 6-11 DETECCIONES EN EL CARRO FÍSICO

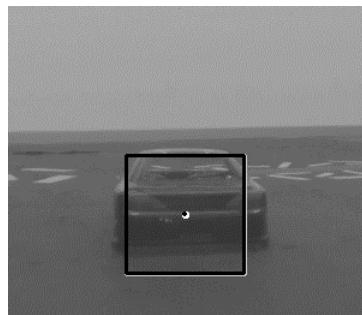


FIGURA 6-12 EJEMPLO DE LAS  
DETECCIONES

La mayor limitante en el modelo de carro autónomo es su poca memoria RAM, pues solo tiene 2 GB. Tan solo para cargar los paquetes básicos, nodos como la cámara, el lidar, motor\_control y el rosmaster, ocupa cerca de 1100 MB (55%). Al añadir el nodo de detección y rastreo, la memoria RAM usada es aproximadamente 1650 MB (82.5%), el problema principal viene de cargar ambas máquinas de soporte vectorial, que son bastante pesadas. En la utilización del CPU, cuenta con 8 núcleos, de los cuales la cámara es la que más ocupa, 17.9%, y el programa de detección y rastreo el 14.75% total. Ambos ocupan más de un núcleo de CPU.

## **7 DISEÑO DE LA RED VANET**

Este capítulo aborda el diseño de las simulaciones para probar los protocolos elegidos para una red Vehicular Ad-hoc Network (VANET). Los protocolos sobre los que se experimenta son el Ad-hoc On-demand Distance Vector (AODV) y Dedicated Short-Range Communications (DSRC). Igualmente, se discuten los contenidos que deberían transmitirse en los mensajes para notificar la ubicación de otros vehículos.

### **7.1 Arquitectura**

Con el propósito de mantener a los vehículos de la red alerta ante la presencia de otros vehículos y prevenir accidentes, se propone utilizar DSRC para la comunicación de las detecciones realizadas en la primera parte del trabajo y AODV para la coordinación entre vehículos. Partiendo de esta premisa, se realiza una simulación para determinar el riesgo o probabilidad de choque entre dos vehículos. En este escenario, un vehículo intenta ingresar a una vía rápida donde no tiene línea de visión con otros vehículos, lo que hace que el nodo detector del vehículo no sea útil por sí solo, como se muestra en la figura 7-1. Sin embargo, en un entorno colaborativo, los otros vehículos comparten información sobre sus detecciones, lo que permite que ambos vehículos notifiquen la presencia del otro y coordinen sus acciones posteriormente. En la simulación, se establece una distancia específica en la que los vehículos deben recibir al menos un número determinado de mensajes de otros vehículos para estar informados.

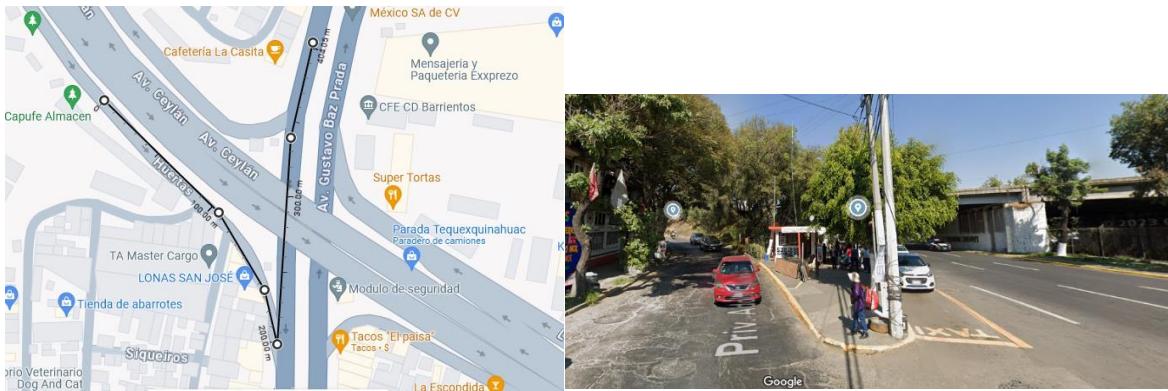


FIGURA 7-1 CORRESPONDE A LA INCORPORACIÓN DE VEHÍCULOS DE LA AVENIDA CEYLÁN, SOBRE LA AV. GUSTAVO BAZ

En ambas simulaciones se calcula el riesgo de colisión entre nodos en función de la cantidad de mensajes que logran recibir. En las simulaciones del protocolo AODV y DSRC, el vehículo que busca incorporarse recorre una distancia de 100 y 200 metros y con una densidad variable de vehículos. Por lo tanto, antes de llegar a su destino, teniendo en cuenta una distancia de seguridad apropiada según la velocidad que lleva, debe recibir al menos dos mensajes. Nos limitamos solo a algunos casos de prueba por limitaciones de tiempo para este trabajo, pero el objetivo es mostrar cómo pueden adaptarse algunos parámetros para la comunicación en entornos variables y conforme se reduce la distancia.

Para la implementación de ambas simulaciones se usa OMNeT++, una biblioteca y marco de simulación en C++ orientado principalmente a modelar el tráfico en redes de telecomunicaciones. Entre sus principales bibliotecas se encuentra INET que proporciona protocolos, agentes y modelos tanto de Internet, de capa de enlace alámbrica e inalámbrica, y soporte para movilidad. Ambos nos servirán para la representación del uso de AODV en

redes VANET. Adicionalmente, para la segunda simulación se utilizó VEINS y SUMO, ya que cuentan con paquetes específicos para el protocolo DSRC.

En OMNeT++, una simulación se compone de tres archivos principales: el archivo .ned se utiliza para agregar los módulos que se emplean y describir la topología de la red; el archivo .ini se utiliza para configurar la red; y la aplicación programada en C++ que describe el comportamiento de los nodos.

En el archivo .ned se agregan los nodos de la simulación tipo ManetRouter que representan a los vehículos. Estos nodos se tratan de hosts inalámbricos que incorporan componentes de enrutamiento, movilidad y energía. Además de estos nodos, se deben incorporar otros módulos para simular el medio físico, como RadioMedium y physicalEnvironment. También se añaden un visualizador y un configurador para redes IP.

En el archivo .ini se manda a llamar el archivo .ned, los archivos .xml para obstáculos y otras configuraciones. En este archivo se puede ajustar el movimiento de los nodos y la visualización de la simulación. Aquí se configura la capa física y de enlace de datos (MAC), definiendo qué protocolos utilizar, la interfaz y asignándoles una radio (frecuencia, sensibilidad, etc.). Los protocolos implementados están disponibles en bibliotecas que contienen los modelos, como INET y VEINS. Es relevante destacar que en ambas simulaciones solo se contempla recibir el mensaje para decir que el vehículo ha quedado notificado, no se toma en cuenta el tiempo de su procesamiento.

La primera simulación basada en AODV requiere los archivos mencionados anteriormente. Para la aplicación, se utiliza una ya predefinida llamada "pingApp", que incluye un contador de paquetes enviados y recibidos. La aplicación genera solicitudes de ping a varios hosts y calcula la pérdida de paquetes y, los tiempos de ida y vuelta de las respuestas (RTT). Además, se incluye un archivo .xml que describe los obstáculos presentes, como edificios y otras estructuras. En el archivo .ini se especifica el movimiento de los nodos.

En la simulación 2, centrada en DSRC, se amplía la funcionalidad de OMNeT++ al incorporar el uso de "Vehicles in Network Simulation" (VEINS), un framework de código abierto diseñado para ejecutar simulaciones de VANETs. Entre sus funciones incluye una implementación del protocolo DSRC y WAVE. También se emplea "Simulation of Urban MObility" (SUMO), un paquete de simulación de tráfico diseñado para manejar redes extensas, lo que permite modelar sistemas de tráfico intermodal que involucran vehículos de carretera, transporte público y peatones.

En esta segunda simulación, la topología y el movimiento de los nodos se describen ahora en SUMO, y esta información se importa en el archivo .ned. Para la aplicación, se parte de la aplicación base "veinsInetSampleApplication" y se modifican las funciones startApplication() para definir el envío de mensajes. En esta sección del código se programa para que periódicamente se informe por broadcasting las detecciones de otros vehículos. La función processPacket(pk) altera el comportamiento de los nodos al recibir un mensaje, indicando que los nodos retransmitan el mensaje una vez (esto requiere de un buffer para

guardar los últimos mensajes recibidos y verificar si son nuevos). También incluye una función stopApplication(), aunque en este caso no se utiliza.

Igual que en AODV, se usa el mismo archivo .xml para definir los obstáculos (mediante SUMO también se podían modelar los obstáculos, pero no se pudieron importar a OMNeT++). El archivo .sumocfg lleva las direcciones de los demás archivos de SUMO. Config.xml incluye el tipo de modelo de propagación usado, el protocolo y frecuencia. El .rou.xml para declarar la rutas, los nodos y su tiempo de partida. Por último, el .net.xml con el mapa y la velocidad máxima en cada camino de las rutas.

## 7.2 Simulación 1

Para la primera simulación se preparan un escenario como el que se observa en la figura 7-2, para los que se experimenta los resultados del protocolo AODV al intentar mandar un mensaje ping entre dos nodos, se mide el porcentaje de mensajes recibidos y, el tiempo de ida y vuelta de un mensaje. Entre los parámetros principales para ver su desempeño, se varía la velocidad de los nodos, y el intervalo de transmisión entre mensajes.



FIGURA 7-2 SIMULACIÓN 1, SE EXPERIMENTA EL DESEMPEÑO CON DIFERENTE DENSIDAD DE NODOS. EL RECUADRO AZUL ES LA FUENTE Y EL ROJO EL DESTINO

Los resultados varían según una serie de parámetros que configuran tanto el medio físico como la capa MAC en la simulación. Para el medio físico, al basarse en el estándar 802.11p, la frecuencia de transmisión se fija en 5.9 GHz, y la potencia máxima en 200 mW (23 dBm). Asimismo, la sensibilidad del receptor juega un papel crucial; en este caso, se optó por un valor normal que es de -85 dBm.

Con estos parámetros, y considerando una ganancia de las antenas de 0 dBm, el modelo de “free space path loss” indica una cobertura con línea de visión de 1 km de radio, distancia más que suficiente para detectar y alertar sobre posibles accidentes con anticipación. El problema en cambio se encuentra con los obstáculos, pues pueden ser puntos ciegos para los vehículos, por lo que se considera el escenario de “ideal obstacle loss”, donde la señal no atraviesa las construcciones.

Igualmente, en el protocolo de AODV es importante considerar puntos como el intervalo en que se mantienen activas las rutas. Un intervalo corto significa mayor inundación en la red de mensajes, mientras que un intervalo largo puede tener problemas debido a que la topología cambia con los nodos en movimiento. Por ello en cada escenario se experimenta con diferentes velocidades: 30, 50 y 80 km/h. En las que los nodos deben cubrir la misma distancia recorrida, la cual es de 100 metros.

En cuanto al interés de variar el intervalo de transmisión entre mensajes ping, es que AODV se trata de un protocolo de inundación reactivo. Un intervalo alto siempre significará mandar mensajes para encontrar una ruta al nodo destino. Por otro lado, un intervalo bajo puede o no

llevar a la inundación de la red. Esto dependerá de cómo cambia la topología de acuerdo con la velocidad y densidad de nodos. En caso de grandes cambios, puede llevar a retrasos en la red si se necesitan buscar constantemente nuevas rutas. De forma que en estos casos conviene que el intervalo de envío de mensajes sea adaptable según las características de la red, dichos parámetros recomendables son los que buscamos en la simulación.

En esta simulación 1, se representa el nodo destino que se va a incorporar a la vía principal y el nodo fuente, que no tiene línea de vista, busca comunicarse con él. Utilizando el protocolo AODV, un nodo comienza buscando una ruta en sus tablas hacia el nodo destino.

En caso de no tener una ruta fresca, envía un mensaje de solicitud por broadcast a los nodos cercanos, los cuales responden si conocen una ruta o retransmiten la solicitud a otros nodos.

Una vez que se obtiene la ruta, el nodo envía el mensaje y espera una respuesta. La idea detrás de que la fuente sea el vehículo sobre la vía principal y no al revés, viene de cómo se enteran de la presencia del otro. Los vehículos que se encuentran delante de él detectan al vehículo que se aproxima para incorporarse y comparten esas detecciones. Los mensajes terminan llegando al vehículo de atrás, quien procesa el mensaje, observa que sus trayectorias coinciden e inicia la conversación por AODV.

Las características de nuestro escenario inicial son:

- Potencia de transmisión 200 mW
- Sensibilidad del receptor -85 dBm
- Velocidad de los nodos 30 km/h
- Tasa de transmisión 6 mbps
- Densidad baja
- Intervalo entre mensajes 1s

- Tiempo de vida de las rutas 0.5s

Para recopilar los resultados de las simulaciones, se llevan a cabo 30 ejecuciones en cada escenario. En la Figura 7-3 se muestran las posiciones iniciales y finales de los nodos. Se diseñó el primer escenario con la intención de crear un momento en el que los nodos quedarán incomunicados, lo cual será relevante al reducir el intervalo entre mensajes, ya que los nodos deberán enviar mensajes del protocolo AODV para buscar nuevas rutas.



FIGURA 7-3 A LA IZQUIERDA LA POSICIÓN INICIAL, A LA DERECHA LA POSICIÓN FINAL

### 7.3 Simulación 2

Para la segunda simulación se observa el desempeño del protocolo DSRC en el que se considera el caso en el que todos los nodos intentan comunicarse entre sí. Con el objetivo de informar de las detecciones, los nodos al recibir un mensaje lo retransmiten siempre una vez, incluido el que lo envió originalmente (para agregar redundancia). Al igual que en la simulación anterior, se experimenta variando la velocidad de los nodos y el intervalo entre los mensajes, se mide la tasa de errores, tiempos de encolamiento, y el número de mensajes enviados y recibidos.

Para esta simulación, gracias al uso de VEINS, se implementa de forma completa el protocolo 802.11p. En el archivo omnetpp.ini es donde se especifican la capa física y MAC, junto a los archivos de configuración (SUMO) y de obstáculos (junto al modelo de propagación) de la red. Las modificaciones principales se hacen sobre la aplicación, para definir eventos de cuando transmitir un mensaje y qué hacer al recibir uno. Como en la anterior, cada escenario se ejecuta 30 veces y se promedian los resultados.

Para experimentar con la velocidad, y ver qué tantos mensajes pueden recibir al recorrer una misma distancia, como se observa en la figura 7-4, los nodos parten de una posición inicial a una final.



FIGURA 7-4 A LA IZQUIERDA SE OBSERVA LA POSICIÓN DE PARTIDA DE LOS NODOS, A LA DERECHA LA POSICIÓN FINAL

Cada nodo intenta transmitir de forma aleatoria dentro de un intervalo de 100 ms, de forma que podamos observar distintos comportamientos, como retrasos en las transmisiones. Durante la simulación los nodos 3, 8 y 6, no son alcanzados por quien transmite el mensaje original, pero si con las retransmisiones.

Con esta simulación se prueban distintos escenarios, variando el intervalo de transmisión, se expande la cantidad de nodos, así como el tamaño de mensajes. Al aumentar la cantidad de

nodos, se espera que algunos intervalos de transmisión, los más pequeños como de 0.2 e incluso 0.5 segundos dejen de ser viables. Igualmente, al aumentar el tamaño del mensaje, el porcentaje de recibidos caerá. Sin embargo, en algunos casos convendrá transmitir mensajes más grandes, en lugar de intentar acceder múltiples veces al medio, sobre todo a altas velocidades. Esto funcionará mientras los tiempos de encolamiento no superen sus intervalos de transmisión, pues usar mensajes más grandes, significa que los nodos deben esperar más en promedio para acceder al medio.

El acercamiento tomado, es que un nodo para estar notificado de la presencia de otro vehículo debe recibir cierta cantidad de mensajes. Partiendo de que cada nodo se desea que comparta sus detecciones y las posiciones predichas de ellas, en forma de coordenadas. Se toma como tamaño de mensaje 2 kB de payload, el cual viene de probar experimentalmente cuanto ocuparía el mensaje que se busca enviar con 15 detecciones. Se compara la probabilidad entonces de recibir 2 mensajes de 1 kB vs 1 mensaje de 2kB, y de recibir 4 mensajes de 1 kB vs 2 mensajes de 2 kB.

Para la probabilidad, el vehículo viaja a diferentes velocidades (30, 50, 80 km/h) en las que recorre un trayecto de 100 metros y 200 metros. Considerando, además, una distancia de seguridad antes de la cual deben haber recibido los mensajes y así reducir el riesgo de colisión. Para esto, se toma de base la distancia que se tiene que guardar entre dos automóviles para frenar a tiempo. Hay varias formas de calcularla, pero la usada fue la regla de 2 segundos, que consiste en multiplicar por dos la velocidad en metros por segundo. Por

ejemplo: en 100 metros, un vehículo viajando a 50 km/h y con un intervalo de transmisión de 1 segundo, tendría la oportunidad de mandar 7 mensajes. Considerando una distancia de seguridad, como que el nodo debe quedar advertido aproximadamente 28 metros antes, solo tiene la oportunidad de enviar 5 mensajes.

Como en la simulación 1, se busca obtener el mejor intervalo para la comunicación, y tamaño de los mensajes para cada escenario. Además, hay que considerar que, en el caso de 100 metros antes de la intersección, algunos intervalos serán insuficientes para algunas velocidades. Estos resultados, asumiendo que siempre hay una ruta entre los nodos, serían nuestros parámetros recomendados para lograr o maximizar la probabilidad de que los vehículos queden enterados.

#### **7.4 Contenido de los mensajes**

Para la prevención de accidentes, es crucial alertar de manera oportuna sobre los movimientos de otros vehículos, como cambios de carril o frenadas bruscas. En el contexto de DSRC, los mensajes transmitidos deben incluir información detallada, como la ubicación precisa del vehículo y la hora proporcionada por GPS, la ubicación aproximada de las detecciones, la velocidad y aceleración aproximadas, y el uso de bits para indicar acciones como frenado y cambio de carril, tanto propias como de otros vehículos. Siguiendo la tabla 3-2, otra información relevante que se puede incluir es el tipo de vehículo, la dirección a la que viaja, y las acciones evasivas que podrían tomar en respuesta a las alertas recibidas.

Siguiendo un enfoque de agregación, cada vehículo no retransmite los mensajes de manera inmediata, sino que los modifica, agrega y compara con su propia información, lo que implica que el tamaño del área de datos de los mensajes puede variar. Además, en un contexto de geocasting, no toda la información es relevante en todas las circunstancias. Por lo tanto, al alejarse de una región específica, ciertos datos, como las ubicaciones de algunos vehículos, deben descartarse para mantener la eficiencia de la comunicación.

## **8 IMPLEMENTACIÓN DE UNA RED VANET**

En este capítulo se describe a detalle cómo se implementan las simulaciones usando OMNeT++ para AODV, y para DSRC se añade el uso de VEINS y SUMO.

### **8.1 Arquitectura y requisitos de la red VANET**

Siguiendo el capítulo anterior, nuestro modelo se fundamenta en los protocolos de AODV y DSRC. Este modelo está diseñado específicamente para la prevención de accidentes, donde los mensajes, de tamaño variable, contienen información tanto del propio vehículo como de las detecciones realizadas. Además, se contempla la posibilidad de que no todos los nodos estén conectados de manera simultánea, es decir, que no todos los nodos sean visibles para todos los participantes en la red en todo momento.

El contenido de los mensajes debe incluir: en el caso del usuario, la identificación del vehículo, ubicación geográfica que requeriría del uso de sistemas como DGPS (que ofrecen mayor precisión), información referente a las rutas que tomará, como próximos movimientos, su velocidad y aceleración; en el caso de las detecciones comunicar su posición actual, velocidad y aceleración estimada, junto a sus próximos movimientos predichos.

Esta información plantea tres casos. En el primero, se considera una actualización periódica de los datos, donde los vehículos procesan la información recibida y la comparan con la suya propia, para luego reenviar el mensaje en forma de un resumen actualizado. En el segundo caso, el envío de mensajes de emergencia, como la detección de un vehículo que frena o

cambia de carril. Ambos escenarios involucran mensajes de tipo broadcast, para los cuales se contempla que solo se retransmitan una vez. En el tercer caso, se plantea la posibilidad de establecer conexiones directas con otros vehículos (unicast) para verificar información o coordinar acciones.

En función de la necesidad de minimizar la latencia, se opta por una red VANET basada en comunicación vehículo a vehículo (V2V), aunque se pueden incorporar modelos V2I o V2X para otros servicios, como la planificación de rutas. Dentro del modelo V2V, se escoge como protocolo de enrutamiento para la comunicación uno a uno AODV, mientras que para broadcast se elige el estándar 802.11p (primera parte de DSRC), con soporte de CSMA/CA.

## **8.2 Implementación**

Para la presentación de resultados, en ambas simulaciones vehículos deben recorrer 100 metros a diferentes velocidades. Es decir, a 30 km/h, el intervalo en el que los nodos se comunican es de 12 segundos; a 50 km/h, de 7.2 segundos; y a 80 km/h, de 4.5 segundos.

### **8.2.1 AODV**

Para la primera simulación, el proyecto se compone de tres archivos principales, el primero es un archivo .ned, que corresponde a la descripción de la topología de la red, como se observa en las figuras 8-1 y 8-2:

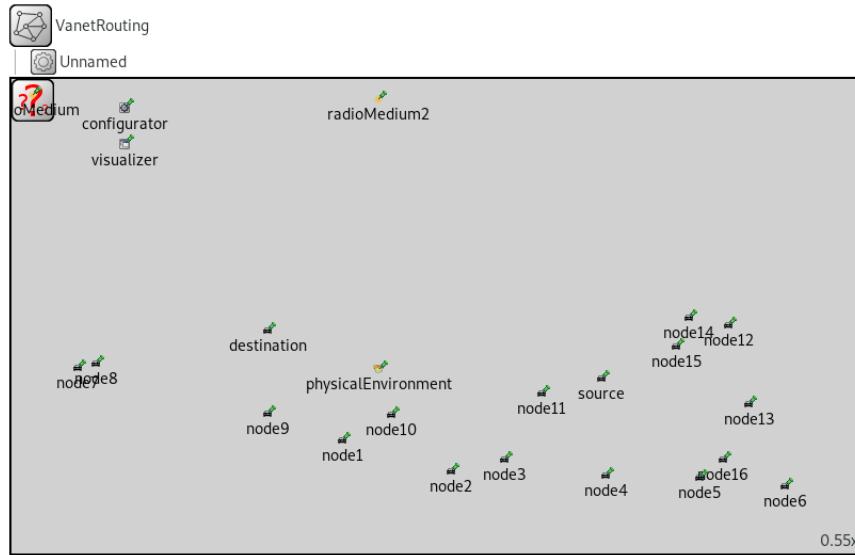


FIGURA 8-1 VENTANA DE DISEÑO PARA UN ARCHIVO .NED

```

@network VanetRouting
{
    parameters:
        @display("bgc=1462,816;bgf=background/mapa2");
        @statistic[numRcvdPk](source=count(source.app[0].pingRxSeq); record=figure; targetFigure=numRcvdPkCounter; checkSignals=false);
        @figure[numRcvdPkCounter](type=counter; transform=scale(0.5); pos=50,300; label="Received ping replies"; decimalPlaces=4);
        @statistic[numLost](source=count(source.app[0].numLost); record=figure; targetFigure=numLostPkCounter; checkSignals=false);
        @figure[numLostPkCounter](type=counter; transform=scale(0.5); pos=50,400; label="Pings lost"; decimalPlaces=4);

    types:
        simple Unnamed
        {

    submodules:
        source: ManetRouter {
            @display("p=1014,515;i=misc/aaa");
        }
        node1: ManetRouter {
            @display("p=569,622;i=misc/aaa");
        }
        node2: ManetRouter {
            @display("p=756,675;i=misc/aaa");
        }
        destination: ManetRouter {
            @display("p=440,434;i=misc/aaa");
        }
        //ApskScalarRadioMedium
        radioMedium: Ieee80211ScalarRadioMedium {
            //radioMedium: ApskScalarRadioMedium{
                @display("p=38,28");
            }
        configurator:Ipv4NetworkConfigurator {
            @display("p=194.24501,49.4775");
        }
        visualizer: IntegratedMultiVisualizer {
            @display("p=194.24501,111.7825");
        }
}

```

FIGURA 8-2 CÓDIGO DEL ARCHIVO .NED, SON LOS MÓDULOS INCLUIDOS

En el archivo .ned se agregan los nodos tipo ManetRouter, que en nuestro caso representan los vehículos, junto a los módulos:

- radioMedium se usa el ApskScalarRadioMedium, que describe el entorno en que ocurre la comunicación, incluye modelos de propagación, calcula el ruido del medio, y cómo afecta la transmisión
- configurator se usa IPv4NetworkConfigurator para la asignación de direcciones a los nodos, y en este caso se debe definir que no se tienen rutas estáticas.
- visualizer se usa el IntegratedMultiVisualizer para generar anotaciones, como los links formados.
- physicalEnvironment para la descripción de geometrías como paredes y construcciones, que afectaran la propagación de la señal.

