

PROGRAMA ACADÉMICO DE INGENIERÍA Mecatrónica y Telecomunicaciones

**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 y Telecomunicaciones

P R E S E N T A: Israel Fonseca Zárate

A S E S O R: Rafael Gregorio Gamboa Hirales

Agosto-Diciembre 2023

Resumen

Dentro de la investigación realizada sobre vehículo autónomos, un punto importante es la detección y el seguimiento de otros participantes en el tráfico que puedan ser riesgos potenciales, no solo para nuestro vehículo, pero también para otros en casos donde se encuentren en puntos ciegos, razón por la cual se habla de lograr un entorno colaborativo entre vehículos autónomos. Como objetivo del presente trabajo se presenta una implementación sobre el AutoModelCar de un sistema de detección y rastreo de vehículos, desde un enfoque que requiera poco poder de procesamiento y logre resultados cercanos a tiempo real, 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 v2v, para ello se simula y experimenta con el protocolo AODV y el estándar DRSC. Si bien en vehículos autónomos el método indiscutible son las redes neuronales, con los métodos usados se logró alcanzar un buen desempeño que, aunque no reemplaza las redes neuronales, sin duda es una opción que puede ser relevante para aplicaciones orientadas a tiempo real y con bajo poder de procesamiento. Mientras que en redes vanet, se experimento 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 entre el envío de mensajes y un bajo RTT, como se muestra, el modelo v2v puede funcionar si toma en cuenta las condiciones de tráfico, para adaptar parámetros como la tasa de envíos, donde los principales retos son la velocidad de los nodos, y situaciones con una alta densidad de nodos, este último, se puede mejorar si se añade el uso de infraestructura en lugares como ciudades, pasando a un modelo v2x, donde esta solo cumpla un rol de coordinador, y que aproveche las ventajas de una comunicación directa entre vehículos, como son la latencia y cobertura geográfica.

**Palabras clave**: Maquinas de soporte vectorial, Filtros de cascada, Filtro de Kalman, Protocolo AODV, Estándar IEEE 802.11p (DSRC).

# ÍNDICE

[ÍNDICE 4](#_Toc140412879)

[ÍNDICE DE FIGURAS 7](#_Toc140412880)

[ÍNDICE DE Tablas 12](#_Toc140412881)

[1. Introducción 13](#_Toc140412882)

[1.1. Contexto 13](#_Toc140412883)

[1.2. Identificación problema 14](#_Toc140412884)

[1.3. Objetivos 16](#_Toc140412885)

[1.4. Metodología a usar 19](#_Toc140412886)

[2. Capítulo 2 20](#_Toc140412887)

[2.1. Estado del arte 20](#_Toc140412888)

[2.2. Alcance 23](#_Toc140412889)

[2.3. Requerimientos funcionales 26](#_Toc140412890)

[3. Capítulo 3: Detección de objetos 27](#_Toc140412891)

[3.1. Descripción 27](#_Toc140412892)

[3.2. Técnicas de detección 30](#_Toc140412893)

[3.1.2. Clasificador de cascada 31](#_Toc140412894)

[3.2.2. Hog+svm 34](#_Toc140412895)

[3.3.2. Comparación con redes neuronales convolucionales 38](#_Toc140412896)

[4. Capítulo 4: Rastreo de objetos 40](#_Toc140412897)

[4.1. Descripción 40](#_Toc140412898)

[4.2. Técnicas 40](#_Toc140412899)

[4.3. Filtro de Kalman 43](#_Toc140412900)

[4.3.1 Caso multivariable 44](#_Toc140412901)

[4.3.2 Forma general 46](#_Toc140412902)

[5. Capítulo 5 49](#_Toc140412903)

[5.1. Implementación 49](#_Toc140412904)

[5.1.1. Entrenamiento del filtro de cascada 49](#_Toc140412905)

[5.2.1. Entrenamiento de svm 52](#_Toc140412906)

[5.3.1. Detección y nodo para el carrito 59](#_Toc140412907)

[5.4.1. Rastreo 66](#_Toc140412908)

[5.2. Resultados 73](#_Toc140412909)

[6. Capítulo 6 Redes vanet 83](#_Toc140412910)