Para mejorar la visualización de la simulación, se descargó un mapa y se agregó un ícono para los vehículos, los cuales se incluyen con el comando display:

```
@display{"bgb=1462,816;bgi=background/mapa2"};
```

Ó

```
@display{"p=1014,515;i=misc/auto"};
```

El parámetro bgi permite agregar una imagen de fondo, mientras que i modifica el ícono. En ambos casos, se debe proporcionar la dirección del archivo correspondiente. Con bgb se establece el tamaño de la imagen de fondo, y con p la posición del nodo.

Una vez descrita la red, se escribe un archivo .xml que contiene los obstáculos. Se incluyen objetos de rectangulares, se especifica la posición, orientación y el material del que están hechos:

```
<environment>

<object position="min 265 342 0" orientation="99.26 0 0"
shape="cuboid 5 305 8" material="concrete" fill-
color="255 255 255" opacity="0.8"/>

</environment>
```

El último archivo es de tipo .ini, el archivo de configuración de la red y todos los parámetros, dentro se define el tiempo de simulación y el archivo NED a usar.

```
sim-time-limit = 12s

network = VANETRouting
```

Los nodos necesitan ejecutar una aplicación para la comunicación. En este caso, se utiliza una aplicación predefinida llamada "PingApp". En el archivo PingApp.ned se modifica el tamaño del ping (packetSize) para representar el tamaño del paquete, que son 2 kB. Aunque, como se mencionó anteriormente, no se toma en cuenta el procesamiento del paquete, solo recibirlo. Por lo que solo nos enfocamos en el encabezado. En este escenario, un vehículo que busca incorporarse a una vía rápida (nodo destino) se coordina con otro vehículo en la vía (nodo fuente) que ha sido previamente informado de su presencia por DSRC.

```
*.source.numApps = 1

*.source.app[0].typename = "PingApp"

*.source.app[0].destAddr = "destination"
```

```
*.source.app[0].printPing = true
```

A continuación, se configura la interfaz de red para que los nodos se comuniquen entre sí, en este caso la de WiFi 802.11, se define el bitrate de 6 Mbps, la potencia de transmisión de 200 mW, el ancho de canal de 10 MHz, estos valores son de acuerdo al estándar de DSRC.

```
*.*.wlan[*].typename = "Ieee80211Interface"  
*.*.wlan[*].bitrate = 6Mbps  
*.*.wlan[*].mac.**.responseAckFrameBitrate = 6Mbps  
*.*.wlan[*].mac.**.*Retry* = 0  
*.*.wlan[*].radio.transmitter.power = 200mW  
*.*.wlan[*].radio.receiver.sensitivity = -85dBm  
*.*.wlan[*].radio.receiver.snirThreshold = 0.1dB  
*.*.wlan[*].radio.receiver.energyDetection = -90dBm  
*.*.wlan[*].radio.bandwidth = 10MHz
```

Para la capa física, se incluye la configuración del radio, aquí se opera sobre el módulo basado en IEEE 802.11, se define la frecuencia central de 5.9 GHz, y el uso de ODFM.

```
*.radioMedium.typename = "Ieee80211ScalarRadioMedium"  
*.*Host.wlan[*].radio.typename = "Ieee80211Radio"  
*.*.wlan[*].radio.typename = "Ieee80211OfdmRadio"  
*.*.wlan[*].radio.centerFrequency = 5.9GHz
```

Para agregar el archivo .xml:

```
*.physicalEnvironment.config = xmlDoc("walls.xml")  
*.radioMedium.obstacleLoss.typename = "IdealObstacleLoss"
```

Para la movilidad de los nodos en este ejemplo, se introdujo de forma manual, especificando la dirección en la que se mueven con un ángulo y su velocidad:

```
*.node*.mobility.typename = "LinearMobility"  
*.node1.mobility.initialMovementHeading = uniform(6deg,10deg)  
*.node5.mobility.initialMovementHeading = uniform(186deg,190deg)  
*.node*.mobility.speed = 8.33mps
```

Mientras que el área se define con:

```
**.constraintAreaMaxX = 1462m  
**.constraintAreaMaxY = 816m
```

Para agregar el uso de AODV, se incluye una aplicación para el enrutamiento:

```
*.*.routingApp.typename = "Aodv"  
*.*.routingApp.activeRouteTimeout = 0.5s  
*.*.routingApp.deletePeriod = 0.1s
```

Para recolectar y generar estadísticas:

```
[Config StatisticBase]  
extends = Aodv  
**.vector-recording = false  
repeat = 30
```

En la figura 8-3 se observa la ventana al ejecutar la simulación:



FIGURA 8-3 IMAGEN DE LA SIMULACIÓN DE AODV

### 8.2.2 IEEE 802.11p

Aquí se comienza con el modelo en SUMO, se trazan de forma gráfica las rutas, junto con los edificios como se observa en la figura 8-4. Sin embargo, no se pudieron agregar en OMNeT++ los obstáculos de esta forma, por lo que se usó el archivo .xml escrito en la simulación anterior.

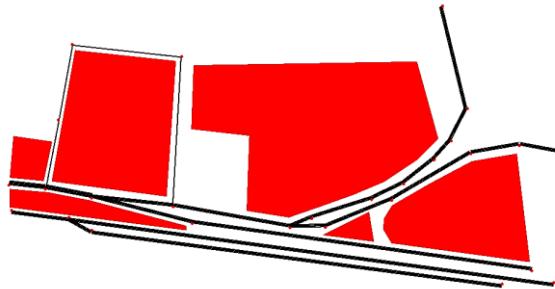


FIGURA 8-4 IMAGEN DEL EDITOR DE UNA RED EN SUMO

Se elaboró un archivo .launch que contiene los programas de SUMO. El archivo mapa2.net.xml describe los caminos, intersecciones y la velocidad máxima en cada uno. Por

su parte, el archivo rutas2.rou.xml son las rutas (figura 8-6) y los vehículos (para estos se define la hora de salida y la ruta que toman). Además, se incluyó el archivo walls.poly.xml para definir las construcciones, indicando que son de tipo "building" (figura 8-5). Finalmente, SUMO crea el archivo hello2.sumocfg al ejecutarse, el cual contiene la ubicación de los tres archivos mencionados anteriormente.

Archivo .launch, únicamente se agrega el nombre de cada archivo.

```
<?xml version="1.0"?>
<!--debug config-->
<launch>
    <copy file="mapa2.net.xml"/>
    <copy file="rutas2.rou.xml"/>
    <copy file="walls.poly.xml"/>
    <copy file="hello2.sumocfg" type="config"/>
</launch>
```

Para definir los caminos, se asigna un nombre (id) y se especifican manualmente dos puntos (from y to), además de la velocidad y el número de carriles. En cuanto a las construcciones, se les asigna un identificador y se especifica el tipo "building". Posteriormente, se trazan manualmente los puntos sobre el mapa que forma el polinomio, al seleccionar la opción shape.

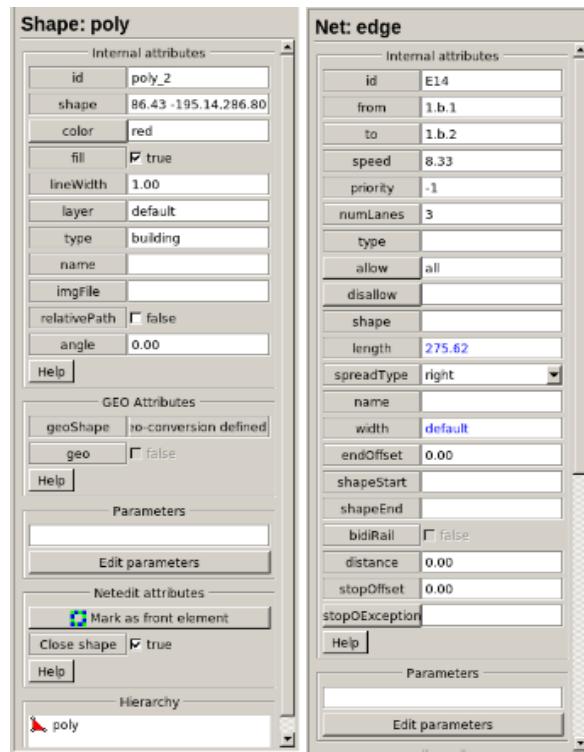


FIGURA 8-5 PESTAÑAS PARA CREAR CONSTRUCCIONES (A LA IZQUIERDA) Y CAMINOS (A LA DERECHA)

Para crear una ruta se seleccionan los caminos que los componen, se elige el tipo de vehículos que pueden transitar por ellos (de pasajeros). Algunos de los caminos deben especificarse que son intersecciones.



FIGURA 8-6 VENTANA PARA CREAR RUTAS

En el caso de los vehículos, se añaden en el archivo de rutas, asignándoles un identificador (id), indicando la ruta seleccionada y especificando el tiempo de salida.

```
<vehicle id="vehicle_1" depart="2.00" route="route_1"/>
<vehicle id="vehicle_2" depart="4.00" route="route_3"/>
<vehicle id="vehicle_3" depart="5.00" route="route_5"/>
```

En la figura 8-7 se observa la simulación corriendo en SUMO:

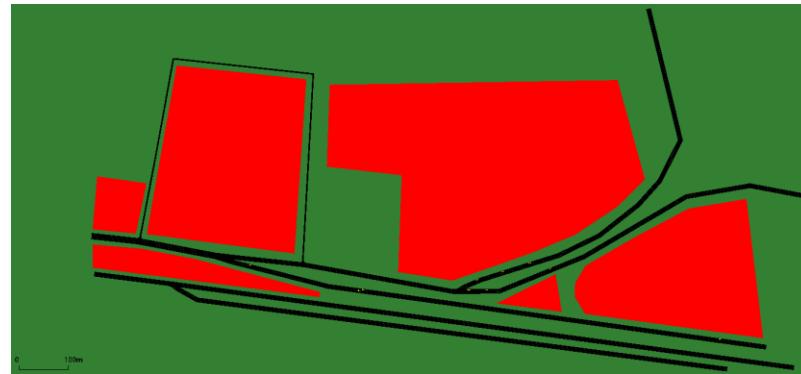


FIGURA 8-7 IMAGEN FINAL EN SUMO

Una vez finalizada la simulación en SUMO, el siguiente paso es integrarla en OMNeT++.

Esto se puede lograr partiendo del ejemplo incluido en `veins_inet/examples/veins_inet`. Es necesario agregar los archivos correspondientes a las rutas, la red, el archivo de configuración y el archivo de lanzamiento. En el caso de los obstáculos, se deben agregar módulos adicionales en el archivo `.ned` para que funcionen, o incluirlos como en la primera simulación. El archivo `.ned`, que se muestra en la figura 8-8, al igual que el anterior, incluye módulos para el radio, modelar el entorno, visualizadores y un manager para controlar la comunicación.

Se diseñaron tres escenarios, dos como ejemplos visuales. En el primer escenario, la simulación 1, se simula un accidente vehicular. En el tercer escenario, se simula la incorporación de un vehículo a una vía. Para los resultados, se analiza la simulación 2, en la que todos los vehículos comparten periódicamente las detecciones realizadas.

```

network sim80211p
{
    parameters:
        bool useOsg = default(false);
        @display("bgb=1467,819;bgi=background/Imagen2A");
    submodules:
        radioMedium: Ieee80211DimensionalRadioMedium {
            @display("p=64,224");
        }
        manager: VeinsInetManager {
            @display("p=192,320");
        }
        visualizer: IntegratedVisualizer {
            @display("p=64,320");
        }
        physicalEnvironment: PhysicalEnvironment {
            @display("p=192,224");
        }
        roadsCanvasVisualizer: RoadsCanvasVisualizer {
            @display("p=64,416");
        }
        roadsOsgVisualizer: RoadsOsgVisualizer if useOsg {
            @display("p=192,416");
        }
        //obstacles: ObstacleControl {
        //    @display("p=240,50");
        //}
        node[0]: VeinsInetCar;
}

```

FIGURA 8-8 ARCHIVO .NED PARA LA SEGUNDA SIMULACIÓN

En cuanto al archivo de configuración omnetpp.ini se agrega nuevamente el archivo .ned, y el tiempo de simulación, el escenario 1 y 2 de 120 segundos, el 3 de 260 s. Para los resultados, se utiliza el escenario 2, en el cual los nodos transmiten durante un recorrido de 100 metros. Sin embargo, al utilizar SUMO, los vehículos deben iniciar desde el origen de la ruta, se simula durante 120 segundos, para esperar a que se posicen correctamente.

```

network = sim80211p
sim-time-limit = 120s #120 sim 1 y 2, 260 sim3 2
debug-on-errors = true
cmdenv-express-mode = true
image-path = ../../../../images

```

Igual que la simulación anterior, se requiere de una aplicación y una interfaz inalámbrica:

```

*.node[*].numApps = 1
*.node[*].app[0].typename =

```

```

"org.car2x.veins.subprojects_VEINS_INET.VeinsInetSampleApplication"

*.node[*].app[0].interface = "wlan0"

```

La siguiente configuración corresponde a los nodos, las líneas resaltadas se refieren directamente el protocolo de DSRC. Para todos los nodos se agrega el estándar 802.11p con la opción de opMode, junto con el radio, la banda de frecuencia que es 5.9 GHz, potencia de transmisión 200 mW y el ancho de banda 10 MHz. Para la capa mac se utiliza el módulo de INET Ieee80211Mac. Mientras el hostAutoConfigurator permite agregar direcciones ip.

```

*.node[*].wlan[0].opMode = "p"
*.node[*].wlan[0].radio.typename =
    "Ieee80211DimensionalRadio"
*.node[*].wlan[0].radio.bandName = "5.9 GHz"
*.node[*].wlan[0].radio.channelNumber = 3
*.node[*].wlan[0].radio.transmitter.power = 200mW
*.node[*].wlan[0].radio.bandwidth = 10 MHz
*.node[*].wlan[*].radio.antenna.mobility.typename =
    "AttachedMobility"
*.node[*].wlan[*].radio.antenna.mobility.mobilityModule =
    "^.^.^.^.mobility"
*.node[*].wlan[*].radio.antenna.mobility.offsetX = -2.5m
*.node[*].wlan[*].radio.antenna.mobility.offsetZ = 1.5m
*.node[*].wlan[0].mac.typename = "Ieee80211Mac"
*.node[*].wlan[0].mac.modeSet="p"
*.node[*].ipv4.configurator.typename = "HostAutoConfigurator"
*.node[*].ipv4.configurator.interfaces = "wlan0"
*.node[*].ipv4.configurator.mcastGroups = "224.0.0.1"

```

El módulo “VeinsInetMobility” permite rastrear la posición del nodo, aquí se usa el manager para agregar los archivos de SUMO (la línea resaltada). Para los obstáculos se agregan con el módulo physicalEnvironment junto a un modelo de propagación, que en este caso IdealObstacleLoss se refiere que la señal no pasará los obstáculos, toda la potencia se pierde.

```

*.node[*].mobility.typename = "VeinsInetMobility"

*.manager.updateInterval = 0.1s
*.manager.host = "localhost"
*.manager.port = 9999
*.manager.autoShutdown = true
*.manager.launchConfig = xmldoc("VANETSumo2Sim2.launchd.xml")
*.manager.moduleType =
    "org.car2x.veins.subprojects.veins_inet.VeinsInetCar"
*.physicalEnvironment.config = xmldoc("walls2.xml")
*.radioMedium.obstacleLoss.typename = "IdealObstacleLoss"

```

Como en AODV, para realizar múltiples corridas se agrega al final del archivo .ini:

```

[Config StatisticBase]

**.vector-recording = false

repeat = 30

```

En cuanto a las aplicaciones, se refieren a módulos que implementan comportamientos específicos en cada nodo para generar patrones de tráfico. En nuestro modelo, existen dos métodos principales: el método startApplication(), utilizado para controlar los nodos y gestionar el envío de mensajes, generar eventos y controlar el comportamiento de los nodos (velocidad, rutas, etc.) con VEINS. El método processPacket() define el comportamiento de los nodos al recibir un mensaje.

Se presentan dos casos distintos: el primero implica el envío de mensajes de emergencia, que pueden representar un accidente o la incorporación de un vehículo a otro carril (simulación 1 y 3). El segundo caso consiste en el envío periódico de mensajes para mantener informados a los nodos. En ambos casos, los nodos deben retransmitir el paquete una vez. Para el segundo

caso, dado que se reciben múltiples mensajes, es necesario contar con un buffer para llevar un registro de los mensajes ya recibidos.

Para controlar el nodo, se utiliza la condición if (`getParentModule()->getIndex() == 0`) para determinar las acciones a realizar. Con `traciVehicle` se puede controlar el vehículo, por ejemplo, cambiando la velocidad (`traciVehicle->setSpeed(0)`) o la ruta. Para programar el envío de un mensaje, se utiliza un módulo que representa el mensaje junto con su contenido. Se define el tamaño del mensaje, se llenan sus variables, se crea el paquete y se le agrega el payload. Finalmente, con `sendPacket` se envía a todos los nodos.

```
auto payload = makeShared<VeinsInetSampleMessage>();
payload->setChunkLength(B(2000));
payload->setRoadId(traciVehicle->getRoadId().c_str());
timestampPayload(payload);
auto packet = createPacket("ChangingLane");
packet->insertAtBack(payload);
sendPacket(std::move(packet));
```

El código anterior es para un vehículo que cambiará de carril, crea un mensaje de 2kB de payload y avisa el Id de la nueva ruta, se envía con `sendPacket`. Por último, se requiere de una función callback para definir cuándo ocurren los eventos:

```
timerManager.create(veins::TimerSpecification(callback).oneshotAt(SimTime(70150, SIMTIME_MS)));
```

En `processPacket` los nodos reciben un paquete enviado por broadcast, con `peekAtFront` se leen las cabeceras del mensaje:

```
auto header = pk->peekAtFront<VeinsInetSampleMessage>();
```

Se puede extraer la información del mensaje y según el contenido realizar alguna acción, ej. (payload->getRoadId()), igual con traciVehicle se controla el comportamiento del nodo que recibió el mensaje.

Los nodos tienen una variable para saber si ya han retransmitido un mensaje (haveForwarded), pero solo funciona con un mensaje. Para retransmitir, se crea un nuevo paquete:

```
auto packet = createPacket("relay");
packet->insertAtBack(payload);
sendPacket(std::move(packet));
haveForwarded = true;
```

Para el caso del envío periódico de mensajes, se agregan variables donde se guardan el identificador de los últimos mensajes recibidos, para saber si ya lo había retransmitido, además pasado cierto tiempo se deben retirar estos mensajes por su antigüedad.

En la figura 8-9 se muestra la simulación final, para ejecutarla debe estar corriendo VEINS (se abre la aplicación veins\_launchd) y se lanza el archivo omnetpp.ini.



FIGURA 8-9 IMAGEN DE LA SIMULACIÓN CORRIENDO EN OMNET++, SE OBSERVA UN NODO RETRANSMITIENDO A TODOS UN MENSAJE

## **9 RESULTADOS DE LA RED VANET**

### **9.1 Resultados simulación 1**

Para la primera simulación se experimenta entonces los resultados del protocolo AODV al intentar mandar un mensaje ping entre dos nodos, se mide el porcentaje de mensajes recibidos y, el tiempo de ida y vuelta de un mensaje. Entre los parámetros principales para ver su desempeño, se varía la velocidad de los nodos, y el intervalo de transmisión entre mensajes.

En el primer escenario, que tiene una duración de 12 segundos, el nodo fuente intenta enviar 12 mensajes, pero solo recibe respuesta en promedio de 7.76, lo que representa una tasa de pérdida del 35.28%, con un RTT promedio de 735 ms.

En la figura 9-1 se pueden observar los mensajes del protocolo AODV enviados por cada nodo, que corresponden al descubrimiento de rutas e informes de errores cuando se rompe un enlace. El nodo fuente intenta enviar 12 mensajes, con un intervalo entre ellos de 1 segundo, por lo que siempre se tienen que descubrir las rutas antes. Para ello, cada nodo envía entre 7 y 15 mensajes durante la duración de la simulación. En la misma figura, se muestran los mensajes recibidos por cada nodo. Los mensajes ping, representados en naranja, son los transmitidos correctamente (en promedio 7.76). La cantidad de mensajes recibidos depende de la posición del nodo, pero los nodos cercanos reciben y procesan alrededor de 80 mensajes cada uno.

Todos estos mensajes transmitiéndose en la red tienen dos impactos importantes. El primero es el aumento de errores en los paquetes que se da por baja potencia de transmisión, colisiones de los mensajes (cuando dos nodos transmiten simultáneamente) y errores de bits. El segundo, son los retrasos para transmitir, pues un nodo debe esperar a que termine el anterior. En las gráficas observamos la tasa de bits de errores y la tasa de paquetes con error (son los paquetes recibidos con uno o más bits con errores), donde como se esperaría, los nodos fuente y algunos nodos que forman la trayectoria al nodo destino son los que más errores tienen.

En el caso de los tiempos de procesamiento (figura 9-2), OMNeT++ lo divide en dos “pending queue” e “in progress frame”. El primero considera desde la necesidad de mandar un paquete y esperar a que el canal esté libre, en la figura, el nodo fuente tarda 1.3 ms. El segundo valor cuenta después de crear el paquete hasta que se transmite en el medio (junto a los tiempos de contención), que para el nodo fuente es 1 ms.

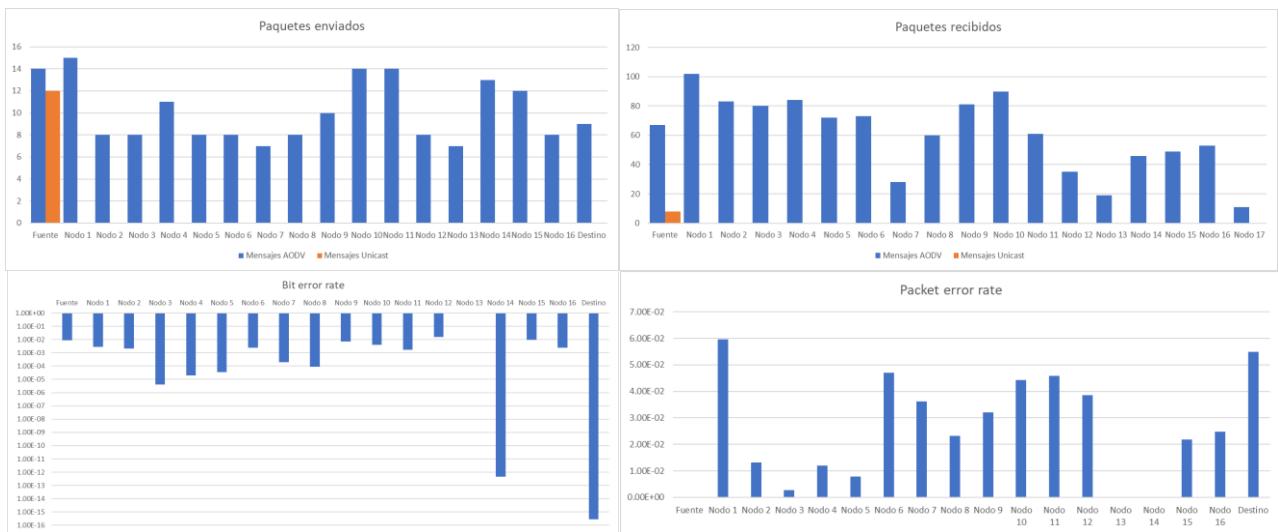


FIGURA 9-1 RESULTADOS DEL ESCENARIO 1

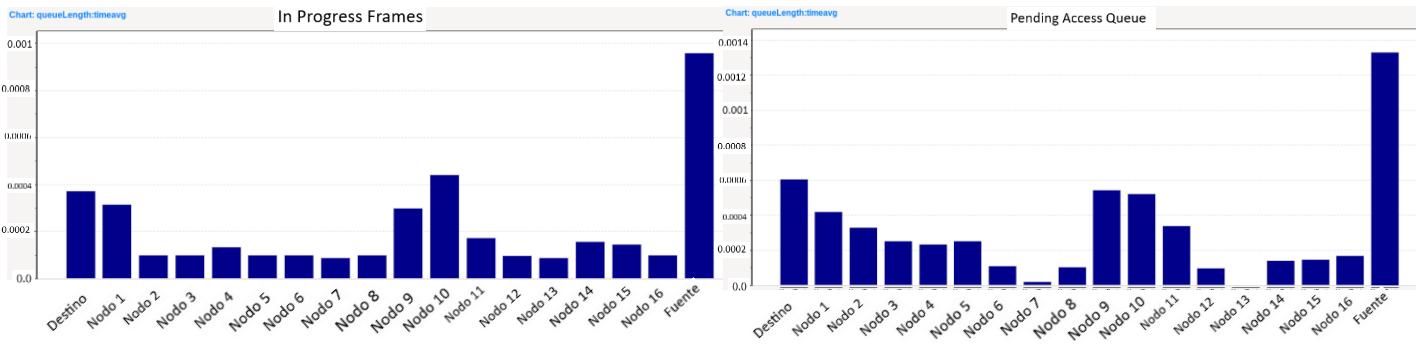


FIGURA 9-2 A LA IZQUIERDA LOS TIEMPOS QUE DEMORA "IN PROGRESS FRAMES" A LA DERECHA "PENDING QUEUE"

En la figura 9-3 se muestra visualmente la simulación.



FIGURA 9-3 RUTAS FORMADAS POR EL PROTOCOLO AODV

A partir del primer caso analizamos y comparamos con las demás simulaciones, al variar la velocidad y tasa de transmisión. Otros parámetros que se pueden variar y experimentar, pero quedan fuera del trabajo, es por ejemplo con la potencia de transmisión o sensibilidad del receptor, lo que significa un menor radio de cobertura. Por una parte, esto puede ser conveniente ya que, para alertar correctamente de un accidente no necesitamos de un radio de cobertura de 1 km, con 100 o 200 metros debe ser suficiente. Pero, esto también significa que los mensajes requieren más saltos para llegar al objetivo y las rutas son más fáciles de romperse, lo que puede llevar a menos mensajes recibidos y una mayor inundación de la red.

### 9.1.1 Diferentes velocidades e intervalos

TABLA 9-1 RTT Y PORCENTAJE DE PÉRDIDAS DE MENSAJES, AL VARIAR LA VELOCIDAD E INTERVALO

Velocidad de los nodos	Intervalo entre mensajes	Mensajes transmitidos	Mensajes recibidos	Porcentaje de pérdidas	RTT (ms)	Probabilidad de recibir 2 mensajes en 100 metros	Probabilidad de recibir 4 mensajes en 100 metros
30 km/h	1s	12	7.76	35.28%	734.7	99.94%	97.28%
30 km/h	0.5 s	24	15.56	35.1%	831.8	100%	100%
30 km/h	0.2 s	60	32.73	45.44%	721.8	100%	100%
30 km/h	0.1 s	120	67.43	43.81%	2031.1	100%	100%
50 km/h	1 s	8	4.17	47.92%	854.7	83.74%	21.46%
50 km/h	0.5 s	16	9.2	42.5%	903.8	99.72%	92.42%
50 km/h	0.2 s	40	21.87	45.33%	1053.5	100%	100%
50 km/h	0.1 s	80	40.5	49.37%	1374.2	100%	100%
80 km/h	1 s	6	3.4	43.33%	343.3	32.11%	0%
80 km/h	0.5 s	11	5.73	47.88%	333.4	83.79%	21.51%
80 km/h	0.2 s	27	15.73	41.73%	285.8	99.95%	97.92%
80 km/h	0.1 s	55	10.97	80.06%	399.7	97.22%	76.38%

En la tabla 9-1 se presenta cómo se modifica tanto la tasa de mensajes recibidos, como el RTT promedio al variar la velocidad y el intervalo entre mensajes. Con AODV se había configurado que las rutas se mantuvieran activas 0.5 segundos, donde este contador se

reinicia cada que se recibe un nuevo mensaje. Aun así, con un intervalo entre mensajes menor a 0.5 segundos, no se garantiza que no se deban buscar nuevas rutas ante cambios en la topología. Esto puede significar un aumento grande de mensajes en la red, especialmente a altas velocidades. Igualmente, un mal intervalo de transmisión, en el que se requiera buscar nuevas rutas constantemente, lleva a un aumento en el número de colisiones y errores en paquetes, aumenta el tiempo de acceso al medio, lo que lleva a que el RTT lo haga también.

Al observar el porcentaje de pérdidas como el RTT, en general los intervalos funcionan a excepción del de 0.1 segundos, cada uno con ciertas ventajas según las condiciones. Los intervalos con los que se tiene menor porcentaje de pérdidas y más bajos RTT son, a 30 km/h un intervalo entre 1 y 0.5 segundos, a 50 km/h el intervalo de 0.5 y 0.2 s, y a 80 km/h el de 0.2 s. Sin embargo, la situación cambia al buscar transmitir cierta cantidad de mensajes durante un recorrido, pues un intervalo más bajo son más mensajes enviados y mayor oportunidad de éxito.

El protocolo AODV muestra un buen comportamiento incluso con velocidades altas, los cambios en la topología de la red no son lo suficientemente significativos como para requerir actualizaciones constantes de las rutas, aunque tiene mayores pérdidas, logra mejor RTT. Esto resulta en una reducción de la saturación de mensajes en la red, ya que se evita la necesidad de enviar mensajes de estado de AODV de manera frecuente. Si bien es cierto que aumentar la tasa de envío de mensajes conlleva a una mayor tasa de pérdida de paquetes, la red cuenta con suficiente redundancia para mantener a los nodos informados. Sin embargo,

es importante controlar la inundación de mensajes al incrementar el número de nodos comunicándose simultáneamente, para evitar introducir retrasos significativos en el acceso al medio, como se comprueba en la simulación 2.

En las figuras 9-4 y 9-5, se muestra el impacto que tiene la velocidad de los vehículos, sobre la probabilidad de recibir 2 y 4 mensajes respectivamente, sobre un recorrido de 100 metros.

En el caso de 4 mensajes, haría que fuera totalmente insuficiente el intervalo para transmitir de 1 segundo. Mientras que un intervalo de 0.2 segundos parece ser valido en todo momento, 1 y 0.5 segundos dejan de ser suficientes para comunicar cierta cantidad de mensajes al aumentar la velocidad. En el caso de 0.1 s como se vio anteriormente, tiene RTT muy altos que lo hacen inviable, y en general peores resultados que el intervalo de 0.2.

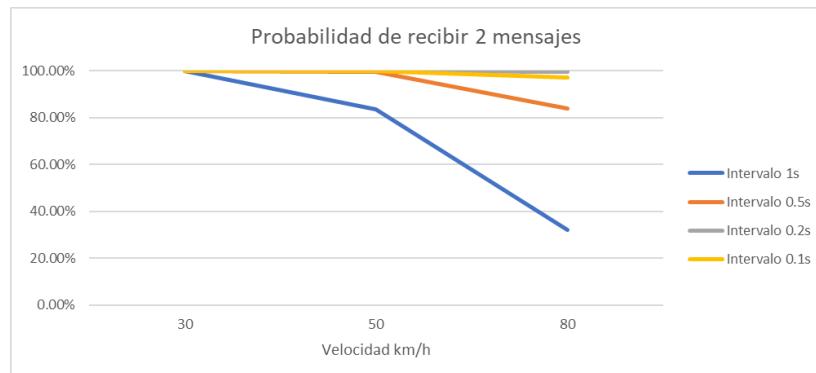


FIGURA 9-4 PROBABILIDAD DE RECIBIR 2 MENSAJES CON AODV

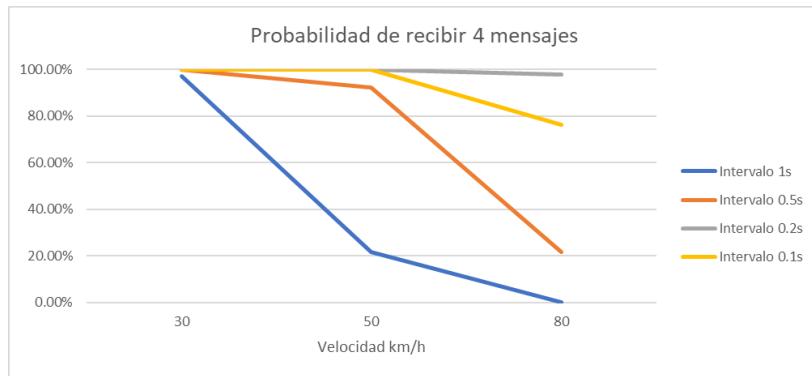


FIGURA 9-5 PROBABILIDAD DE RECIBIR 4 MENSAJES CON AODV

Con las gráficas anteriores, parecería que el intervalo óptimo es 0.2 segundos, pero esta situación cambiará cuando todos los nodos se comuniquen simultáneamente. En general, se podría decir que al recibir dos mensajes a 30 km/h, intervalos de 1 a 0.2 segundos pueden funcionar. Para velocidades de 50 km/h, los intervalos ideales serían de 0.5 y 0.2 segundos. En cambio, a 80 km/h, solo un intervalo de 0.2 segundos tiene una alta probabilidad de éxito. Con la recepción de cuatro mensajes, a 30 km/h, un intervalo de 1 segundo ya no es ideal, y a 50 km/h, 0.5 segundos también deja de serlo.

Aun en ausencia de comunicaciones con otros nodos, se observan limitaciones significativas con intervalos de transmisión muy bajos, como 0.1 segundos. El RTT obtenido a 30 y 50 km/h es muy alto, lo que lo hace inviable, y en velocidades altas la cantidad de mensajes recibidos con respecto a otros intervalos empeora significativamente. Este fenómeno se debe a que, con una baja densidad de nodos en la red, los enlaces se rompen con frecuencia y se necesita encontrar nuevas rutas.

Además, se debe considerar el tiempo necesario para preparar los mensajes, que en con los resultados de la primera parte del trabajo, superaba el intervalo de 0.1 segundos, lo que introduce retrasos adicionales junto con el encolamiento de mensajes. En el caso del nodo de detección del vehículo, ni siquiera se logra alcanzar una frecuencia de 10 Hz debido a estas limitaciones.

Los resultados obtenidos sugieren la conveniencia de utilizar un intervalo adaptativo entre el envío de mensajes, el cual se ajuste según la densidad y la velocidad del tráfico. En escenarios con una baja densidad de nodos, no es viable mantener intervalos tan cortos como 100 milisegundos entre mensajes, e intervalos altos pueden no ser suficientes para transmitir el mensaje a tiempo. Por otro lado, aunque es posible mantener intervalos cortos a altas velocidades, se observa un aumento en el tiempo de ida y vuelta (RTT) y en el número de mensajes necesarios para mantener las rutas. A medida que se añaden más nodos transmitiendo y comunicándose, se produce un incremento en las colisiones y en los tiempos de espera en las colas de transmisión. Esto hace necesario reducir la frecuencia con la que los nodos actualizan sus estados, incluso sin tener en cuenta los tiempos de procesamiento de los mensajes.

A continuación, se muestra el desempeño de DSRC con la presencia de más nodos. El mismo comportamiento debería poder trasladarse a AODV cuando más nodos se comunican entre sí.

## 9.2 Resultados simulación 2

Para la segunda simulación se experimenta ahora con el protocolo de DSRC y todos los nodos intentan comunicarse entre sí para compartir sus detecciones. Para mostrar el funcionamiento y la idea de cómo se aplicaría, primero se hizo un ejemplo práctico en el que el nodo 6 llega a una intersección y quiere incorporarse a los carriles principales, la secuencia se observa de las figuras 9-6 a la 9-9.



FIGURA 9-6 EL NODO 6 EN ROJO QUIERE TRANSMITIR



FIGURA 9-7 EL NODO 6 TRANSMITE EL MENSAJE, Y LOS DEMÁS NODOS AL RECIBIRLO RETRANSMITEN UNA VEZ, EL NODO 7 Y 4 NO SON VISTOS POR EL NODO 6, CON LAS RETRANSMISIÓNES QUEDAN INFORMADOS.



FIGURA 9-8 LOS NODOS CERCANOS 5, 11 Y 7 BAJAN SU VELOCIDAD Y PERMITEN QUE EL NODO 6 SE INCOPORE, EL NODO 4 NO BAJA SU VELOCIDAD



FIGURA 9-9 POSICIÓN FINAL, EL NODO 6 SE INCORPORA

Como en la simulación 1, partimos del primer escenario para comparar, en este, todos transmiten sus detecciones cada segundo. Los vehículos se desplazan a 30 km/h en una simulación para recorrer 100 metros, igual que en AODV. En la figura 9-10 se pueden observar los mensajes enviados y recibidos por cada nodo y en la figura 9-11 los tiempos de encolamiento, teniendo en cuenta tanto las transmisiones como las retransmisiones, para el caso de un intervalo de 1 segundo y una velocidad de 30 km/h.

El intervalo entre transmisiones de 0.1 segundos no se experimentó. Este intervalo no es viable (además de presentar problemas computacionales), pues ocasiona muchas colisiones y retransmisiones de mensajes con lo que los tiempos de encolamiento aumentan bastante. Además, considerando que la red apenas está compuesta de 10 nodos, se observará que, con

el aumento del número de participantes, intervalos para comunicarse como de 0.2 segundos también presenten problemas.

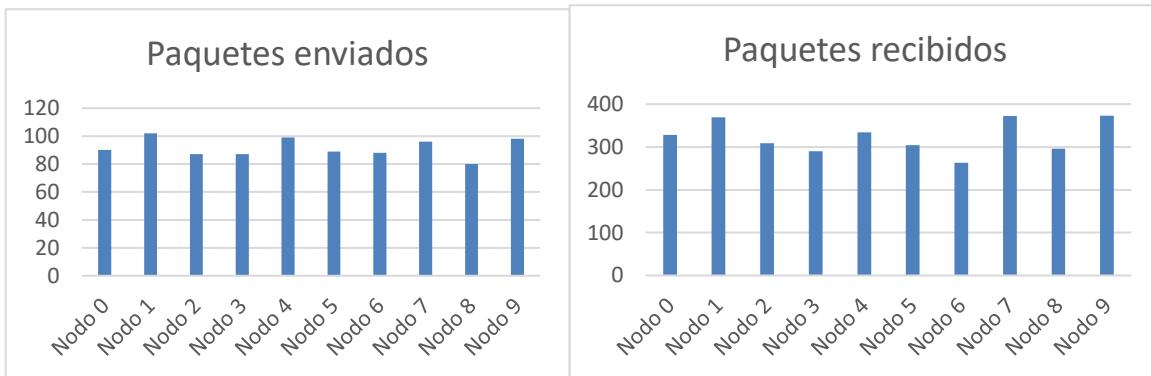


FIGURA 9-10 PAQUETES ENVIADOS (INCLUYE RETRANSMISIONES) Y PAQUETES RECIBIDOS POR CADA NODO

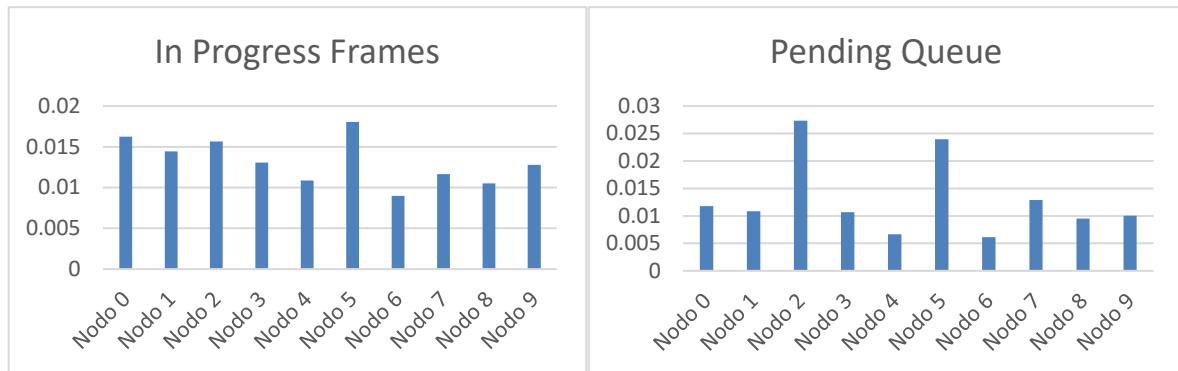


FIGURA 9-11 TIEMPOS DE ESPERA PROMEDIO DE CADA NODO

Como en la simulación 1, los tiempos están divididos en “pending queue” e “in progress frames”, estando en promedio para acceder al medio en 15 ms, y para transmitir 30 ms. Ya desde el primer caso, que sería el mejor con una velocidad de 30 km/h y un intervalo grande entre transmisiones, los nodos dejan de recibir muchos mensajes. Si bien no todos los nodos se alcanzan entre sí durante todo el tiempo de la simulación, la mayoría se debe a colisiones

a la hora de transmitir o retransmitir. En la tabla 9-2 se resumen los resultados, para el caso de transmitir un mensaje de 1 kB de Payload.

**TABLA 9-2 MENSAJES RECIBIDOS EN PROMEDIO DE CADA NODO,  
RETRANSMISIONES EN PROMEDIO SON EL NÚMERO DE MENSAJES ÚNICOS QUE  
RECIBEN**

Velocidad	Intervalo de transmisión	Duración de la simulación	Mensajes programados para el envío	Retransmisiones en promedio	Mensajes enviados por nodo en promedio	Mensajes recibidos promedio
30 km/h	1s	12 s	12	64.71	76.71	245.39
30 km/h	0.5 s	12 s	25	122.99	146.99	461.82
30 km/h	0.2 s	12 s	61	770	364.71	1142.63
50 km/h	1 s	8 s	8	54.87	62.87	268.87
50 km/h	0.5 s	8 s	17	102.83	119.83	510.21
50 km/h	0.2 s	8 s	41	257.69	298.69	1275.45
80 km/h	1 s	5 s	6	37.86	42.86	195.31
80 km/h	0.5 s	5 s	11	68.55	79.55	366.19
80 km/h	0.2 s	5 s	26	174.06	200.06	936.1

De la tabla, son solo los mensajes programados para enviar, por retrasos para obtener acceso al medio no se envían todos. Como se esperaría, al reducir el intervalo de transmisión, se tiene un pequeño aumento en los tiempos de encolamiento, junto a un aumento en las colisiones. Esto lleva a que los mensajes recibidos aumenten en menor proporción al de nuevos mensajes transmitidos. Con el aumento de velocidad, los tiempos de encolamiento se

modifican poco. Parte importante de los mensajes perdidos, no solo se dan por las colisiones, también por el aumento en la velocidad de los nodos.

En el intervalo de 0.2 segundos el acceso al medio comienza a ser problemático, lo que lleva a que varios mensajes se retrasasen. Al mismo tiempo, la inundación de la red es considerable debido a que los nodos reciben muchos más mensajes que deben procesar.

Como se había adelantado en la simulación 1, tener intervalos bajos de actualización, en este caso para nosotros es informar de las detecciones de otros vehículos, dependerá como se había visto de la velocidad, pero también de la densidad de vehículos que intenten comunicarse. Con diez nodos, un intervalo de comunicación de 200 ms parece ser solo funcional a 30 km/h. A pesar de que a altas velocidades tiene mayor porcentaje que otros intervalos, los tiempos de encolamiento son perjudiciales y superan su propio intervalo de 200 ms. En este punto, eso implica descartar el mensaje porque el siguiente ya fue generado. En el caso de intervalos de 0.5 y 1s funcionan, pero su dificultad se encuentra en la poca cantidad de mensajes que pueden enviar en distancias cortas. Con estos dos intervalos, la inundación de la red con mensajes no parece ser tan perjudicial, ni siquiera al aumentar la velocidad en los nodos.

### **9.2.1 Diferentes densidades de nodos**

Los resultados anteriores corresponden al caso de 10 nodos en la red con un tamaño de paquetes de 1kB. Los siguientes escenarios muestran el comportamiento al variar la cantidad de nodos y el tamaño de los mensajes.

## 10 nodos

La tabla 9-3 muestra los resultados con 10 nodos, vehículos, en la red compartiendo sus detecciones variando el tamaño del paquete, la velocidad y el intervalo de transmisión.

**TABLA 9-3 PROBABILIDAD DE RECIBIR CIERTO NÚMERO DE MENSAJES Y LOS TIEMPOS DE ESPERA PROMEDIO, AL VARIAR EL TAMAÑO DEL MENSAJE, VELOCIDAD E INTERVALO. EN UNA RED CON 10 NODOS**

Tamaño Payload kB	Intervalo de transmisión (segundos)	Velocidad km/h	Porcentaje de mensajes recibidos	Probabilidad de recibir 2 mensajes en 100 m	Probabilidad de recibir 4 mensajes en 100 m	Tiempo promedio de encolamiento (ms)
1	1	30	53.93%	99.45%	88.51%	22.85
1	0.5	30	48.80%	100.00%	99.82%	41.25
1	0.2	30	49.79%	100.00%	100.00%	111.33
1	1	50	68.59%	96.36%	49.94%	56.59
1	0.5	50	60.49%	99.85%	94.88%	110.09
1	0.2	50	62.85%	100.00%	100.00%	300.81
1	1	80	61.44%	37.75%	0%	67.87
1	0.5	80	62.32%	92.96%	37.82%	134.75
1	0.2	80	66.95%	100.00%	99.64%	393.36
				Probabilidad de recibir 1 mensaje en 100 m	Probabilidad de recibir 2 mensajes en 100 m	
2	1	30	42.45%	99.60%	96.66%	46.72
2	0.5	30	38.14%	99.99%	99.91%	92.76

2	0.2	30	39.09%	100.00%	100.00%	241.0
2	1	50	46.23%	95.51%	76.18%	102.29
2	0.5	50	46.19%	99.80%	98.05%	174.6
2	0.2	50	48.97%	100.00%	100.00%	500.0
2	1	80	46.18%	71.03%	21.33%	103.29
2	0.5	80	48.34%	96.32%	79.11%	211.08
2	0.2	80	54.67%	99.99%	99.88%	647.36

En la figura 9-12 se grafican los tiempos de encolamiento, como se observa, estos experimentan ligeros cambios al aumentar la velocidad, siendo el mayor cambio con los intervalos pequeños como 0.2 segundos. Igualmente, como se esperaba, los tiempos son mayores con tamaños de mensajes más grandes.