[6.1. Descripción 83](#_Toc140412911)

[6.2. Patrones de comunicación 85](#_Toc140412912)

[6.3. Modelos v2x, v2i y v2v 88](#_Toc140412913)

[6.4. DSRC 89](#_Toc140412914)

[6.1.4. Capa física 89](#_Toc140412915)

[6.2.4. Capa MAC 91](#_Toc140412916)

[6.3.4. Capa lógica 93](#_Toc140412917)

[6.4.4. Capas superiores 93](#_Toc140412918)

[6.5. Contenido de los mensajes 95](#_Toc140412919)

[7. Capítulo 7 Implementación de una red vanet 96](#_Toc140412920)

[7.1. Arquitectura y requisitos de la red vanet 96](#_Toc140412921)

[7.2. Implementación 97](#_Toc140412922)

[7.2.1 AODV 98](#_Toc140412923)

[7.2.2 IEEE 802.11p 101](#_Toc140412924)

[7.3 Resultados 107](#_Toc140412925)

[8 Conclusiones 133](#_Toc140412926)

[8.2 Resultados 133](#_Toc140412927)

[8.3 Discusión y mejoras 133](#_Toc140412928)

[9 Referencias 134](#_Toc140412929)

ÍNDICE DE FIGURAS

[Figura 5‑1 Ejemplos de vehículos 48](#_Toc140413050)

[Figura 5‑2 Ejemplos de imágenes negativas para el entrenamiento 49](#_Toc140413051)

[Figura 5‑3 imágenes de vehículos de lado 51](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413052)

[Figura 5‑4 imágenes de la parte trasera de vehículos 51](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413053)

[Figura 5‑5 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. 53](#_Toc140413054)

[Figura 5‑6 Con blocksize (64,64), cellsize y blockstride de (4,4). 54](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413055)

[Figura 5‑7 Con filtro gaussiano (5,5) 54](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413056)

[Figura 5‑9 Imagen del vehículo del simulador. 54](#_Toc140413057)

[Figura 5‑10 hog features de la carretera 54](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413058)

[Figura 5‑11 HOG features de un árbol 54](#_Toc140413059)

[Figura 5‑13 Imagen de la función main, la configuración inicial, hasta el inicio del ciclo 60](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413060)

[Figura 5‑16 Con ProcessNoisecov de 0.5, los puntos azules reaccionan más rápido y como consecuencia se disparan 70](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413061)

[Figura 5‑15 Con ProcessNoiseCov de 0.00005, Los puntos azules reaccionan más lento a los cambios 70](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413062)

[Figura 5‑23 Ejemplos de detecciones sobre frames, en la primer imagen se indentifican correctamente los dos vehículos, en el segundo por el cambio de iluminación no detecta uno de los vehículos, en la tercer imagen se muestra un falso positivo 77](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413063)

[Figura 5‑22 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 77](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413064)

[Figura 5‑21 Candidatos arrojados por el filtro de cascada 77](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413065)

[Figura 5‑24 Imagen del simulador desde gazebo, mapa con dos coches 79](#_Toc140413066)

[Figura 5‑25 Ejemplo de detecciones en el simulador, arriba se muestr la detección, abajo la predicción 80](#_Toc140413067)

[Figura 6‑1 [Hartenstein y Laberteaux 2010] Algunas aplicaciones y su modelo sugerido 87](#_Toc140413068)

[Figura 6‑2 [Hartenstein y Laberteaux 2010] Capas del estándar dsrc, se compone de dos partes principales, el estándar IEEE 802.11p e IEEE 1609 o wave 88](#_Toc140413069)

[Figura 6‑3 [«IDENTIFICACIÓN DE NECESIDADES DE ESPECTRO PARA SISTEMAS DE TRANSPORTE INTELIGENTE EN LA BANDA 5850-5925 MHz» 2021] 89](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413070)

[Figura 6‑4 OFDM en el dominio de la frecuencia [Cabrera y Tarrés, s. f.] 90](#_Toc140413071)

[Figura 6‑5 [Kenney 2011] encabezados de capa 2 91](#_Toc140413072)

[Figura 6‑6 [Kenney 2011] Encabezados de la subcapa llc y snap 92](#_Toc140413073)