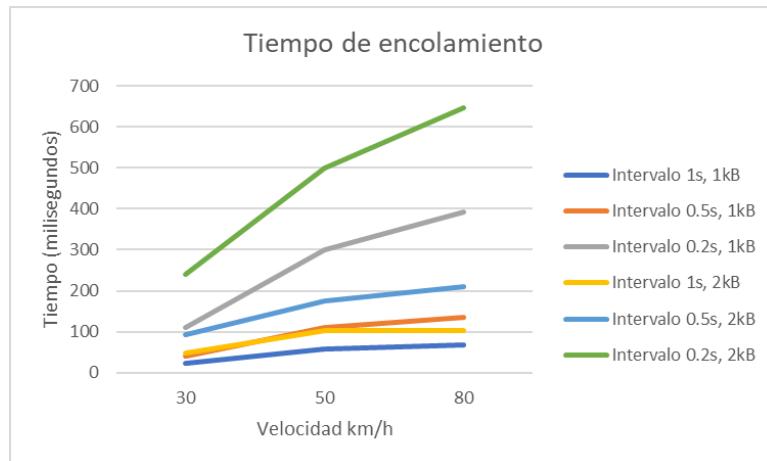


FIGURA 9-22 TIEMPOS DE ENCOLAMIENTO

Con una distancia de 100 metros antes de la intersección, la gráfica 9-13 muestran la probabilidad de que un nodo quede enterado de la presencia del otro, según dos diferentes

casos. En la izquierda que reciban 2kB de mensajes y en la derecha que deban recibir 4kB, siendo más difícil este último.

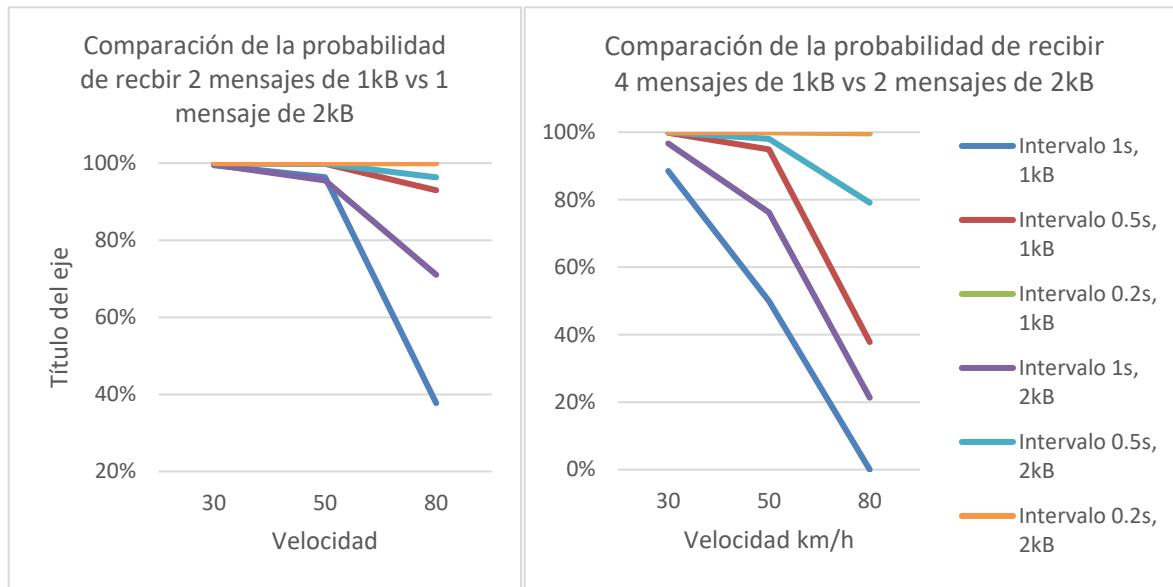


FIGURA 9-33 PROBABILIDAD DE RECIBIR 2 Y 4 MENSAJES CON 10 NODOS, LOS DOS INTERVALOS DE 0.2 SEGUNDOS ESTÁN SUPERPUESTOS

Similar a la anterior, en la figura 9-14 se muestra la probabilidad de que un nodo quede enterado de la presencia del otro, pero ahora con 200 metros de anticipación.

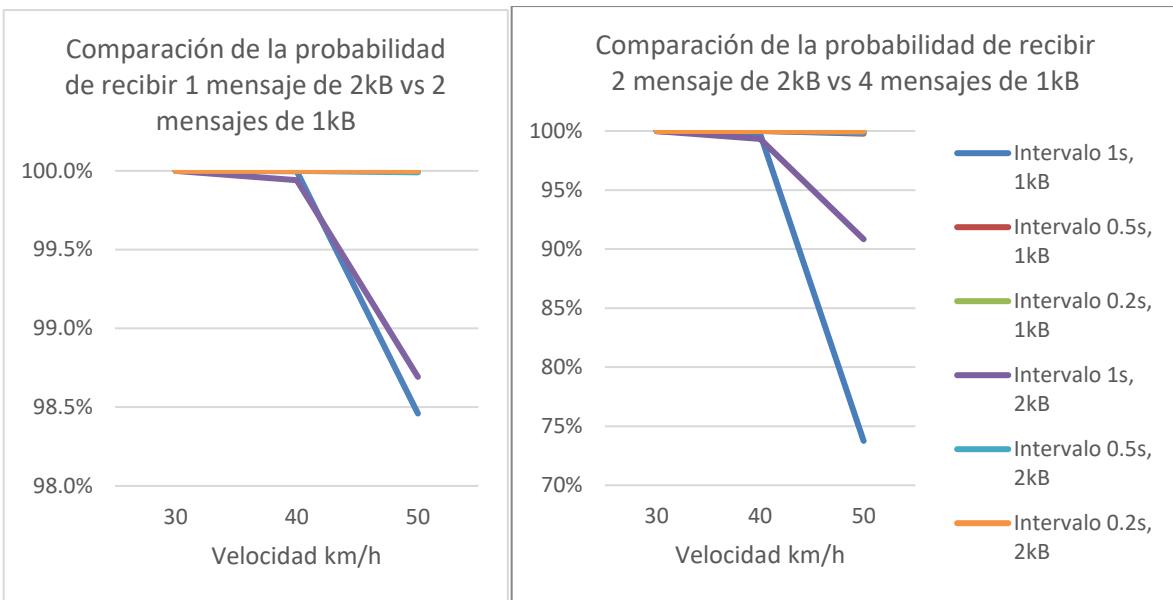


FIGURA 9-14 PROBABILIDAD DE RECIBIR 2 Y 4 MENSAJES CON 10 NODOS, EN UN TRAYECTO DE 200 METROS

A primera vista el intervalo de 0.2 segundos parece ser ideal, ya que logra conseguir la mayor probabilidad de recibir los mensajes. Sin embargo, la inundación de la red es considerable, y lleva un serio aumento en los tiempos de encolamiento. En cambio, a bajas velocidades, hay casos para los que un intervalo bajo como de 1 segundo es suficiente para la comunicación.

Como se observa en los resultados, el impacto de la velocidad en los vehículos se percibe más en forma de retrasos para acceder al medio, que en errores en los mensajes. Esto se debe a que los protocolos están diseñados para manejar la comunicación, sin importar la velocidad. En cambio, la gráfica de porcentaje de mensajes recibidos muestra comportamientos interesantes que no deben tomarse como consecuencia directa de la velocidad, sino de los cambios en la topología. Que dependerá según los nodos con los que se encuentren en rango

y la velocidad con la que varía, esto lleva a que aumente o se reduzca la probabilidad de que ocurran colisiones en los mensajes, mejorando o no el porcentaje de mensajes recibidos.

La velocidad juega otro papel importante en la cantidad de mensajes que se pueden enviar a lo largo de cierta distancia. Con 50 km/h, todavía se puede encontrar un buen intervalo y tamaño de mensaje para lograr comunicarse con una alta probabilidad, cosa que a 80 km/h ya no se puede garantizar.

## 20 nodos

**TABLA 9-4 COMPARACIÓN DE LOS TIEMPOS DE ENCOLAMIENTO Y PROBABILIDAD DE RECIBIR CIERTO NÚMERO DE MENSAJES, EN UNA RED CON 20 VEHÍCULOS**

Tamaño Payload kB	Intervalo de transmisión (segundos)	Velocidad km/h	Porcentaje de mensajes recibidos	Probabilidad de recibir 2 mensajes en 100 m	Probabilidad de recibir 4 mensajes en 100 m	Tiempo promedio de encolamiento (milisegundos)
1	1	30	53.17%	99.37%	87.52%	81.69
1	0.5	30	47.23%	99.99%	99.72%	169.62
1	0.2	30	-----	-----	-----	431.38
1	1	50	60.01%	91.30%	33.71%	149.63
1	0.5	50	54.54%	99.51%	89.26%	312.60
1	0.2	50	-----	-----	-----	5982.1
1	1	80	54.98%	30.23%	0%	170.52
1	0.5	80	56.48%	88.31%	27.89%	401.19
1	0.2	80	-----	-----	-----	945.1

				Probabilidad de recibir 1 mensaje en 100 m	Probabilidad de recibir 2 mensajes en 100 m	
2	1	30	37.75%	99.13%	93.83%	118.53
2	0.5	30	34.73%	99.98%	99.77%	243.42
2	0.2	30	-----	-----	-----	4331
2	1	50	38.69%	91.34%	64.00%	156.79
2	0.5	50	34.78%	98.61%	91.18%	331.70
2	0.2	50	-----	-----	-----	1721
2	1	80	34.53%	57.14%	11.92%	192.43
2	0.5	80	35.53%	88.86%	58.17%	382.88
2	0.2	80	-----	-----	-----	1490

En la tabla 9-4 se muestran ahora los resultados cuando se tiene la presencia de 20 nodos en la red. El intervalo de 0.2 segundos se sigue mostrando únicamente para ejemplificar lo que sucede con los tiempos de encolamiento, que como se observa no son viables. Por lo que, para un caso con una red con 20 nodos, solo se deberían considerar intervalos de transmisión de 0.5 y 1 segundos, como se observa en la figura 9-15, cuyos tiempos de encolamiento caen en un rango aceptable.

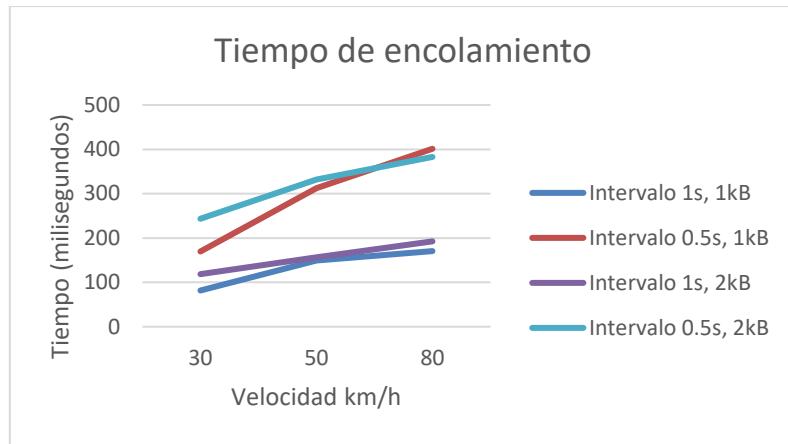


FIGURA 9-15 TIEMPOS DE ENCOLAMIENTO CON 20 NODOS

La figura 9-16 muestra cómo cambia la probabilidad de recibir una determinada cantidad de mensajes, cuando se tiene una distancia a la intersección de 100 metros.

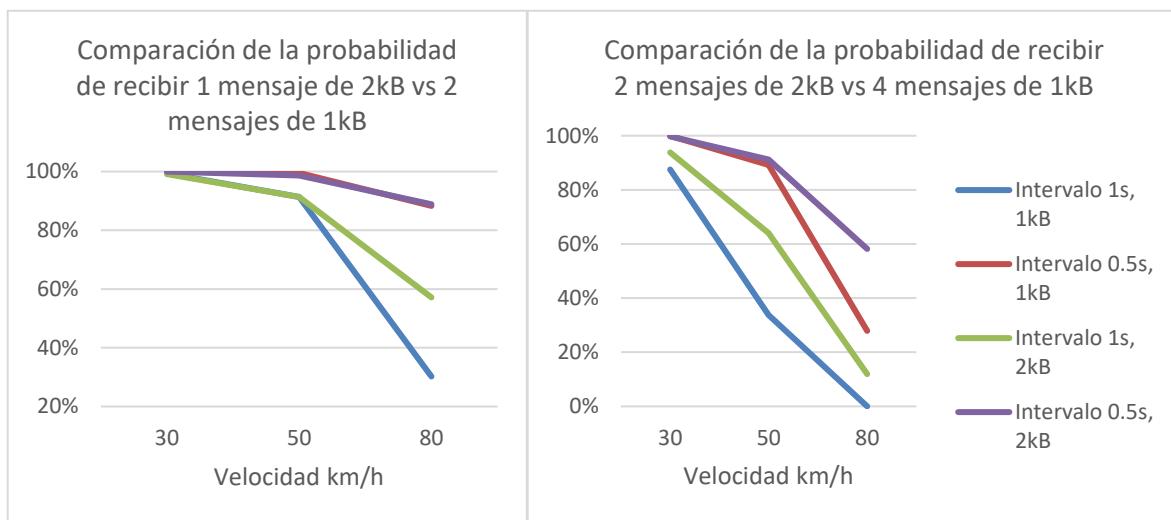


FIGURA 9-16 PROBABILIDAD DE RECIBIR 2 Y 4 MENSAJES CON 20 NODOS

La figura 9-17 muestra el mismo caso, pero ahora con una distancia de 200 metros a la intersección.

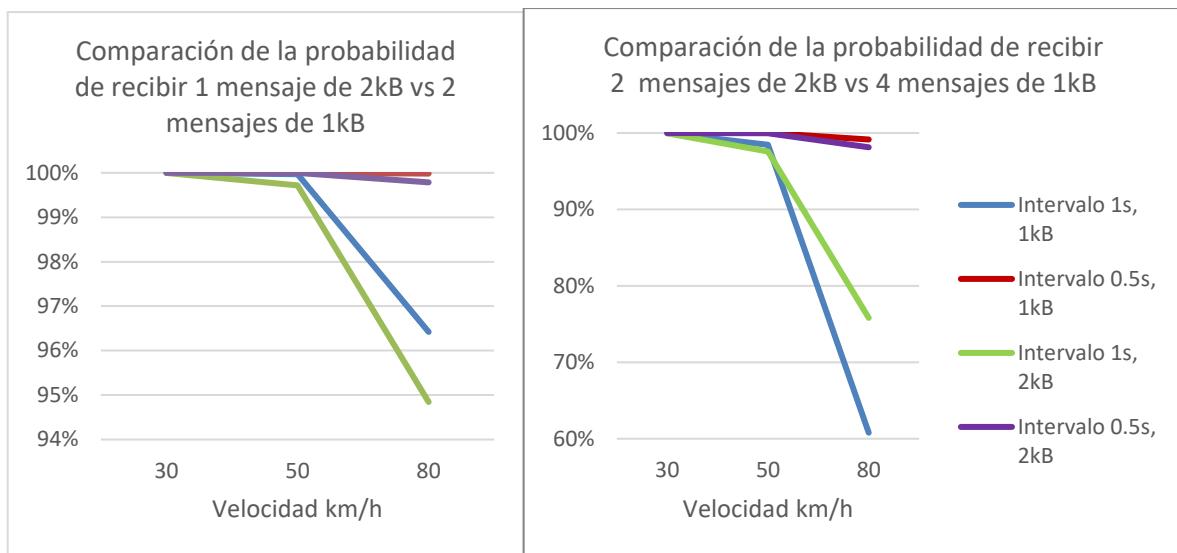


FIGURA 9-17 PROBABILIDAD DE RECIBIR 2 Y 4 MENSAJES CON 20 NODOS, EN UN TRAYECTO DE 200 METROS

Comienzan a mostrarse algunos retrasos con el intervalo de 0.5 segundos y con un tamaño de mensaje de 2 kB.

### 30 nodos

Para el último caso, la tabla 9-5 muestra los resultados con 30 nodos en la red. Como en el anterior, el intervalo de 0.2 segundos solo se incluye para mostrar como crece el tiempo promedio de encolamiento. La figura 9-18 grafica el tiempo de encolamiento promedio para los intervalos de 0.5 y 1 segundos, como en los anteriores, sigue la misma tendencia donde crecen con la velocidad.

**TABLA 9-5 COMPARACIÓN DE LOS TIEMPOS DE ENCOLAMIENTO Y PROBABILIDAD DE RECIBIR CIERTO NÚMERO DE MENSAJES, EN UNA RED CON 30 VEHÍCULOS**

Tamaño Payload kB	Intervalo de transmisión (segundos)	Velocidad km/h	Porcentaje de mensajes recibidos	Probabilidad de recibir 2 mensajes en 100 m	Probabilidad de recibir 4 mensajes en 100 m	Tiempo promedio de encolamiento (milisegundos)
1	1	30	52.43%	99.29%	86.51%	221.48
1	0.5	30	46.46%	99.99%	99.66%	560.98
1	0.2	30	-----	-----	-----	11773
1	1	50	58.68%	90.24%	31.45%	219.38
1	0.5	50	53.73%	99.43%	88.25%	523.73
1	0.2	50	-----	-----	-----	13233
1	1	80	53.23%	28.33%	0%	278.16
1	0.5	80	54.83%	86.71%	25.36%	771.94
1	0.2	80	-----	-----	-----	15432
				Probabilidad de recibir 1 mensaje en 100 m	Probabilidad de recibir 2 mensajes en 100 m	
2	1	30	35.46%	98.75%	91.86%	204.11
2	0.5	30	32.25%	99.96%	99.56%	455.17
2	0.2	30	-----	-----	-----	13112
2	1	50	35.23%	88.60%	57.60%	195.51
2	0.5	50	31.25%	97.64%	86.92%	437.94
2	0.2	50	-----	-----	-----	88349

2	1	80	34.31%	56.85%	11.77%	254.20
2	0.5	80	34.65%	88.08%	56.48%	639.04
2	0.2	80	-----	-----	-----	15392

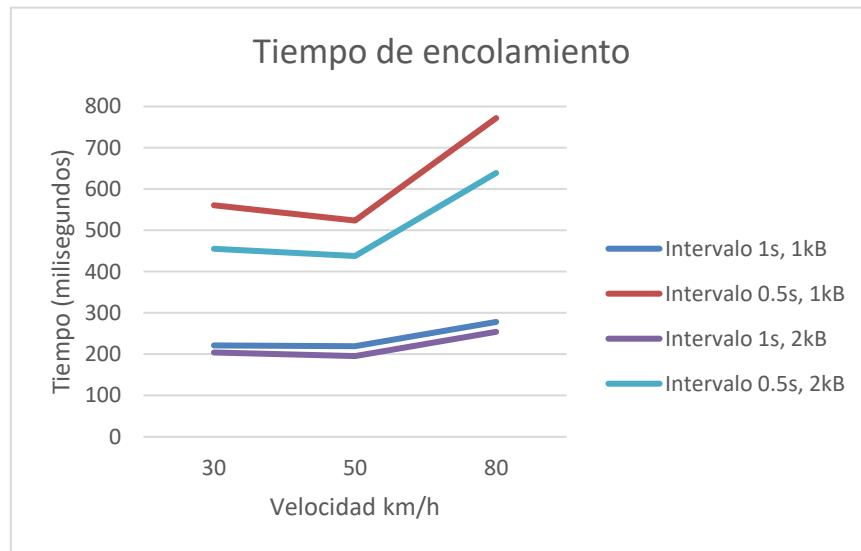


FIGURA 9-18 TIEMPOS DE ENCOLAMIENTO CON 30 NODOS

En la figura 9-19 se muestra su viabilidad si se requiriera que se comunicaran antes de 100 metros a la intersección.

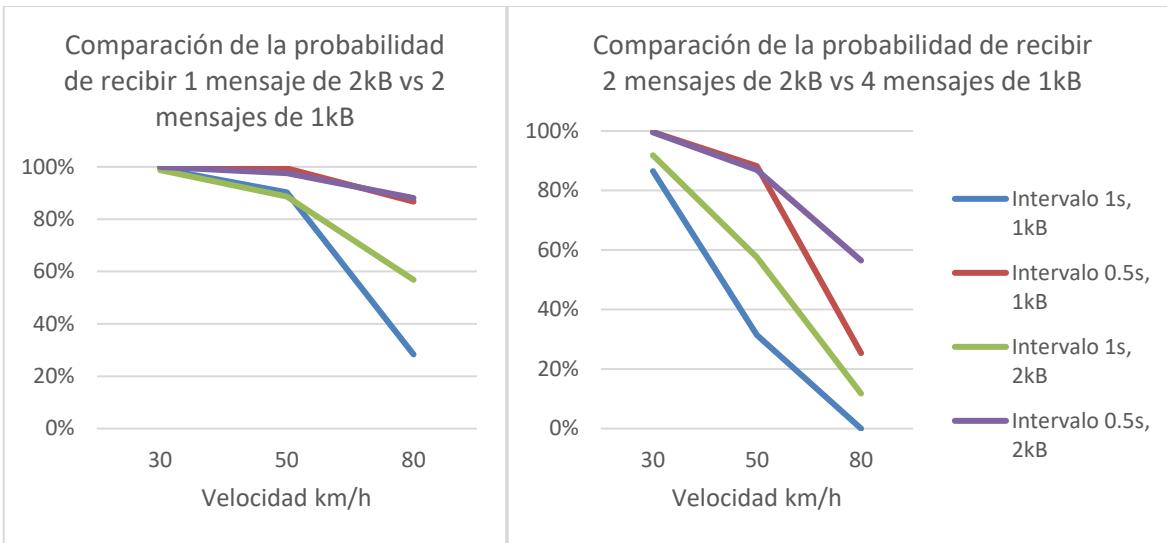


FIGURA 9-19 PROBABILIDAD DE RECIBIR 2 Y 4 MENSAJES CON 30 NODOS

Para el caso de una distancia de 200 metros a la intersección, los resultados se grafican en la figura 9-20.

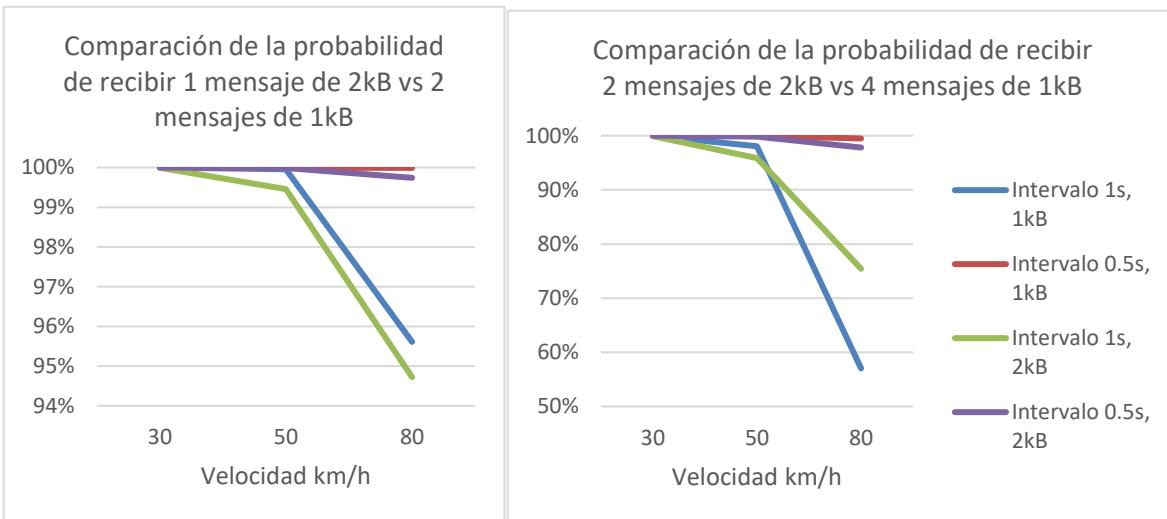


FIGURA 9-20 PROBABILIDAD DE RECIBIR 2 Y 4 MENSAJES CON 30 NODOS, EN UN TRAYECTO DE 200 METROS

Con 30 nodos, los tiempos de encolamiento para un intervalo de 0.5 segundos, empiezan a mostrar sus límites, el cual deja de ser viable para una velocidad de 80 km/h.

En la gráfica 9-21 se muestra el incremento en el tiempo de encolamiento al agregar más vehículos a la simulación, para una velocidad de 30 km/h. No se grafica para el intervalo de 0.2 segundos, porque a partir de 20 vehículos, los tiempos se disparan.

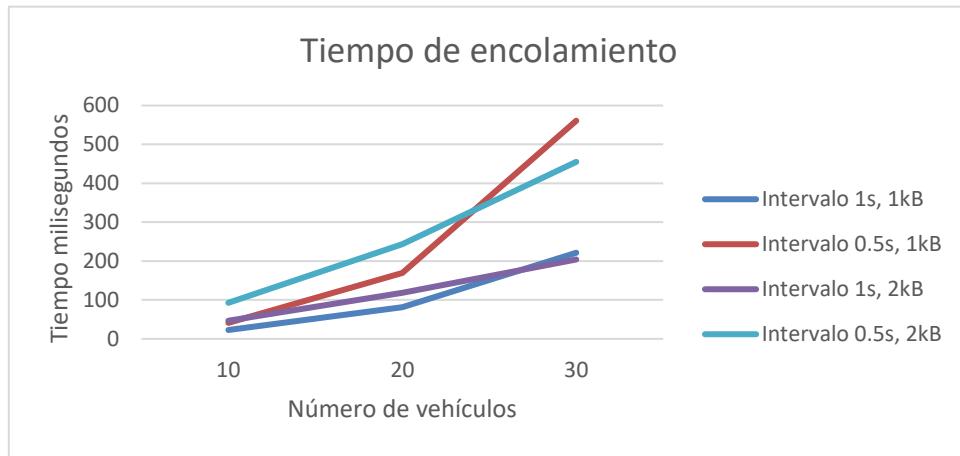


FIGURA 9-21 TIEMPOS DE ENCOLAMIENTO AL AUMENTAR EL NÚMERO DE NODOS

En el caso del porcentaje de mensajes recibidos, como se observa en la figura 9-22 para el caso de una velocidad a 50 km/h, si bien no se notaba un impacto realmente con el cambio de la velocidad, si se observa cómo empeora al aumentar el número de vehículos en la red.

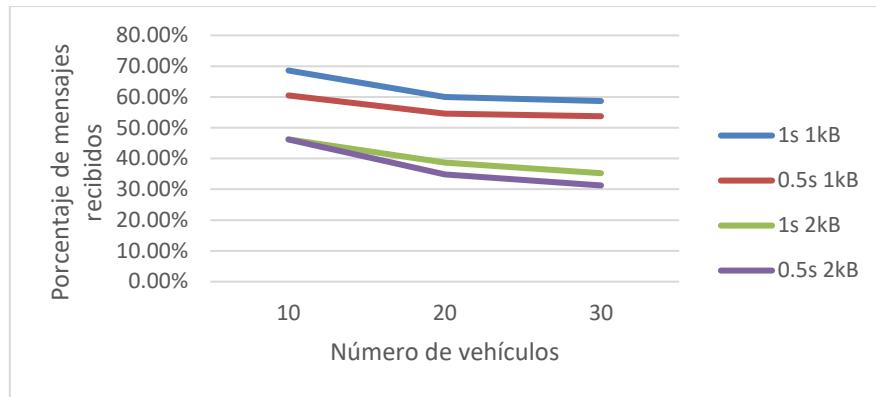


FIGURA 9-22 PORCENTAJE DE MENSAJES RECIBIDOS A 50 KM/H

El vehículo tiene información suficiente para modificar su intervalo de transmisión, conoce su velocidad, y la estimada de otros nodos. Además, puede darse una idea de la cantidad de nodos en la red, mediante los mensajes recibidos. Si se agrega que conoce su ubicación, puede conocer la localización por ejemplo de cruceros, y entonces considerar su distancia a ciertos puntos ciegos.

De las mediciones observadas se puede ver que, con 10 nodos, si el mensaje es de 2 kB, el intervalo de 0.2 segundos no es viable, mientras que con mensajes de 1 kB deja de serlo a partir de 50 km/h. Con 20 nodos, el intervalo de 0.2 segundos nunca es viable. Y con 30 nodos, el intervalo de 0.5 segundos con mensajes de 1 kB no es viable, y 0.5 s con mensajes de 2 kB deja de serlo a los 80 km/h. Además, a partir de 0.5 segundos y 20 nodos, con mensajes de 2 kB se puede presentar retrasos en algunos escenarios.

A continuación, en las figuras 9-23 a 9-25 se muestran los intervalos recomendados según donde se tenga mayor probabilidad de que los nodos queden notificados, con diferentes

velocidades. Se considera que deben recibir 4 mensajes de 1 kB o 2 mensajes de 2 kB. En las siguientes imágenes, el centro del círculo representa la intersección antes de la cual los vehículos deben comunicarse. El círculo naranja corresponde a un radio de 100 metros y el amarillo a uno de 200 metros. Se propone que los vehículos adapten la tasa de transmisión y el tamaño de los mensajes según las condiciones del tráfico y la distancia a la intersección (según el círculo en que se encuentren).

A una velocidad de 30 km/h:

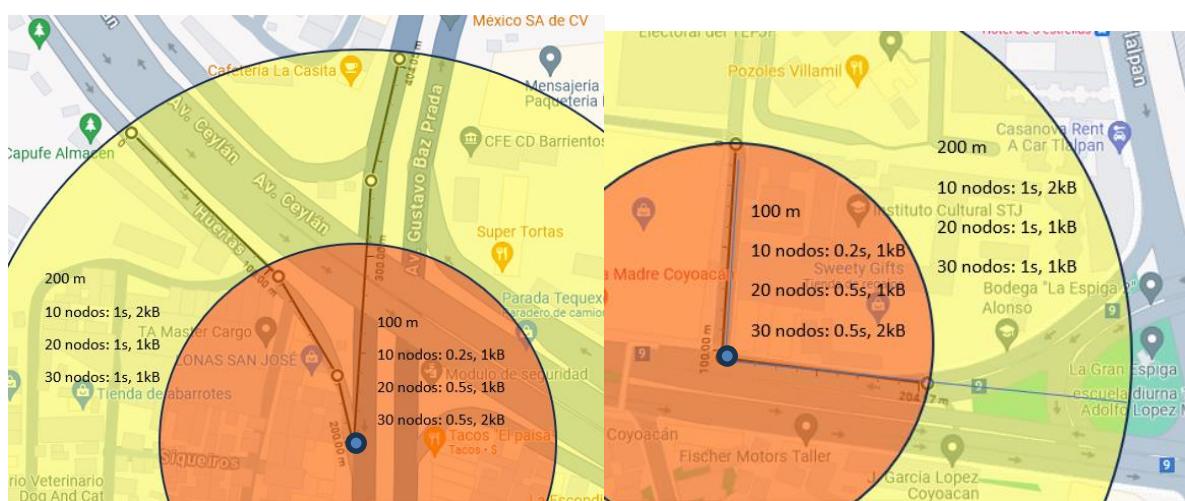


FIGURA 9-43 INTERVALOS DE TRANSMISIÓN RECOMENDADOS A UNA VELOCIDAD DE 30 KM/H

Los autos miden su velocidad y pueden estimar la densidad de nodos. Además, para el caso que se planea, se asume que los autos pueden conocer su ubicación y distancia a puntos conflictivos, como un crucero. Para que logren transmitir sus detecciones y que los vehículos queden notificados de la presencia de otros, con la información que tienen se espera que

puedan transmitirlas con alta probabilidad de recepción, variando el intervalo y el tamaño de los mensajes.

Por ejemplo, con 10 nodos, a 30 km/h y a 200 metros de un crucero, los nodos podrían mandar mensajes de 1 kB cada segundo, como se muestra en la figura 8-29. Conforme se acerque, debería aumentar la tasa de envío, a mandarlo cada 0.2 segundos. Para la elección de los intervalos, no se toma en cuenta ningún aspecto energético. Únicamente, se elige de acuerdo con cuál tienen la mayor probabilidad de que se reciban 4 mensajes, que produzca la menor inundación y por tanto tenga los menores tiempos de encolamiento posibles.

A 50 km/h:

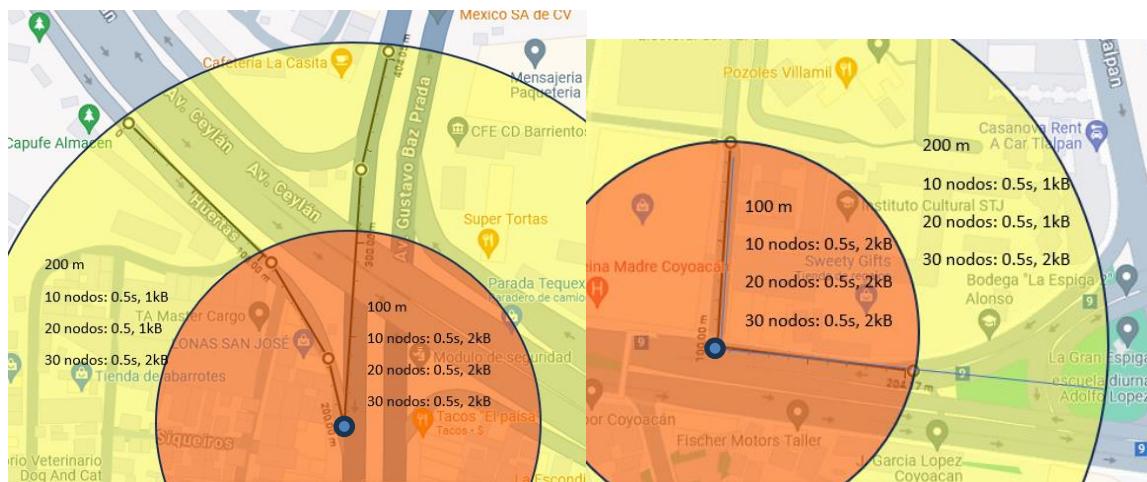


FIGURA 9-24 INTERVALOS DE TRANSMISIÓN RECOMENDADOS A 50 KM/H

A 80 km/h:

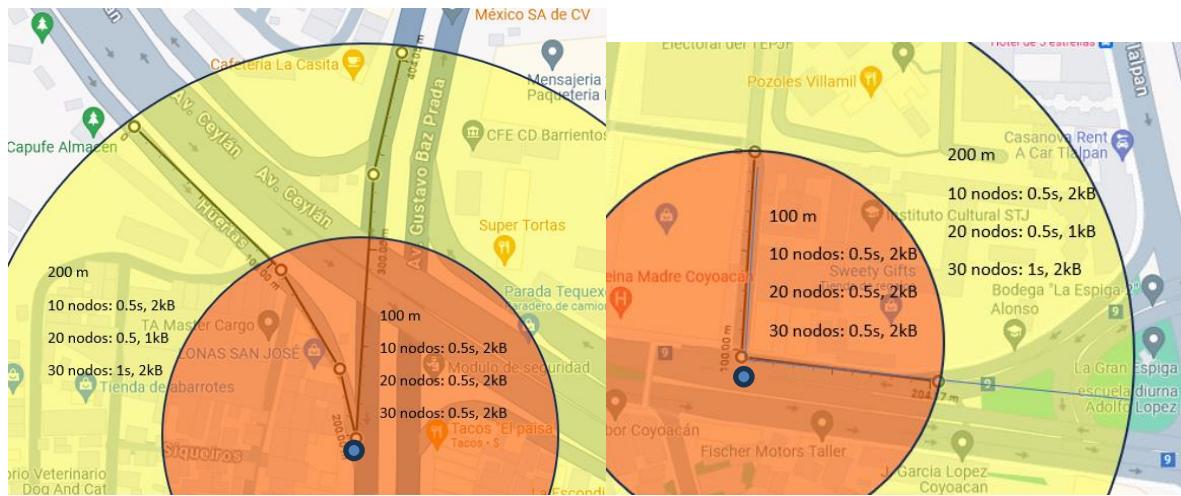


FIGURA 9-25 INTERVALOS DE TRANSMISIÓN RECOMENDADOS A 80 KM/H

Los mensajes con un tamaño de paquete mayor pueden funcionar en escenarios a altas velocidades, como se muestra a 80 km/h, donde incluso funciona con 20 nodos. En ellos, en lugar de tener que acceder al medio múltiples veces en un corto intervalo (por la velocidad), resulta mejor mandar menos paquetes de tamaño más grande. Esto funciona siempre que la densidad de nodos no sea muy alta, pues para estos casos, se observaría, sobre todo la necesidad de una estación central para coordinar y evitar la colisión de mensajes.

Respecto a los patrones de comunicación, con la idea de compartir las detecciones para la prevención de accidentes, en realidad no nos interesa abarcar gran área. Como se ve en los resultados, con 200 metros se logra notificar adecuadamente a los vehículos en la mayoría de los casos, y por la misma razón, en general un solo salto es suficiente para cubrir puntos ciegos. En cambio, buscar una gran cobertura (por medio de más saltos) puede traernos problemas de inundación. Lo que podría ser interesante, es el uso de alguna forma de geocasting, lo que requeriría de mayor inteligencia en el modelo, para que el mensaje solo

fueras procesados por autos a los que le sirva (por ejemplo, que se encuentren en el mismo carril).

## **10 CONCLUSIONES**

El tema abordado en el trabajo fue la programación de un nodo de ROS para la detección y rastreo de vehículos, junto con la simulación de una red VANET para la prevención de accidentes, en la que se identificaron los parámetros y se verificó su fiabilidad. En ambos casos se ha buscado marcar los pasos para la continuación en la investigación en vehículos autónomos, especialmente para el modelo de coche autónomo AutoModelcar.

Dentro de los detectores se mostró con detalle la implementación y se realizó un manual, junto con la documentación del carro, que permita avanzar en la programación de nuevos nodos que se ocupen de tareas como navegación, así mismo poder expandir el problema de detección. Desde el enfoque sobre el que se abordó, se buscó priorizar la velocidad de los métodos usados, para que fuera posible usar en el carro físico durante la conducción. La combinación de máquinas de soporte vectorial con filtros de cascada resultó ser efectiva en este punto, alcanzando una frecuencia de 5 Hz. Con tiempos para la detección de un fotograma completo de 300 a 400 ms y de solo regiones de 90-100 ms.

Sin embargo, dentro de la precisión se requiere de mayor trabajo. Para ello se debe ampliar el tamaño de los datasets para el entrenamiento, y mejorar el entrenamiento del filtro de cascada, que terminó siendo la parte más débil de la detección. Igualmente se puede replicar y ampliar para la detección de otros objetos, siguiendo los mismos pasos descritos. Dentro de las ideas aplicadas con SVM, fue el uso de dos detectores para el reconocimiento tanto de vehículos desde atrás, como de lado. Expandiendo esto, otras posibles mejoras a futuro, es

dividir la imagen en regiones, a las que a cada una aplicar detectores diferentes. De forma que se puedan suplir algunas de las fallas encontradas en los resultados, como detectar partes del coche cuando no está completo en la imagen, sin afectar gravemente el rendimiento (en lugar de aplicar todos los detectores sobre la ventana completa). Igualmente, agregar el uso del LiDAR para elegir regiones de interés, sobre las cuales hacer la detección.

La idea de utilizar estos métodos, en lugar de otras soluciones más simples para la evasión de obstáculos aplicadas directamente al vehículo físico, radica en que, al identificar el objeto y su movimiento, el vehículo puede generar un contexto y reconocer el entorno en el que se encuentra, lo que le permite modificar su comportamiento de manera más efectiva y segura. Si bien esto funciona para el modelo de AutoModelcar, como se ha mencionado en el trabajo, no puede igualar las ventajas que ofrecen las redes neuronales, especialmente con el continuo aumento en la capacidad computacional.

En el ámbito de las redes VANET, la combinación elegida incluye el uso de DSRC para que los nodos transmitan sus detecciones mediante broadcasting. Esto es especialmente útil para cubrir puntos ciegos como cruceros, donde la información propia de los vehículos puede ser insuficiente. Con esta información, los vehículos pueden tomar medidas evasivas o, como se propone, utilizar el protocolo AODV para la negociación y coordinación entre dos vehículos. Esta coordinación se basaría en la información propia que los vehículos comparten, como su ubicación geográfica mediante GPS y ruta. Información de otros vehículos detectados, como

la posición y velocidad estimada. Y estimaciones como la densidad del tráfico, obtenida a través del número de mensajes que reciben.

Durante las pruebas, se logró compartir mensajes con un payload de 1 kB, utilizando intervalos de aproximadamente 0.5 segundos en la mayoría de los escenarios. En algunos, incluso se puede recurrir a intervalos menores de 0.2s con 10 nodos, usar paquetes con 2 kB de payload, o en situaciones no críticas compartir con un intervalo de 1 segundo. En la mayoría de los casos, el mensaje se logra compartir con una alta probabilidad, lo que ayudaría a reducir incluso riesgo de colisiones. Las limitaciones se encuentran con velocidades de hasta 80 km/h donde el desempeño empeora considerablemente, y también, en el caso con 30 vehículos (todos tratándose de comunicar simultáneamente) a partir de velocidades de 50 km/h. Esto nos muestra que pueden ser fiables para la comunicación. Y en especial si se puede considerar que conoce su entorno, y sabe que se aproxima por ejemplo a un crucero. Los resultados pueden mejorar si considera la presencia de infraestructura para coordinar (ayudaría a reducir colisiones en las transmisiones, como retrasos para acceder al medio).

La investigación, por cuestiones de lo amplio que es el tema y por tiempo, fue muy limitada, pero se espera que, con los parámetros identificados, se pueda profundizar más. En especial se requiere la simulación de más escenarios, donde considerar más velocidades (y que no todos los nodos se muevan a la misma velocidad), densidades de tráfico y niveles de ruido. Mejoras que pueden ayudar a que los resultados de las simulaciones sean más realistas, se incluye también, el probar otros modelos de path loss, pues el rebote de las señales puede

empeorar los resultados. Así mismo, considerar temas de seguridad y realizar pruebas con autos de verdad.

Por último, se plantea cómo podría implementarse en el modelo de carro físico. Para ello se requiere de hardware para la conexión a redes inalámbricas. En AODV, se necesita de un módulo transceptor inalámbrico que opere en la frecuencia deseada, como 5.9 GHz (aquí se tiene mayor flexibilidad). El protocolo se puede implementar de forma directa en Linux, y hay opciones open source como AODV-UU. Para DSRC se necesita específicamente de un módulo compatible con IEEE 802.11p. Igualmente, se requiere de un receptor gps y software, para Linux se tiene gspd Daemon, que comparte las mediciones con todas las aplicaciones. Para su uso con Ros, a partir de sus versiones melodic y noetic, contiene la biblioteca gspd\_client.

Juntar DSRC con ROS de forma directa es una tarea más difícil, que requiere la implementación de una aplicación que extienda la capa 3. Pues DSRC no usa direcciones IP y ROS no considera el uso de IP dinámicas. Además, sería necesario el uso de bibliotecas como ROS\_multimaster. En este punto, para compartir las detecciones, sería mejor no usar ros para publicar y suscribirse, ni considerar cada vehículo como un nodo de ros. En su lugar, manejar DSRC para la comunicación de los nodos y la información recibida leerla con un nodo de ROS (en lugar de intentar conectarse a otro vehículo, para suscribirse a sus mensajes).

## REFERENCIAS

- “Access the tf Transformation Tree in ROS - MATLAB & Simulink - MathWorks América Latina”. s. f. Accedido 15 de abril de 2023. <https://la.mathworks.com/help/ros/ug/access-the-tf-transformation-tree-in-ros.html>.
- Arróspide, Salgado, y Nieto. “Video analysis based vehicle detection and tracking using an MCMC sampling framework” *EURASIP Journal on Advances in Signal Processing*. 2012. doi: 10.1186/1687-6180-2012-2
- Aswathy, M C. 2012. “A Cluster Based Enhancement to AODV for Inter-Vehicular Communication in VANET”. *International Journal of Grid Computing & Applications* 3 (3): 41-50. <https://doi.org/10.5121/ijgca.2012.3304>.
- Ayala-Alfaro, Victor. (2019) 2021. “AutoModelCar packages v3”. Python. <https://github.com/Victor-ayala/autoModelCar>.
- Bay, Herbert, Tinne Tuytelaars, y Luc Van Gool. 2006. “SURF: Speeded Up Robust Features”. En *Computer Vision – ECCV 2006*, editado por Aleš Leonardis, Horst Bischof, y Axel Pinz, 404-17. Berlin, Heidelberg: Springer. [https://doi.org/10.1007/11744023\\_32](https://doi.org/10.1007/11744023_32).
- Becker, Alex. s. f. “Online Kalman Filter Tutorial”. Accedido 5 de junio de 2023. <https://www.kalmanfilter.net/>.
- Bougharriou, Hamdaoui, y Mtibaa, 2017, “Linear SVM classifier based HOG car detection”. *2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)* 241-245. doi: 10.1109/STA.2017.8314922.
- Cabrera, Margarita, y Francesc Tarrés. s. f. “Multiplexación por división en frecuencias ortogonales (OFDM)”, Universidad Oberta de Cataluña, . [https://openaccess.uoc.edu/bitstream/10609/63345/2/Teor%C3%ADa%20de%20la%20codificación%20y%20modulaciones%20avanzadas\\_M%C3%B3dulo%205\\_Multiplexaci%C3%B3n%20por%20divisi%C3%B3n%20en%20frecuencias%20ortogonales%28OFDM%29.pdf](https://openaccess.uoc.edu/bitstream/10609/63345/2/Teor%C3%ADa%20de%20la%20codificación%20y%20modulaciones%20avanzadas_M%C3%B3dulo%205_Multiplexaci%C3%B3n%20por%20divisi%C3%B3n%20en%20frecuencias%20ortogonales%28OFDM%29.pdf).
- Coppola, Riccardo, y Maurizio Morisio. 2016. “Connected Car: Technologies, Issues, Future Trends”. *ACM Comput. Surv.* 49 (3). <https://doi.org/10.1145/2971482>.
- “catkin/workspaces - ROS Wiki”. s. f. Accedido 12 de abril de 2023. <http://wiki.ros.org/catkin/workspaces>.

“Converting between ROS images and OpenCV images (C++)”. 2017. Accedido 4 de septiembre de 2023.

[http://wiki.ros.org/cv\\_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages](http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages).

Das, Samir R., Charles E. Perkins, y Elizabeth M. Belding-Royer. 2003. “Ad hoc On-Demand Distance Vector (AODV) Routing”. Request for Comments RFC 3561. Internet Engineering Task Force. <https://doi.org/10.17487/RFC3561>.

“es - ROS Wiki”. s. f. Accedido 12 de abril de 2023. <http://wiki.ros.org/es>.

“Estadísticas a propósito del día mundial en recuerdo de las víctimas de accidentes de tránsito”. 2022, INEGI Comunicado de prensa, n.<sup>o</sup> 662/22 (noviembre): p. 6

Faisal, Asif, Tan Yigitcanlar, Md. Kamruzzaman, y Graham Currie. 2019. “Understanding Autonomous Vehicles: A Systematic Literature Review on Capability, Impact, Planning and Policy”. *Journal of Transport and Land Use* 12 (1). <https://doi.org/10.5198/jtlu.2019.1405>.

Felipe Pérez. (2017) 2017. “AP-automodelcar”. C++. <https://github.com/Fperez/AP-automodelcar>.

Feng Han, Ying Shan, Ryan Cekander, Harpreet S. Sawhney, y Rakesh Kumar. 2006. “A Two-Stage Approach to People and Vehicle Detection With HOG-Based SVM”. *PERFORMANCE METRICS FOR INTELLIGENT SYSTEMS WORKSHOP*, PERFORMANCE METRICS FOR INTELLIGENT SYSTEMS WORKSHOP, 133-40.

Fernández Villaverde, Jesús. s. f. “Kalman and Particle Filtering”. University of Pennsylvania. [https://www.sas.upenn.edu/~jesusfv/filters\\_format.pdf](https://www.sas.upenn.edu/~jesusfv/filters_format.pdf).

Freund, Yoav, y Robert E. Schapire. 1995. “A desicion-theoretic generalization of on-line learning and an application to boosting”. En *Computational Learning Theory*, editado por Paul Vitányi, 23-37. Berlin, Heidelberg: Springer Berlin Heidelberg.

Geiger A., P. Lenz, C. Stiller, y R. Urtasun. 2013. “Vision meets Robotics: The KITTI Dataset”. *International Journal of Robotics Research (IJRR)*, International Journal of Robotics Research (IJRR), . <https://www.cvlibs.net/publications/Geiger2013IJRR.pdf>.

Geiger, A., P. Lenz, y R. Urtasun. 2012. “Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. En *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 3354-61. Providence, RI: IEEE. <https://doi.org/10.1109/CVPR.2012.6248074>.

- Girshick, Ross. 2015. “Fast R-CNN”. arXiv. <https://doi.org/10.48550/arXiv.1504.08083>.
- Granados Osegueda, Edgar Alejandro. 2019. “Estimación y rastreo de riesgo en planificadores basados en árboles hacia un movimiento seguro en entornos dinámicos”. Ciudad de México: Instituto Tecnológico Autónomo de México.
- Hartenstein, Hannes, y Kenneth P Laberteaux, eds. 2010. *VANET: Vehicular Applications and Inter-Networking Technologies*. 1.<sup>a</sup> ed. Wiley. <https://doi.org/10.1002/9780470740637>.
- “Hardware – AutoModelcarWiki”. 2017. GitHub. Accedido 31 de mayo de 2024. <https://github.com/AutoModelCar/AutoModelCarWiki/wiki/Hardware>.
- “Hardware (AutoNOMOS Model v2) · AutoModelCar/AutoModelCarWiki Wiki · GitHub”. s. f. Accedido 11 de julio de 2023. [https://github.com/AutoModelCar/AutoModelCarWiki/wiki/Hardware-\(AutoNOMOS-Model-v2\)](https://github.com/AutoModelCar/AutoModelCarWiki/wiki/Hardware-(AutoNOMOS-Model-v2)).
- “Home · AutoModelCar/AutoModelCarWiki Wiki”. s. f. Accedido 5 de octubre de 2023. <https://github.com/AutoModelCar/AutoModelCarWiki/wiki>.
- Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, y Hartwig Adam. 2017. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. arXiv. <https://doi.org/10.48550/arXiv.1704.04861>.
- “IDENTIFICACIÓN DE NECESIDADES DE ESPECTRO PARA SISTEMAS DE TRANSPORTE INTELIGENTE EN LA BANDA 5850-5925 MHz”. 2021. Instituto Federal de Telecomunicaciones. [https://www.ift.org.mx/sites/default/files/industria/temasrelevantes/17437/documentos/documentodereferenciaidentificaciondenecesadesstien59ghz\\_0.pdf](https://www.ift.org.mx/sites/default/files/industria/temasrelevantes/17437/documentos/documentodereferenciaidentificaciondenecesidadesstien59ghz_0.pdf).
- “IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”. 2012. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, marzo, 1-2793. <https://doi.org/10.1109/IEEESTD.2012.6178212>.
- Jakubiak, Jakub, y Yevgeni Koucheryavy. 2008. “State of the Art and Research Challenges for VANETS”. En *2008 5th IEEE Consumer Communications and Networking Conference*, 912-16. Las Vegas, Nevada, USA: IEEE. <https://doi.org/10.1109/ccnc08.2007.212>.

- Janai, Joel, Fatma Güney, Aseem Behl, y Andreas Geiger. 2021. “Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art”. arXiv. <http://arxiv.org/abs/1704.05519>.
- Janne Salmi. 2000. “AODV Multicast Features”. Department of Computer Science and Engineering Helsinki University of Technology. <https://www.niksula.hut.fi/~janski/iwork/>.
- Kecman, Vojislav. 2005. “Support Vector Machines – An Introduction”. En *Support Vector Machines: Theory and Applications*, 177:605-605. [https://doi.org/10.1007/10984697\\_1](https://doi.org/10.1007/10984697_1).
- Kenney, John. 2011. “Dedicated Short-Range Communications (DSRC) Standards in the United States”. *Proceedings of the IEEE* 99 (agosto): 1162-82. <https://doi.org/10.1109/JPROC.2011.2132790>.
- Kevin Gay. 2022. “WG: VT/ITS/AV Decision Making”. IEEE. 2022. <https://sagroups.ieee.org/2846/>.
- Khairnar, Vaishali, y Ketan Kotecha. 2013. “Performance of Vehicle-to-Vehicle Communication using IEEE 802.11p in Vehicular Ad-hoc Network Environment”. *International Journal of Network Security & Its Applications* 5 (abril). <https://doi.org/10.5121/ijnsa.2013.5212>.
- Kyle Barratt. 2021. “DSRC vs C-V2X: Comparing the Connected Vehicles Technologies - GTT Wireless”. 18 de noviembre de 2021. <https://www.gttwireless.com/dsrc-vs-c-v2x-comparing-the-connected-vehicles-technologies/>.
- Lindeberg, Tony. 2012. “Scale Invariant Feature Transform”. En *Scholarpedia*. Vol. 7. <https://doi.org/10.4249/scholarpedia.10491>.
- Luenberger, D. 1971. “An introduction to observers”. *IEEE Transactions on Automatic Control* 16 (6): 596-602. <https://doi.org/10.1109/TAC.1971.1099826>.
- Martínez-Díaz, Margarita, y Francesc Soriguera. 2018. “Autonomous Vehicles: Theoretical and Practical Challenges”. *Transportation Research Procedia* 33: 275-82. <https://doi.org/10.1016/j.trpro.2018.10.103>.
- Nelson, Joseph. 2020. “The Importance of Blur as an Image Augmentation Technique”. Roboflow Blog. 13 de marzo de 2020. <https://blog.roboflow.com/using-blur-in-computer-vision-preprocessing/>.
- O’Kane, Jason M. 2014. *A Gentle Introduction to ROS*. Version 2.1.1 (3e3d9c5), Generated in November 20, 2014. Columbia, SC: Jason M. O’Kane.

Ojala, Timo, Matti Pietikäinen, y David Harwood. 1996. “A comparative study of texture measures with classification based on featured distributions”. *Pattern Recognition* 29 (1): 51-59. [https://doi.org/10.1016/0031-3203\(95\)00067-4](https://doi.org/10.1016/0031-3203(95)00067-4).

“OpenCV: Cascade Classifier”. s. f. Accedido 29 de mayo de 2023.  
[https://docs.opencv.org/3.4/db/d28/tutorial\\_cascade\\_classifier.html](https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html).

“OpenCV: Cascade Classifier Training”. s. f. Accedido 21 de abril de 2023.  
[https://docs.opencv.org/3.4/dc/d88/tutorial\\_traincascade.html](https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html).

“OpenCV: Introduction to Support Vector Machines”. s. f. Accedido 31 de mayo de 2023.  
[https://docs.opencv.org/4.x/d1/d73/tutorial\\_introduction\\_to\\_svm.html](https://docs.opencv.org/4.x/d1/d73/tutorial_introduction_to_svm.html).

“OpenCV: Smoothing Images”. s. f. Accedido 31 de mayo de 2023.  
[https://docs.opencv.org/4.x/d4/d13/tutorial\\_py\\_filtering.html](https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html).

OpenCV. 2015. “Open Source Computer Vision Library”.

Pérez Reyes. 2023. “AutoModelCar\_2023\_UPV”. C++.  
[https://github.com/OmarPerezReyes/AutoModelCar\\_2023\\_UPV](https://github.com/OmarPerezReyes/AutoModelCar_2023_UPV).

Pietikäinen, Matti. 2010. “Local Binary Patterns”. *Scholarpedia* 5 (3): 9775.  
<https://doi.org/10.4249/scholarpedia.9775>.

“Point Cloud Library (PCL): pcl::PointXYZRGBA Struct Reference”. s. f. Accedido 12 de abril de 2023. [https://pointclouds.org/documentation/structpcl\\_1\\_1\\_point\\_x\\_y\\_z\\_r\\_g\\_b\\_a.html](https://pointclouds.org/documentation/structpcl_1_1_point_x_y_z_r_g_b_a.html).

Pressman, R. 2010. *Ingeniería del Software un Enfoque Práctico*. México, D.F: McGraw-Hill

Redmon, Joseph, Santosh Divvala, Ross Girshick, y Ali Farhadi. 2016. “You Only Look Once: Unified, Real-Time Object Detection”. arXiv. <https://doi.org/10.48550/arXiv.1506.02640>.

Ren, Shaoqing, Kaiming He, Ross B. Girshick, y Jian Sun. 2015. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. *CoRR* abs/1506.01497.  
<http://arxiv.org/abs/1506.01497>.

“Road vehicles — Functional safety ISO 26262-1:2011”. ISO. 2011.  
<https://www.iso.org/standard/43464.html>.

Rosebrock, Adrian. 2015. *Local Binary Patterns with Python & OpenCV*. PyImageSearch (blog). 7 de diciembre de 2015. <https://pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/>.

Saleh, Kaziwa, Sándor Szénási, y Zoltán Vámossy. s. f. *Occlusion Handling in Generic Object Detection*

*Setting Up Transformations — Navigation 2 1.0.0 documentation*. s. f. Accedido 15 de abril de 2023. [https://navigation.ros.org/setup\\_guides/transformation/setup\\_transforms.html](https://navigation.ros.org/setup_guides/transformation/setup_transforms.html).

Shetty, Anirudha B., Bhoomika, Deeksha, Jeevan Rebeiro, y Ramyashree. 2021. *Facial recognition using Haar cascade and LBP classifiers*. Global Transitions Proceedings 2 (2): 330-35. <https://doi.org/10.1016/j.gltip.2021.08.044>.

Shuttleworth, Jennifer. 2019. “SAE J3016 Automated-Driving Graphic”. 2019. <https://www.sae.org/site/news/2019/01/sae-updates-j3016-automated-driving-graphic>.

Song, Xibin, Peng Wang, Dingfu Zhou, Rui Zhu, Chenye Guan, Yuchao Dai, Hao Su, Hongdong Li, y Ruigang Yang. 2019. “ApolloCar3D: A Large 3D Car Instance Understanding Benchmark for Autonomous Driving”. En *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 5447-57. Long Beach, CA, USA: IEEE. <https://doi.org/10.1109/CVPR.2019.00560>.

*SVM bias on weights of positives and negatives - OpenCV Q&A Forum*. s. f. Accedido 18 de abril de 2023. <https://answers.opencv.org/question/26818/svm-bias-on-weights-of-positives-and-negatives/>.

Tariq Islam y Cheolhyeon Kwon. 2022. *Survey on the State-of-the-Art in Device-to-Device Communication: A Resource Allocation Perspective / Elsevier Enhanced Reader*. 2022. <https://doi.org/10.1016/j.adhoc.2022.102978>.

Tyagi, Mrinal. 2021. *HOG(Histogram of Oriented Gradients)*. Medium. 24 de julio de 2021. <https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f>.

Viola, P., y M. Jones. 2001. “Rapid object detection using a boosted cascade of simple features”. En *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 1:I-I. <https://doi.org/10.1109/CVPR.2001.990517>.

*What Are Convolutional Neural Networks? / IBM*. s. f. Accedido 5 de junio de 2023. <https://www.ibm.com/topics/convolutional-neural-networks>.

Zetina, Erick. (2018) 2023. “ai\_controlled\_car”. Python C++.  
[https://github.com/Zetinator/ai\\_controlled\\_car](https://github.com/Zetinator/ai_controlled_car).

Zhou, Zhi-Qiang. 2019. *HOG Visualization Using OpenCV(C++)*. Github.  
<https://github.com/zhouzq-thu/HOGImage>.

## APÉNDICE 1: DOCUMENTACIÓN

### Documentación AutoModelCar

#### AutoModelCar

Consiste en vehículos autónomos a escala 1/10 desarrollados en Freie Universität Berlin para propósitos de educación, en el ITAM se cuenta con 2 vehículos uno corresponde al modelo 1 y el otro al 2. En la figura A1-1 se presenta la arquitectura de ambos.

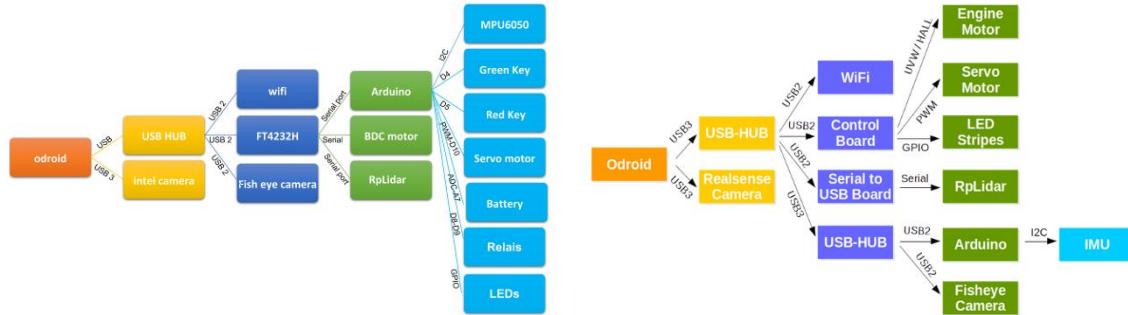


FIGURA A1-1 A LA IZQUIERDA MÓDULOS Y CONEXIONES DEL MODELO 1, A LA DERECHA EL MODELO 2. FUENTE: (“HARDWARE (AUTONOMOS MODEL V2) · AUTOMODELCAR/AUTOMODELCARWIKI WIKI · GITHUB” S. F.)

#### Inicio

Pulse el botón verde durante 3 segundos para que encienda.

#### Conexión a la odroid

Para la conexión al robot se puede realizar conectándose a través de la red del ITAM, tanto por ethernet como wifi, mediante una terminal se debe escribir:

Carro 1

```
ssh root@148.205.37.48
ssh root@10.3.79.41      (Wifi) se conecta a la red WIFI_ITAM
ssh root@10.10.0.240    (Wifi) se conecta a la red ITAM
```

Carro 2

```
ssh root@10.10.0.250 ITAM
ssh root@192.168.1.199 conexión directa
```

Password: elfmeter

El nombre del robot 1 es Robot8, del robot 2 es Modelcar-126.

Para ejecutar el programa principal se debe pulsar el botón verde nuevamente o escribir  
.autostart.sh (se debe activar los motores para mandarle velocidades con auto\_stop)

## Instalación

Para instalar los repositorios del carro autónomo se deben bajar de GitHub:

```
$ git clone https://github.com/AutoModelCar/model_car.git
```

Si se requiere otra versión en concreto, se debe buscar el commit hash correspondiente a la versión deseada. Por ejemplo, para la versión 1 sobre la que se trabaja:

```
$ git checkout fed4cef394e412ec1f3aaa51586a6eb7689b4cc7
```

Los paquetes se pueden construir con dos herramientas:

- Catkin\_build

Corresponde a los pasos de instalación indicados en la wiki del AutoModel:

```
$ cd model_car
```

```
$ ./copy.sh
```

¡Importante se debe cambiar la dirección por la del carro!

Copiará archivos como el bashrc y el autostart a la raíz, extrae los archivos set y set2.sh

Actualizar la fecha

```
$ date --set "yyyy-mm-dd hh:mm:ss"
```

```
$ ./set.sh
```

Reiniciar el carro

```
$ ./set2.sh
```

Reiniciar nuevamente el carro

- Catkin make

Otra alternativa que es la usada es mediante catkin make, para ello se debe primero crear un nuevo catkin workspace:

```
$ mkdir -p ~/catkin_ws/src  
$ cd ~/catkin_ws/
```

Se descarga el programa en otra carpeta

```
$ git clone https://github.com/AutoModelCar/model_car.git  
$ git checkout fed4cef394e412ec1f3aaa51586a6eb7689b4cc7
```

Sobre el directorio descargado, se copia únicamente la carpeta src dentro del catkin\_ws creado:

```
$cp -r src /path donde se creó el nuevo workspace  
$ catkin_make
```

Por un lado, catkin make es una herramienta para compilar código en un workspace de catkin, equivale a la unión de los comandos cmake y make, y debe ser llamado siempre desde la raíz.

En el proceso se agregan los directorios build y devel, junto con los archivos CMakeLists, los cuales preparan y ejecutan el proceso de compilación.

Por otro lado, catkin build se comporta como catkin\_make\_isolated, el cual aísla cada paquete en source y los construye de manera separada. De tal manera que no hay un archivo CMakeLists en el nivel más alto, sino que cada paquete es independiente y, por lo tanto, se puede llamar catkin build desde cualquier directorio.

Sin embargo, se debe tener cuidado con la información que se pasan entre paquetes en los archivos CMake, pues estos generan las dependencias entre todos los paquetes.

Catkin\_build y catkin\_make son incompatibles entre sí, usar uno u otro depende de cómo se construyó el repositorio originalmente. Debido a que se cuentan con dos repositorios en el vehículo, el original catkin\_ws y sobre el que se trabaja en workspaceAutonomos\catkin\_ws1\., ambos están construidos con catkin\_make. Sin embargo, hay algunas configuraciones dentro de opt/ros/indigo que catkin\_make no realiza por lo que se deben reemplazar manualmente, para poder operar la cámara de profundidad. Aun así, el uso de catkin\_build ya no es recomendado por ROS y ya no recibe mantenimiento.

Si se quiere cambiar el repositorio en el que se trabaja se puede hacer modificando el .bashrc que se encuentra en raíz, hasta abajo se debe agregar la dirección del que se desea usar. Es posible que se creen algunas dependencias entre repositorios si se usa catkin\_build, el cual

se puede usar para correr los nodos base, pero no se pueden compilar nuevos programas por incompatibilidad con los otros workspaces.

## Correcciones

Las siguientes modificaciones solo se hacen sobre la versión 1 del robot, y corresponden a cambios en algunos archivos launch relacionados con los nombres de los tópicos que utiliza:

Dentro de launch random movement

- `nano random_movement/launch/auto.launch`

Reemplazar:

```
<include file="$(find light)/launch/light.launch"/>  
<include file="$(find  
send_steering)/launch/servo_odroid.launch"/>
```

Por:

```
<include file="$(find  
send_steering_light)/launch/servo_light_odroid.launch"/>
```

- `nano manual_control/launch/manual_PC.launch`

Reemplazar:

```
<include file="$(find  
send_steering)/launch/servo_odroid.launch"/>
```

Por:

```
<include file="$(find  
send_steering_light)/launch/servo_light_odroid.launch"/>
```

Como se mencionó anteriormente, debido a que los paquetes fueron construidos con catkin\_make, es posible que algunas variables no se establezcan correctamente, por lo que, para poder usar la cámara de profundidad, es necesario usar el nodelet que se construye con catkin\_build. Lo que se hizo fue copiar la biblioteca de la versión 2 del vehículo:

Se copio librealsense del directorio raíz y el directorio /opt/ros/indigo/ completo. El problema se da en algunas configuraciones que realiza catkin\_build a la hora de construir los paquetes que catkin\_make no realiza, por lo que el programa de nodelet\_manager no puede levantar todos los nodos y entonces la cámara SR300 no logra funcionar.

Dentro de workspaceAutonomos se tienen los directorios lib y lib2, que corresponden a la biblioteca librealsense que traía originalmente el carro y la de la versión 2 respectivamente. Se tienen también op y op2, de igual forma op corresponde al directorio original y op2 al de la versión 2 por la que se sustituye.

Para copiar las bibliotecas se utilizan los comandos:

```
cp -r workspaceAutonomos/lib2/ /root/  
cp -r workspaceAutonomos/op2/indigo/ /opt/ros/
```

Las bibliotecas originales lir y op se guardaron dentro de workspaceAutonomos.

Cada que se utiliza catkin\_make para compilar todos los paquetes completos, hay que remplazar nuevamente la biblioteca de opt/ros/indigo. Una solución más cómoda es usar Catkin\_make solo la primera vez que se agrega un nuevo paquete, después se puede usar:

```
catkin_make --only-pkg-with-deps prueba
```

Lo que evita tener que copiar nuevamente las bibliotecas. Igualmente, como se mencionó en la guía algunos paquetes fallan al compilar con cmake, lo que interrumpe su operación, por lo que se debe volver a correr hasta que no haya problemas para que agregue todos los paquetes. De lo contrario, habrá algunos nodos que no podrá encontrar al correr el programa principal.

En el caso específico del carro puede tener problemas con el reloj que impiden compilar “clock skew detected”. Para corregirlo se puede actualizar las marcas de tiempo con:

```
find . -exec touch {} \\;
```

Desde las direcciones /root/workspace/catkin\_ws1 y /opt/ros/

Otro problema que puede aparecer es el error: c++: internal compiler error: Killed (program cc1plus), que normalmente ocurre porque se queda sin memoria. Como solución se agrega el parámetro -j, que limita el número de trabajos a costa de aumentar el tiempo de la compilación:

```
catkin_make -j 4
```

## Configuración

Se tienen 4 archivos principales que puede requerirse modificar para el uso del modelo a escala, los mostrados corresponden a la versión 1 del carro, para la versión 2 es similar, con pequeños cambios como en algunas direcciones.

### Autostart.sh

```
-----  
#!/bin/bash  
  
modprobe uvcvideo  
  
# clear ROS log  
  
rm -rf /root/.ros/log  
  
# load environment settings from .bashrc  
  
PS1='\$ '  
  
source /root/.bashrc  
  
### Start ROS core and wait a few seconds  
  
roscore &  
  
sleep 5  
  
### Start ROS launch scripts  
  
#timeout 10 roslaunch realsense_camera realsense_sr300.launch  
roslaunch manual_control manual_odroid.launch #|| exit  
  
#roslaunch random_movement auto.launch #|| exit
```

El autostart es un script que nos permite iniciar el programa elegido por defecto, lo que nos ahorra hacer algunos comandos para iniciar todos los programas. En él se hace la llamada al archivo .bashrc para realizar los exports y el source al workspace, aquí se inicia el roscore y se llama al programa principal con roslaunch que por defecto es el manual\_control. Es posible dentro del archivo mandar a llamar múltiples archivos launch simultáneamente, como son los launch para la cámara de ojo de pez, (sin embargo, el autor de los repositorios originales no recomienda usar las dos cámaras simultáneamente por incompatibilidades).

---

### .bashrc

---

```
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion

fi

# configure bash history

HISTSIZE=100          # save only up to 100 commands
HISTCONTROL=ignoreboth # ignore duplicate commands
shopt -s histappend    # append to history file instead of
                       # overwriting it

export PROMPT_COMMAND="history -a; $PROMPT_COMMAND"

# Some applications read the EDITOR variable to determine your
# favourite text
```

```

# editor. So uncomment the line below and enter the editor of
# your choice :-)

export EDITOR=/usr/bin/joe

test -s ~/.alias && . ~/.alias

# Set shell prompt

NOCOLOR="\[\033[0;0m\]"

MAGENTA="\[\033[0;35m\]"

GREEN="\[\033[0;32m\]"

RED="\[\033[0;33m\]"

if test "$UID" = 0 ; then

LOGINCOLOR=$RED

else

LOGINCOLOR=$GREEN

fi

PS1="$RED# $NOCOLOR\$ (date +%H:%M:%S)

$LOGINCOLOR\h:\w>$NOCOLOR "

export PS1

# Set path

export PATH=$PATH:/root:/sbin:/usr/sbin:/usr/local/sbin

# Look for libraries in local folder as well

export LD_LIBRARY_PATH=${LD_LIBRARY_PATH:+$LD_LIBRARY_PATH}.

export ROS_IP=148.205.37.48

export ROS_HOSTNAME=148.205.37.48

export ROS_MASTER_URI=http://148.205.37.48:11311

```

```
source /opt/ros/indigo/setup.bash  
#source /root/catkin_ws/devel/setup.bash  
source workspaceAutonomos/catkin_ws1/devel/setup.bash
```

El bashrc es otro script que realiza una serie de configuraciones sobre las variables tanto del Shell como del entorno, como definir algunos colores y alias, así como definir el tamaño del historial. Sin embargo, lo que nos importan son los comandos:

- Ros IP y Hostname que definen la dirección de algún nodo que se quiera comunicar con el robot.
- Ros Master URI define la dirección del maestro que corresponde a la odroid, de donde se ejecutan los nodos.
- El bashrc realiza los export, realiza el source al setup.bash tanto del workspace activo como a la versión de ROS instalada. Llama, además, a exportar el path y las bibliotecas, que definen en qué orden buscar los paquetes:

```
PATH=/opt/ros/indigo/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr  
/bin:/sbin:/bin:/root:/sbin:/usr/sbin:/usr/local/sbin
```

- Por último, carga con source el repositorio a utilizar. Source es un comando que ejecuta los archivos pasados como argumento sobre el mismo shell.

Para el caso del proyecto el repositorio que se usa es el que se encuentra ubicado en source workspaceAutonomos/catkin\_ws1/

---

## **interfaces**

Permite definir direcciones de red tanto cableada como inalámbrica, ubicado en /etc/network/interfaces

```
-----  
# /etc/network/interfaces -- configuration file for ifup(8),  
ifdown(8)  
  
# The loopback interface  
  
auto lo  
  
iface lo inet loopback  
  
# Wired interfaces  
  
auto eth0  
  
iface eth0 inet static  
  
    address 148.205.37.48  
  
    netmask 255.255.252.0  
  
    gateway 148.205.36.2  
  
    broadcast 148.205.39.255  
  
    dns-nameservers 148.205.228.11  
  
#  
  
# Wireless interfaces  
  
#  
  
iface wlan0 inet static  
  
    address 10.10.0.240  
  
    netmask 255.255.224.0
```

```
broadcast 10.10.31.255  
gateway 10.10.10.1  
wpa-conf /etc/wpa_supplicant.conf  
wpa-driver wext  
down killall wpa_supplicant
```

El archivo interfaces permite definir los nombres lógicos para interfaces de redes y su configuración. Aquí se tienen definidas la dirección de red ethernet del carro como: 148.205.37.48, y la dirección inalámbrica: 10.10.0.240.

---

### **wpa\_supplicant**

Ubicado en /etc/wpa\_supplicant.conf

---

```
#network={  
#      ssid="EK"  
#      key_mgmt=WPA-PSK  
#      psk="robotica2016"  
#      Unencrypted  
network={
```

```
    ssid="ITAM"  
    key_mgmt=NONE  
}
```

WPA supplicant es un archivo de configuración para implementar WPA u otros protocolos de seguridad que usan las redes Wifi. El robot (cliente) debe realizar la negociación con el autenticador WPA, en el que indica el nombre de la red a la que debe conectarse, contraseña y tipo de clave.

## Conexión a la aplicación

<https://github.com/AutoModelCar/AutoModelCarWiki/wiki/Android-APP>

En la wiki se encuentra una aplicación para controlar el carro a control remoto, permite mandar la velocidad, el ángulo de giro y un botón de paro de emergencia. Únicamente en Master\_URI se debe ingresar la dirección IP de la interfaz wlan0 a la que está conectado.

<http://10.10.0.240:11311/>

Possiblemente por problemas de la versión y el formato de los mensajes, solo es posible mandar la velocidad. Pues como se ve en las correcciones en los launch, se realizan modificaciones sobre el tipo de mensaje del nodo steering.

## **Conexión a la computadora**

```
$ export ROS_IP=148.205.37.41 %Dirección de la computadora  
$ export ROS_HOSTNAME=148.205.37.41 %Dirección de la  
computadora  
$ export ROS_MASTER_URI=http://148.205.37.48:11311/ %Dirección  
del robot
```

Permite enviar comandos de forma remota o visualizar datos:

Ros\_ip y ros\_hostname basta con usar solo uno de los dos. Ambas se excluyen mutuamente y de usarse las dos, se tomará hostname.

No es necesario que la versión de ROS de la computadora sea la misma que la del carro.

Para pasar datos de una computadora al vehículo:

```
scp /home/ubuntu/myfile  
username@IP_of_windows_machine:/C:/Users/Name/Desktop  
  
scp -r  
ifons@192.168.1.115:/C:/Users/ifons/source/repos/detec2/dete  
c2/pruebas/my_hogE10000.xml  
/root/workspace/catkin_ws1/src/vision_camara/src/
```

## **Visualización**

```
$ rqt_graph
```

Es una interfaz gráfica de ROS que nos permite visualizar en forma de grafo los nodos y tópicos que se están ejecutando.

```
$ rviz
```

Es una herramienta de visualización en 2D y 3D que despliega las lecturas de los sensores como el lidar y las cámaras.

```
$ gazebo
```

Es un simulador de robótica de código abierto en 3D, que permite la manipulación de sensores y actuadores, junto a un motor de físicas, que le permite modelar la forma en que un robot interactúa con el ambiente simulado.

### **Nodos y programas**

Programas principales:

*random\_movement*

Manda algunos valores aleatorios de velocidad y giro.

*manual\_control*

Recibe la velocidad y steering para el carro, publica la información de la cámara de profundidad SR300 y el LiDAR.

## Actuadores

### Speed

```
rostopic -r 10 pub /manual_control/speed std_msgs/Int16  
{"data: -1000"}
```

Tipo: std\_msgs/Int16 carro 1, Int32 carro 2

Campos:

int16 data

Valores hasta -10,000, basta con -1,000

Para avanzar dar los valores en negativo

-r 10 es la tasa en Hertz a la que se publica el mensaje

### Steering

```
rostopic pub /manual_control/steering std_msgs/Int16 {"data: 180"}
```

Tipo: std\_msgs/Int16

Campos:

int16 data

Valores entre 0-180 para el carro 1

0 gira a la derecha

180 a la izquierda

90 centra las llantas

Carro 2 entre 1040-2140

1540 centro

Stop start

```
rostopic pub /manual_control/stop_start std_msgs/Int16 {"data: 1"}
```

Tipo: std\_msgs/Int16

Campos:

int16 data

Valores entre 0 y 1

0 para encender los motores

1 para apagarlos, se usa como paro de emergencia

Lights

```
rostopic pub /manual_control/lights std_msgs/String {"data: le"}
```

Tipo: std\_msgs/Int16

Campos:

string data

Enciende los leds con los que cuenta el vehículo.

string               Luz

---

le                   Parpadeo de la luz amarilla izquierda

ri                   Parpadeo de la luz amarilla derecha

stop               Enciende los 3 leds rojos de atrás (freno)

pa	Enciende 2 leds rojos de atrás, y 2 blancos frontales de la izquierda
ta	Enciende 2 leds rojos de atrás, y 2 blancos frontales de la derecha
re	Enciende las luces blancas traseras, para cuando va de reversa
fr	Enciende las luces blancas delanteras, para cuando se conduce en la oscuridad
diL	Apaga las luces

## Sensores

LiDAR

Tópico:

`/scan`

Tipo: sensor\_msgs/LaserScan

Campos:

std_msgs/msg/Header header	float scan_time
float angle_min	float range_min
float angle_max	float range_max
float angle_increment	float[] ranges
float time_increment	float[] intensities

Devuelve la información del LiDAR, el cual hace un recorrido de 360° y tiene un alcance máximo de 8 metros, devuelve un arreglo con una medida para cada ángulo. Es el arreglo de ranges el que contiene las distancias en metros.

## Cámara de profundidad

Tópicos:

/camera/color/image\_raw (sensor\_msgs/Image)

/camera/color/camera\_info

camera/depth/image\_raw

camera/depth/camera\_info

camera/depth/points

camera/infrared1/image\_raw

camera/infrared1/camera\_info

camera/infrared2/image\_raw

camera/infrared2/camera\_info

Los mensajes tipo raw corresponden al formato sensor\_msgs/Image, los tipos info son datos con los que están calibradas. Mientras que depth/points usa el mensaje tipo PointCloud2 e incorpora los datos de las 3 cámaras.

Tópico depth/points

Tipo sensor\_msgs/PointCloud2

Campos:

std_msgs/msg/Header header	sensor_msgs/msg/PointField[] fields
uint32 height	boolean is_bigendian
uint32 width	uint32 point_step
	uint32 row_step

uint8[] data	boolean is_dense
--------------	------------------

El campo fields define el formato con el que se representan los datos, que en este caso es float32, mientras los datos vienen en data. Este tópico no se publica en el simulador.

Tópico: /app/camera/rgb/image\_raw

Tipo /sensor\_msgs/Image

Campos:

std_msgs/msg/Header header	uint8 is_bigendian
----------------------------	--------------------

uint32 height	uint32 step
---------------	-------------

uint32 width	uint8[] data
--------------	--------------

string encoding

Cámara ojo de pez

No está incluido en el launch de manual\_control, se debe ejecutar de forma independiente con:

```
roslaunch fisheye_camera_matrix camera_matrix_publisher.launch
```

Se puede usar para navegación mediante marcas en el techo, con las cuales obtener la pose del carro.

## Otros

Tópico /odom

Tipo: nav\_msgs

Campos:

std\_msgs/msg/Header header

string child\_frame\_id

geometry\_msgs/msg/PoseWithCovariance pose

geometry\_msgs/msg/TwistWithCovariance twist

geometry\_msgs/PoseWithCovariance:

geometry\_msgs/msg/Pose pose

double[36] covariance

geometry\_msgs/Pose:

geometry\_msgs/msg/Point position

geometry\_msgs/msg/Quaternion orientation

geometry\_msgs/Point:

double x

double y

double z

geometry\_msgs/Quaternion:

double x=0.0

double y=0.0

double z=0.0

double w=1.0

geometry\_msgs/TwistWithCovariance:

    geometry\_msgs/msg/Twist twist

        double[36] covariance

geometry\_msgs/Twist:

    geometry\_msgs/msg/Vector3 linear

    geometry\_msgs/msg/Vector3 angular

geometry\_msgs/Vector3:

    double x

    double y

    double z

Tópico /Pose

Tipo: geometry\_msgs/Pose2D

Campos:

    float64 x

    float64 y

    float64 theta

Solo en el simulador.

Para correr el programa de planificador de movimiento:

roslaunch lane\_navigation simulador\_gary.launch

## Mensajes

Ros funciona a través de nodos o programas independientes que corren simultáneamente y se comunican entre sí, para hacerlo emplean un mecanismo de publicadores y suscriptores. Un nodo que quiere escuchar determinado mensaje debe suscribirse al tópico correspondiente, los cuales se pueden ver con:

`rostopic list`

`rosnode list` muestra los nodos que se están ejecutando.

Igualmente se puede obtener información adicional, como la estructura y tipo de los mensajes con:

`rostopic info` y `rosmsg info`

Se pueden mostrar los mensajes publicados con:

`rostopic echo`

Para la comunicación con mensajes se pueden definir nuevos, o usar bibliotecas que ya tienen definidos algunos formatos como:

Std\_msgs: Tipos primitivos de mensajes y arreglos, se usan para mandar la velocidad y el steering con tipo Int16

Sensor\_msgs: incluye mensajes tipo PointCloud2 para la cámara y LaserScan para el LiDAR.

Nav\_msgs: mensajes para mapas y odometría.

Geometry\_msgs: mensajes de tipo pose, transformadas, puntos y vectores.

Existe otro mecanismo que es el de servicios, sin embargo, para el caso del carro autónomo no se utiliza.

### **Nodelets**

Son un tipo de nodo de ROS que permite correr múltiples nodos en un solo proceso, lo que permite se comuniquen entre sí de forma eficiente. En el carro autónomo se utilizan por ejemplo para la cámara de profundidad.

Es importante para la comunicación entre nodos que haya un master, el cual se ejecuta mediante:

```
roscore
```

El cual ya es llamado de forma automática con `autostart.sh`

Los nodos se pueden arrancar de forma individual con:

```
rosrun package-name executable-name
```

O incluirlos dentro de un launch

## Simulador

[https://github.com/ITAM-Robotica/Eagle\\_Knights-Wiki/wiki](https://github.com/ITAM-Robotica/Eagle_Knights-Wiki/wiki)

Si gazebo truena al correr:

```
roslaunch autonomos_gazebo_simulation empty_autonomos.launch
```

usar:

```
killall gzserver  
killall gzclient
```

Es posible que hagan falta instalar algunas bibliotecas como ignition fuel, msgs y transport:

```
sudo apt install libignition-msgs-dev  
sudo apt-get install libignition-transport4-dev
```

Se diseño un control sencillo de posición. El simulador ya cuenta con un tópico que da la posición del robot en 2D llamado Pose2D, el cual da x, y, theta. Datos que se pueden obtener idealmente de una cámara que visualice al robot, u otro tipo de nodo como GPS, o usando navegación con la cámara ojo de pescado y marcas en el techo.

El simulador ya contiene algunos mundos para hacer pruebas con el carro. Estos se encuentran en la carpeta de autonomos\_gazebo\_simulation, donde al ejecutarse los launch, lanzan también el programa de manual\_control para el carro. Dentro de la carpeta, se encuentran tres folders. En launch son los archivos ejecutables, dentro de ellos se llama siempre a empty\_world.launch y el .world respectivo. En models contienen los archivos para

simular el carro y sus módulos, y las secciones de pista. Por último, en worlds a partir de los modelos, se crean las pistas y se agrega el modelo del carro.

Por la versión de ros se requiere hacer unos cambios para que aparezca la pista completa, de lo contrario únicamente se agrega la primera ocurrencia de un modelo. Para ello, a cada pedazo de pista se debe asignar un identificador con <name>a1</name>.

```
<include>

    <uri>model://straight_road</uri>

    <name>a13</name>

    <pose>.5 -2.5 0.1 0 0 0 </pose>

</include>
```

El procedimiento sería similar para agregar un segundo vehículo:

```
<include>

    <uri>model://AutoNOMOS_mini2</uri>

    <name>a1</name>

    <pose>1.75 0 0.16 0 0 1.5707 </pose>

</include>
```

Pero, como se observa, el modelo para el segundo vehículo se llama AutoNOMOS\_mini2. Esto se debe a que los módulos de la cámara y el laser son independientes al carro, y no se duplicarían. Como solución, en models se copian los archivos de AutoNOMOS\_mini, y en model.sdf se deben asignar otros nombres a los tópicos.

Para el LiDAR, en la línea 517:

```
<topicName>/scan2</topicName>
```

Para la cámara, a partir de la línea 590:

```
<cameraName>app2</cameraName>

<imageTopicName>camera2/rgb/image_raw</imageTopicName>

<cameraInfoTopicName>camera2/rgb/camera_info</cameraInfoTopicName>

<depthImageTopicName>camera2/depth/image_raw</depthImageTopicName>

<depthImageCameraInfoTopicName>camera2/depth/camera_info</depthImageCameraInfoTopicName>

<pointCloudTopicName>camera2/points</pointCloudTopicName>
```

Esto modifica los tópicos. Para la cámara y laser se usan los nombres definidos anteriormente. Para el resto de tópicos, se añade el identificador dado al modelo del carro en el archivo .world, por ejemplo:

```
/a1/manual_control/speed
```

## Odometría

Odometry está programado para resetearse cada x tiempo. Mide el ángulo con un acelerómetro, por las vibraciones del robot produce lecturas incorrectas, por lo que no es recomendable usar el control con odometría.

```
/odom
```

Mensaje tipo nav\_msgs/odometry. Publica la pose, orientación, giro linear y angular, al momento de iniciar el carro establece como punto 0,0,0. Si se desplaza comienza a contar en x, según el modelo ackerman y de bicicleta. No registra cuando se mueve de reversa

/tf

Transforma odometria al base\_link. Tanto odom como tf tienen los mismos valores, porque no hay ninguna rotación ni desplazamiento entre los frames base\_link y odom.

### Nodos Programados

Dentro de /catkin\_ws/src

```
catkin_create_pkg nombre roscpp geometry_msgs sensor_msgs std_msgs
```

Agregar dentro de la carpeta en source los nodos

En CMakeLists buscar la parte para ejecutar nodos C++ y agregar:

```
add_executable(nombre src/nombre.cpp)  
target_link_libraries(nombre ${catkin_LIBRARIES})
```

Si faltara algún paquete, en Package.xml:

```
<build_depend>package</build_depend>  
<run_depend>package</run_depend>
```

Y en CMakeLists:

```
catkin_package(  
    CATKIN_DEPENDS  
        nav_msgs  
        std_msgs)
```

Posteriormente, sobre catkin\_ws ejecutar:

```
catkin_make
```

Una vez ejecutado catkin\_make puede usar:

```
catkin_make --only-pkg-with-deps nombre
```

En el caso de Python únicamente requiere hacerlo ejecutable con chmod y correr:

```
source devel/setup.bash
```

Para correr los nodos debe ejecutar autosart y con rosrun ejecutar cada uno de los nodos.

Se presentan los nodos programados que se pueden encontrar en el vehículo dentro del paquete vision\_camara y en [Github](#): corresponden a programas tanto del proyecto para la detección y rastreo, como ejemplos para usar la cámara y transformadas.

- validacion.cpp

Programa para entrenar las máquinas de soporte vectorial, hacer la validación con un set de imágenes y poder visualizar los elementos extraídos por HOG.

Paquetes: Hace uso de HOGImage y OpenCV

- detecRastreoSim.cpp

Programa que funciona sobre el simulador del vehículo autónomo, realiza la detección y rastreo de vehículos en imágenes proporcionadas por la cámara realsense y hace la publicación de las detecciones. El simulador corre en cualquier versión de ROS, en este caso se usó Noetic, y la versión de OpenCV es la 4.0.

Suscribe: /app/camera/rgb/image\_raw

Publica: /detec, /pred y /detecciones

Los primeros publican imágenes con recuadros para marcar las detecciones y la siguiente predicción de Kalman, respectivamente. El último, usa un mensaje personalizado, para comunicar todas las detecciones y sus 10 siguientes predicciones de cada una.

Paquetes: Se usa OpenCV4, cv\_bridge, sensor\_msgs,

Se requieren importar los paquetes al programa, e incluirlos como se menciona en el tutorial de ROS a los archivos CMakeLists y package.xml, todos los nodos requieren importar también roscpp.

- detecRastreoC.cpp

Programa que funciona en el vehículo, igualmente hace la detección y rastreo, en este caso el carro tiene ROS Indigo, y la versión de OpenCV es la 2.4.

Suscribe: /app/camera/rgb/image\_raw

Publica: /detec, /pred y /detecciones

Paquetes: OpenCV2.4, sensor\_msgs, cv\_bridge

- detecRastreo.cpp

Este programa no es de ROS, se usa para las pruebas pasándole una carpeta con imágenes tomadas por una cámara y devuelve otras dos carpetas de imágenes con las detecciones y las predicciones.

Paquetes: OpenCV4

- validKalman.cpp

Permite realizar pruebas para ajustar el filtro de Kalman.

## **APÉNDICE 2: TUTORIAL AUTOMODELCAR**

En este capítulo se presenta una guía para el uso de ROS y OpenCV enfocado a la programación del vehículo. Para comenzar hablemos de ROS o “Robotic Operating System”, el cual se trata de un *middleware* que “provee bibliotecas y herramientas para ayudar a los desarrolladores de software a crear aplicaciones para robots. ROS provee abstracción de hardware, controladores de dispositivos, bibliotecas, herramientas de visualización, comunicación por mensajes, administración de paquetes, entre otras cosas, siendo *open source*” (“es - ROS Wiki” s. f.). La plataforma principal para usar ROS es en distribuciones de Linux como Ubuntu, y permite la programación en C++ y Python.

### **Guía de ROS**

Tomando de base el libro “A gentle introduction to ROS” de Jason M. O’Kane (O’Kane 2014). A continuación, se realiza una pequeña guía introductoria de ROS con los comandos principales, así como la descripción y estructura para lograr una mejor comprensión del programa.

### **Bibliotecas**

Para empezar, es importante mencionar que los paquetes instalados por *apt-get* para ros y otros paquetes externos como los *nodelet* de la cámara, tienen su raíz en */opt/ros/indigo*. Los ejecutables se almacenan en el subdirectorio *lib*, al igual que los archivos *include*. Cuando los necesita, ROS encuentra estos archivos buscándolos en los directorios listados en la

variable de entorno *CMAKE\_PREFIX\_PATH*, que se establece en el *setup.bash*. Para ver los archivos en un directorio se puede usar:

```
Rosls nombre_del_paquete
```

### **Catkin workspace**

Dentro de nuestro ambiente se tiene un *workspace* o un folder que permite modificar, construir e instalar paquetes de *catkin* (“*catkin/workspaces - ROS Wiki*” s. f.). Si bien no es requerido, facilita la organización. Dentro del folder normalmente se tienen las carpetas de *build*, *devel* y *src*. En esta última se agregan nuestros proyectos, estos paquetes *catkin* son los que podemos ejecutar como nodos.

### **Nodos**

Uno de los objetivos básicos de ROS es permitir el diseño de software como una colección de pequeños programas, en su mayoría independientes, llamados nodos, que se ejecutan todos al mismo tiempo. Para que esto funcione, esos nodos deben poder comunicarse entre sí. La parte de ROS que facilita esta comunicación se llama ROS master y, para iniciar el nodo maestro, se utiliza:

```
roscore
```

Mientras que los nodos se pueden arrancar de forma individual con:

```
rosrun nombre_del_paquete nombre_del_ejecutable
```

Como para el programa hecho:

```
rosrun vision_camara kalman
```

Los nombres de los nodos no son necesariamente los mismos que los nombres de los ejecutables de esos nodos. Se puede establecer explícitamente el nombre de un nodo como parte del *rosrun command*:

```
rosrun nombre_del_paquete nombre_del_ejecutable  
    __name:=nombre_del_nodo
```

Esto es útil si se quieren correr múltiples nodos del mismo tipo. Igualmente se pueden arrancar múltiples nodos de forma simultánea mediante un *launch*, como en la figura A2-1.

En el vehículo se tiene el programa *autostart* que se encarga ya de correr el *roscore*, llamar a las variables de entorno y correr un *roslaunch* con todos los nodos necesarios para manejar el vehículo, llamado *manual\_control*:

```
roslaunch manual_control manual_odroid.launch
```

```
1 <launch>  
2     <include file="$(find motor_communication)/launch/motor_odroid.launch"/>  
3     <include file="$(find send_steering_light)/launch/servo_light_odroid.launch"/>  
4     <include file="$(find heading)/launch/heading_odroid.launch"/>  
5     <include file="$(find odometry)/launch/odometry.launch"/>  
6     <include file="$(find realsense_camera)/launch/realsense_sr300.launch"/>  
7     <!--include file="$(find auto_stop)/launch/auto_stop.launch"-->  
8     <include file="$(find rplidar_ros)/launch/rplidar.launch"/>  
9 </launch>
```

FIGURA A2-1 EJEMPLO DE ROSLAUNCH CORRESPONDIENTE A  
MANUAL\_CONTROL

Dicho *launch* corre los nodos que se encargan de controlar los motores, la comunicación con el Arduino, los nodos para mandar la velocidad y el *steering*, así como el LiDAR, las cámaras, entre otros. Como se observa en la imagen, llama a buscar los paquetes dentro de la variable de *path*, y arranca otros archivos *.launch*, para llamar a los nodos.

Para llamar a algún nodo, dentro de un archivo *launch* se escribe:

```
<node pkg="nombre_del_paquete" type="nombre_del_nodo"  
      name="nombre_al_ejecutarse"> </node>
```

Por ejemplo, para el nodo *auto\_stop*:

```
<node pkg="auto_stop" type="auto_stop_node" name="auto_stop"  
      output="screen">  
  
    <param name="angle_front" type="int" value="35" />  
  
    <param name="angle_back" type="int" value="35"/>  
  
    <param name="break_distance" type="double"  
          value="0.6"/>  
  
</node>
```

## Tópicos y mensajes

Al tener ROS un enfoque distribuido, los diferentes nodos deben comunicarse con otros compartiendo mensajes bajo un mismo contexto llamado tópico. Para hacerlo cada nodo debe suscribirse y publicar los mensajes que le interesan, esta información se puede ver con el comando:

```
rosnode info nombre_del_nodo
```

Para ver los nodos ejecutándose, como en la figura A2-2, se usa el comando:

```
rostopic list
```

```
# 18:02:20 Modelcar-126:~> rostopic list
/SR300_nodelet_manager/bond
/app/camera/rgb/image_raw
/app/camera/rgb/image_raw/compressed
/app/camera/rgb/image_raw/compressed/parameter_descriptions
/app/camera/rgb/image_raw/compressed/parameter_updates
/app/camera/rgb/image_raw/compressedDepth
/app/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/app/camera/rgb/image_raw/compressedDepth/parameter_updates
/app/camera/rgb/image_raw/theora
/app/camera/rgb/image_raw/theora/parameter_descriptions
/app/camera/rgb/image_raw/theora/parameter_updates
/camera_nodelet/parameter_descriptions
/camera_nodelet/parameter_updates
/depth/camera_info
/depth/image_raw
/depth/image_raw/compressed
/depth/image_raw/compressed/parameter_descriptions
/depth/image_raw/compressed/parameter_updates
/depth/image_raw/compressedDepth
/depth/image_raw/compressedDepth/parameter_descriptions
/depth/image_raw/compressedDepth/parameter_updates
/depth/image_raw/theora
/depth/image_raw/theora/parameter_descriptions
/depth/image_raw/theora/parameter_updates
/depth/points
/detec
```

FIGURA A2-2 SE MUESTRAN ALGUNOS TÓPICOS RELACIONADOS A LA CÁMARA DEL CARRO, AL CORRER MANUAL\_CONTROL

O se puede ver información específica de los mensajes, como en la figura A2-3, con *rostopic*.

El comando *echo* que permite ver los mensajes reales que se están publicando sobre un solo tópico:

```
rostopic echo nombre_del_tópico
```

FIGURA A2-3 MENSAJE PUBLICADO EN EL TÓPICO /SCAN RELACIONADO AL LIDAR

Este comando descargará todos los mensajes publicados sobre la terminal.

Para desplegar más información sobre un tópico se puede usar el comando *info*. El cual devuelve el tipo de mensaje, quiénes son sus publicadores y quiénes son los suscriptores, así como la frecuencia a la que se publican, un ejemplo se muestra en la figura A2-4:

```
rostopic info nombre_del_tópico
```

```
israel@ubuntu:~/EK_AutoNOMOS_Sim$ rostopic info /app/camera/rgb/image_raw
Type: sensor_msgs/Image

Publishers:
 * /gazebo (http://ubuntu:42829/)

Subscribers:
 * /detec_vehiculos (http://ubuntu:41789/)
```

FIGURA A2-4 EL TÓPICO /APP/CAMERA/RGB/IMAGE\_RAW CORRESPONDE A LA IMAGEN A COLOR PUBLICADA POR EL NODO DE LA CÁMARA, SE LEE PARA LA DETECCIÓN

Este comando, además, nos permite publicar de forma directa a algún tópico, aunque lo ideal sería realizarlo dentro de un nodo.

```
rostopic pub frecuencia_en_hz nombre_del_topico tipo_de_mensaje
```

```
{contenido: }
```

El cual repite el mensaje a una frecuencia dada. Por ejemplo, se puede mandar la velocidad al carro con el mensaje:

```
rostopic -r 10 pub /manual_control/speed std_msgs /Int16  
{"data: -1000"}
```

En ROS los mensajes publicados cuentan con un formato con el que se deben escribir, lo que permite una mayor estandarización. Uno puede definir nuevos tipos de mensajes o aprovechar de las bibliotecas que tienen tipos de mensajes ya definidos. Se puede visualizar el formato del mensaje con el comando:

```
rosmsg show nombre_del_mensaje
```

Los mensajes típicamente usan campos con datos compuestos de otros en una forma de anidamiento, por ejemplo: geometric\_msgs/Vector3, cuyos mensajes constan a su vez de tres campos que representan a: “x”, “y” y “z”.

Con *rosmsg show* se visualiza como están compuestos los campos, junto a los tipos y su identación. Además, pueden contar con arreglos de longitud fija o variable, y constantes. Por ejemplo, en la figura A2-5 se muestra el mensaje LaserScan.

```

lsrael@ubuntu:~/EK_AutoNOMOS_Sim$ rosmsg show sensor_msgs/LaserScan
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities

```

FIGURA A2-5 COMPOSICIÓN DEL MENSAJE TIPO LASERSCAN PARA EL LIDAR

### Creación de mensajes

Se presenta un ejemplo de mensaje personalizados. Para la publicación de las detecciones del vehículo, se realiza en forma de arreglos, donde su contenido es similar a la estructura rectángulo de OpenCV. En él se incluyen x, y, ancho y altura. Para esto, en el paquete que los utiliza, se debe crear un folder para los mensajes “msg”, dentro se colocan los archivos tipo .msg. Para las detecciones usamos detec.msg, detecArray.msg y detecTiempos.msg.

Tenemos tres mensajes. En el primero se encuentra la estructura de una solo detección. En el segundo es el que se usa para publicar diferentes predicciones a lo largo del tiempo. Y el último para mandar un arreglo con las detecciones de diferentes vehículos:

detec.msg	detecTiempos.msg	detecArray.msg
uint8 x	detec posA	uint8 numDetec
uint8 y	detec[] posSig	detecTiempos[] array
uint8 width		
uint8 height		

Para compilarlo en package.xml se agrega:

```
<build_depend>message_generation</build_depend>  
<run_depend>message_runtime</run_depend>
```

En CMakeLists:

```
find_package(catkin REQUIRED COMPONENTS  
    message_generation  
)  
add_message_files(  
    FILES  
    detec.msg  
    detecTiempos.msg  
    detecArray.msg  
)
```

En generate messages se agregan los tipos de menajes que se usan:

```
generate_messages(  
    DEPENDENCIES  
    std_msgs  
)  
catkin_package(  
    CATKIN_DEPENDS geometry_msgs roscpp std_msgs  
    message_runtime  
)
```

Una vez compilado, para usarlo en un programa se debe llamar con include:

```
#include "detecMSG/detec.h"  
#include "detecMSG/detecArray.h"
```

## Workspace y paquetes

Para crear un *workspace* primero se debe crear el directorio con *mkdir*. Lo siguiente es crear una carpeta *src* para los paquetes, desde ella se usa el comando:

```
catkin_create_pkg nombre_del_paquete
```

El cuál únicamente crea el directorio que llevará los nodos, junto con las carpetas de *build* y *devel*. Además se agregan dos archivos de configuración.

```
catkin_create_pkg nombre_del_paquete rospy roscpp sensor_msgs
```

Permite durante la creación, incluir las bibliotecas que se utilizarán, como *rospy* que es para Python y *roscpp* para C++.

Los archivos que se crean son *package.xml*, el cual es el archivo de configuración donde se agregan las dependencias. Y *CMakeLists.txt* que contiene una lista de instrucciones, como que ejecutables deben crearse, qué archivos fuente usar, dónde encontrar los archivos y bibliotecas que se usarán. Es el archivo *package.xml* el que lo identifica como un paquete de ROS.

Si se quiere agregar nuevas dependencias se debe modificar:

- Package.xml

Por ejemplo, para agregar *cv\_bridge* se incluye:

```
<build_depend>cv_bridge</build_depend>
<build_export>cv_brdige</build_export>
<exec_depend>cv_bridge</exec_depend>
```

El primero para compilar, el segundo para exportarlo y el último para ejecutarlo.

También se debe modificar el archivo *CMakeLists* en:

```
find_package(
    cv_bridge
)
```

Se recomienda usar *catkin\_make* para compilar antes de empezar a escribir. Con ello generará cabeceras, dependencias y bibliotecas.

## Programar un nodo

ROS permite programar nodos tanto en C++ como en Python. En el caso de programar en C++, se debe incluir lo siguiente:

- El encabezado *ros/ros.h*
- La función *ros::init*
- Y *ros::NodeHandle* que registra el programa como un nodo.

En el caso de Python:

- `import rospy`
- `rospy.init_node('talker', anonymous=True)`
- `rospy.resolve_name(name, caller_id=None)`

Este último, equivale a *NodeHandle*, pero para Python no es necesario.

## Suscripciones

Para leer un mensaje se requiere declarar un suscriptor, el cual llama a una función a la que le envía el mensaje como parámetro, se llaman funciones de *Callback*:

```
ros::Subscriber camara_sub =
nh.subscribe("/app/camera/rgb/image_raw", 1,
camaraRGBCallback);
```

Dicha instrucción se escribe en el *main*, recibe el mensaje publicado en el tópico */app/camera/rgb/image\_raw* que corresponde a la cámara, y llama a la función *camaraRGBCallback*, el 10 corresponde al tamaño del buffer de los mensajes.

```
void camaraRGBCallback(const sensor_msgs::Image& msg) {}
```

Como se ve, en la función de *callback* recibe el mensaje, aquí se debe declarar el tipo, en este caso es un *sensor\_msgs::Image* (previamente se debía haber incluido *#include <sensor\_msgs/Image.h>*).

## Publicar

Para publicar se requiere agregar el tipo de mensaje con `#include`. Aquí se requiere declarar un *Publisher* y necesita del `ros::NodeHandle`. Por lo que, si se quiere publicar desde dentro de una función es mejor declarar el *Publisher* como variable global:

```
ros::Publisher detec_publisher;
```

E incluir dentro de la función:

```
ros::NodeHandle nh("~");
detec_publisher =
    nh.advertise<sensor_msgs::Image>("/detec", 1);
```

Se inicializa el *Publisher* con el tipo de mensaje, dentro el tópico o nombre con que se puede leer: `/detec`.

Para publicarlo bastaría con crear una variable del mismo tipo como `sensor_msgs::Image nombre`, y usar `detec_publisher.publish(nombre)`.

```
sensor_msgs::Image img_msg;
detec_publisher.publish(img_msg);
```

Para usar OpenCV se requiere importar los módulos correspondientes a las funciones que se usarán y `cv_bridge`:

```
#include "opencv2/opencv.hpp"
#include <cv_bridge/cv_bridge.h>
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/ml/ml.hpp"
```

Por un lado, `Cv_bridge` permite traducir una imagen de ROS a tipo `math` para OpenCV:

```
Mat img;
```

```

cv_bridge::CvImagePtr cv_ptr;
cv_ptr = cv_bridge::toCvCopy(msg,
                            sensor_msgs::image_encodings::MONO8);
img = cv_ptr->image.clone();

```

En *encodings* se puede convertir al formato deseado. En este caso es una imagen a blanco y negro, y cada pixel con un valor de 8 bits (0 a 255).

Por otro lado, para convertir una imagen tipo math a ROS, se requiere crear una variable del tipo del mensaje y llenar sus campos, como el encabezado.

```

sensor_msgs::Image img_msg;
std_msgs::Header header;
//header.seq = counter;
header.stamp = ros::Time::now();
img_bridge =
    cv_bridge::CvImage(header,sensor_msgs::image_encodings::TYPE_8UC1, img1);
img_bridge.toImageMsg(img_msg);
dete_publisher.publish(img_msg);

```

En el código anterior, se muestra cómo se convierte y se publica una imagen.

## **Compilar un nodo**

Por un lado, con Python hay que asegurarse que el archivo es ejecutable, desde la consola de comandos se aplica:

```
chmod u+x
```

Una vez finalizado se ejecuta:

```
source devel/setup.bash
```

Que debería establecer variables que le permiten a ros encontrar el paquete y sus ejecutables.

Por otro lado, si el nodo está escrito en C++, primero debe agregarse dentro de CMakeLists como un ejecutable:

```
add_executable(nombre_del_nodo fuente_del_nodo)
target_link_libraries(nombre_del_nodo ${catkin_LIBRARIES})
${OpenCV_LIBRARIES}
```

La primera instrucción declara el nombre del ejecutable y una lista de archivos fuente que lo componen. En caso de ser más de uno se separan con espacios. La segunda instrucción le indica a *CMake* que el nodo está ligado a las bibliotecas definidas en *find\_package*. Además, para el uso de OpenCV, en build include directories se agrega:

```
${OpenCV_INCLUDE_DIRS}
```

Posteriormente se debe usar:

```
catkin_make
```

*catkin\_make* volverá a compilar todo el proyecto. Se puede usar para compilar un solo paquete cuando se quiera reflejar pequeños cambios en un programa (solo cuando el paquete ya fue construido anteriormente con *catkin\_make*):

```
catkin_make --only-pkg-with-deps <paquete>
```

Aunque cmake falle, salvo que se deba a un error de código, se debe correr cuantas veces sea necesario hasta que finalice sin errores. Esto es para que agregue todos los paquetes.

Al finalizar se debe usar también:

```
source devel/setup.bash
```

## Transformaciones

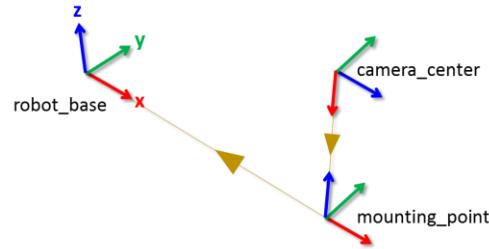


FIGURA A2-6 CONEXIÓN CON TRANSFORMACIONES DE LAS COORDENADAS DE LA CÁMARA CON EL VEHÍCULO. FUENTE (“ACCESS THE TF TRANSFORMATION TREE IN ROS - MATLAB & SIMULINK - MATHWORKS AMÉRICA LATINA” S. F.)

Para cada objeto tanto del vehículo, como un punto detectado por una cámara o el LiDAR se puede representar su posición dentro de un *frame* o sistemas de coordenadas en 3D. Estos *frames*, como se observa en la figura A2-6, se pueden relacionar entre sí mediante transformaciones que se componen de traslaciones y rotaciones.

Entonces, con el objetivo de poder visualizar los datos obtenidos por los sensores todos juntos y en la posición correcta, se requiere asignarles *frames* de coordenadas a cada sensor respecto

al carro, y mediante transformaciones unirlos al sistema de coordenadas del vehículo, llamado *base\_link*.

Los sensores al publicar mensajes de posición ya cuentan con sus propios *frames*:

Odometría: */odom*

Realsense camera: */SR300\_depth\_optical\_frame*

LiDAR: */laserScan*

Estos *frames* pueden visualizarse por separado, y partiendo de que están centrados en las coordenadas (0,0). Para ello en rviz se debe cambiar el *frame* global a cada uno y agregar el respectivo tópico que quiere visualizarse.

Por defecto en los paquetes del vehículo, el nodo Odometry ya cuenta con un nodo de transformadas tf, que para visualizar en rviz se debe hacer referencia al *frame* *base\_link*

Global options

Fixed frame

Cambiar a *base\_link* o *odom*

Cambiar a *laser* para ver *laserScan*

*SR300\_depth\_optical\_frame*

Para ver la cámara en el espacio 3D

(En el caso de la cámara mediante la opción de *add by topic* se pueden visualizar los mensajes tipo raw de color, IR y depth.)

*base\_link frame*: Está unido al robot, tal que es fijo, la x apunta al frente del carro. Se encuentra centrado y se mueve con él.

Frames de los sensores:

- *Base\_camara*: para la cámara de profundidad, se debe medir la distancia x, y, z que tiene la cámara respecto al centro del robot. Además, se deben hacer unas rotaciones: 1.57 (90°) en roll y 1.57 en pitch.
- *Base\_laser*: Para el LiDAR, no requiere rotaciones, solo traslación. Por lo que se debe medir x, y, z respecto al centro del robot.
- *Odom*: frame que debe ser continuo y evolucionar junto a la pose, lo ideal es que se publique a altas frecuencias para evitar saltos discretos. Representa el inicio del movimiento.

### **Estructura del programa de las transformadas**

Para las transformaciones se requiere de un *broadcaster*, el cual usa 5 argumentos:

- La rotación especificada en cuaterniones
- Un vector de tres dimensiones que representa la traslación (x,y,z)
- Una marca de tiempo, normalmente se usa la del sistema: `ros::Time::now()`

- Nombre del frame padre, si es con respecto al carro: “base\_link”
- Nombre del frame hijo que se crea, como “base\_laser” para el LiDAR

Dentro de un programa:

```
#include <tf/transform_broadcaster.h>
```

Se crea el mensaje

```
tf::TransformBroadcaster laser_broadcaster;
ros::Time current_time, last_time;
current_time = ros::Time::now();
last_time = ros::Time::now();
geometry_msgs::Quaternion
```

Para dar la rotación en cuaterniones, se puede simplemente llamar a una rutina de *transform* y dar el ángulo:

```
laser_quat = tf::createQuaternionMsgFromYaw(th);
```

Se rellenan los campos del mensaje, el cual es de tipo *TransformStamped* y se encuentra en *geometry*:

```
geometry_msgs::TransformStamped laser_transf;
laser_transf.header.stamp = current_time;
laser_transf.header.frame_id = "base_laser";
laser_transf.child_frame_id = "base_link";
```

```

laser_transf.transform.translation.x = x;
laser_transf.transform.translation.y = y;
laser_transf.transform.translation.z = 0.0;
laser_transf.transform.rotation = laser_quat;

```

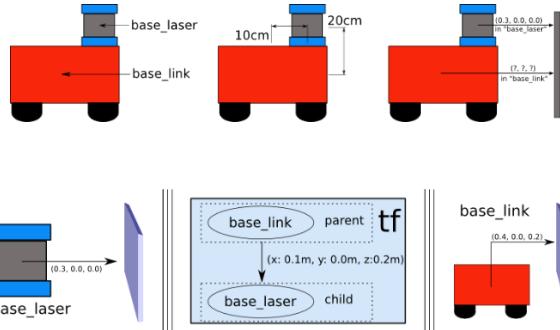


FIGURA 0-7 RESULTADO DE LA CONEXIÓN DE LOS FRAMES. FUENTE (“SETTING UP TRANSFORMATIONS — NAVIGATION 2 1.0.0 DOCUMENTATION” S. F.)

Como en la figura A2-7, se le da los valores en xyz de la traslación, la rotación, el nombre del frame y el frame padre.

```

laser_broadcaster.sendTransform(laser_transf);

```

Esta es la transformada de un frame a otro. Falta entonces publicar el mensaje con la posición, esto se hace asociando el mensaje al frame. Como ventaja, los mensajes de ROS cuentan con un campo header en el cual se puede asociar. Por ejemplo, un mensaje de tipo laser, con frame\_id al frame creado, el cual ya tiene una transformada para conectarlo al vehículo:

```

sensor_msgs::LaserScan tflaser;
tflaser.header.frame_id = "base_laser";

```

## **Uso de la cámara**

A continuación, se describe el uso de la cámara de profundidad realsense. Dentro de los primeros pasos realizados fue programar un par de nodos con la tarea de localizar una marca de algún color y el control para moverse a dicha marca. El objetivo fue comprender el funcionamiento de la cámara y cómo extraer los datos de la nube de puntos. Para comenzar, la cámara realsense publica los siguientes tópicos:

- Cámara de color en formato RGB

`camera/color/image_raw`

- Cámara de profundidad con distancias en mm.

`camera/depth/image_raw`

- Nube de puntos en formato XYZRGB, junta la imagen de profundidad con la de color.

`camera/depth/points (sensor_msgs/PointCloud2)`

- Para las cámaras de infrarrojo:

`camera/infrared1/image_raw`

`camera/infrared2/image_raw`

Los tipos `image_raw` publican mensajes del tipo `sensor_msgs/Image`. Además, están los tipos `Info` que publican datos para la calibración.

El tópico Depth/points incorpora la información de las cámaras infrarroja y de color. Se debe tomar en cuenta que la información publicada es incompleta. Mezcla en un mensaje de tipo xyzrgb la información de la cámara de profundidad con la argb. En caso de que falte algún dato de profundidad, su correspondiente color de ese píxel tampoco se incluye. Estos valores están llenos de 0 y en r,g,b tienen un valor de 96.

Para extraer la información de la nube de puntos hay varias alternativas. Con Python se puede convertirlo con numpy asarray para separar en x, y, z, argb. Para separar los colores el problema es que se trata de un proceso muy lento usando unpack y pack de bytes. Se puede hacer la separación en C++ con reinterpret\_cast que accede al lugar de memoria y lo maneja como si se tratara de otro tipo de variable.

Desafortunadamente no hay una mejor opción para realizar este proceso en python. En cambio, en C++ se pueden usar unas bibliotecas:

```
#include <pcl_ros/point_cloud.h>
#include <pcl/point_types.h>
#include <pcl_conversions/pcl_conversions.h>
#include <pcl/conversions.h>
```

Lo que se hace es cambiar del tipo de mensaje sensor\_msgs::PointCloud2 a pcl::PointCloud<XYZRGB> ("Point Cloud Library (PCL): pcl::PointXYZRGB Struct Reference" s. f.) que permite acceder más fácilmente a los valores:

```
pcl::PCLPointCloud2 pcl_pc2;
```

```

pcl_conversions::toPCL(msg,pcl_pc2);

pcl::PointCloud<pcl::PointXYZRGBA>::Ptr pt_cloud(new
pcl::PointCloud<pcl::PointXYZRGBA>);

pcl::fromPCLPointCloud2(pcl_pc2,*pt_cloud);

```

Entonces por medio de un ciclo se puede acceder para cada punto de la imagen a su valor correspondiente:

```

for(int i = 0; i < height*width; ++i) {

    xA.push_back(pt_cloud->points[i].x);
    yA.push_back(pt_cloud->points[i].y);
    zA.push_back(pt_cloud->points[i].z);
    r.push_back(pt_cloud->points[i].r);
    g.push_back(pt_cloud->points[i].g);
    b.push_back(pt_cloud->points[i].b);
}

```

En cambio, para usar las imágenes de tipo sensor\_msgs/Image se puede hacer uso de cv\_bridge para convertirlas a una matriz de OpenCV:

```

cv_bridge::CvImagePtr cv_ptr;
cv_ptr = cv_bridge::toCvCopy(msg,
sensor_msgs::image_encodings::MONO8);
Mat img = cv_ptr->image.clone();

```

msg es la lectura recibida del tópico, y se define el formato al que se convierte la imagen, en este caso MONO8 es una imagen en blanco y negro, donde cada píxel vale 8 bits.

Para publicar una imagen se usa el camino inverso:

```
sensor_msgs::Image img_msg;  
std_msgs::Header header;  
header.stamp = ros::Time::now();  
img_bridge = cv_bridge::CvImage(header,  
sensor_msgs::image_encodings::TYPE_8UC1, img);  
img_bridge.toImageMsg(img_msg);  
detec_publisher.publish(img_msg);
```

Se crea el tópico y se le agregan encabezados, con cv\_bridge se elige un encoding que transforme la imagen, y al final se puede publicar.