[Figura 6‑7 [Kenney 2011] Formato de wave short message 93](#_Toc140413074)

[Figura 7‑1 Imagen de la simulación de aodv 100](#_Toc140413075)

[Figura 7‑2 Imagen final en sumo 101](#_Toc140413076)

[Figura 7‑3 Imagen de la simulación corriendo en omnet++, se observa un nodo retransmitiendo a todos un mensaje 106](#_Toc140413077)

[Figura 7‑4 Simulación 1 experimentando el desempeño con diferente densidad de nodos 106](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413078)

[Figura 7‑5 A la izquierda la posición inicial, a la derecha la posición final 109](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413079)

[Figura 7‑6 resultados de la simulación 1 110](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413080)

[Figura 7‑8 Rutas formadas por el protocolo aodv 110](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413081)

[Figura 7‑7 a la izquierda "in progress frames" a la derecha "pending queue" 111](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413082)

[Figura 7‑9 El nodo 6 en rojo quiere transmitir 123](#_Toc140413083)

[Figura 7‑10 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 retransmisiones quedan informados. 124](#_Toc140413084)

[Figura 7‑11 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 124](#_Toc140413085)

[Figura 7‑12 Posición final, el nodo 6 se incorpora 124](#_Toc140413086)

[Figura 7‑13 A la izquierda se observa la posición de partida de los nodos, a la derecha la posición final 125](file:///D:\ifons\Descargas\VersionCorta.docx#_Toc140413087)

ÍNDICE DE Tablas

[Tabla 6‑1 Matriz de confusión para svm con imágenes de la parte trasera de coches 73](#_Toc140413088)

[Tabla 6‑2 Matriz de confusión de svm con coches de lado 73](#_Toc140413089)

[Tabla 6‑3 Matriz de confusión resultante de juntar ambas svm 74](#_Toc140413090)

[Tabla 6‑4 Matriz de confusión de svm completa con la parte trasera de coches 74](#_Toc140413091)

[Tabla 6‑5 Matriz de confusión de svm completa con coches de lado 75](#_Toc140413092)

[Tabla 6‑6 Matriz de confusión de ambas svm completas 75](#_Toc140413093)

[Tabla 6‑7 Tiempos medidos de los detectores en los tres ambientes de pruebas 80](#_Toc140413094)

[Tabla 6‑1 Comparación entre manet, vanet y fanet 82](#_Toc140413095)

# Introducción

## Contexto

Los vehículos autónomos son uno de los temas que más han acaparado la atención en los últimos años, gracias a los notorios avances y muestras del progreso tecnológico que se vive actualmente, en diferentes áreas como la variedad de sensores con los que se pueden contar, el incremento en el poder computacional que ha permitido integrar la inteligencia artificial, así como el desarrollo de comunicaciones a alta velocidad, junto con bajas latencias como es 5G. Esto ha permitido la integración de soluciones que reducen la intervención humana a la hora de conducir vehículos, con el claro objetivo de reducir accidentes, mejorar la movilidad, comodidad y agilizar el transporte.

Este desarrollo en coches autónomos se ha visualizado desde mucho tiempo atrás, en 1939 General Motors presentaba un primer concepto de vehículo autónomo, era 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, con científicos intentando enviar vehículos a la Luna, los cuales ya incorporaban la idea de cámaras para procesar imágenes del camino. Desde entonces muchos programas de investigación gubernamentales o académicos han surgido, como “Darpa Grand Challenge” en 2004, una competencia de vehículos autónomos celebrada en el desierto de Mojave en Estados Unidos, en los que se han incorporado 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, junto con otras compañías como TESLA, Audi, BMW que se han unido en esta investigación. [1] [Faisal et al. 2019]

Dentro de los vehículos autónomos se tienen en cuenta diferentes niveles de autonomía, según los cuales varían 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, incluyendo una parte importante de adquisición de información que es resultado tanto del uso de sensores, como de la comunicación y negociación con otros vehículos o infraestructura.

## Identificación problema

Para este trabajo se presenta una implementación sobre la parte de percepción, para la identificación y rastreo de otros vehículos, mientras que también se aborda el problema de la comunicación presentando una simulación para una red vehicular.

Para la primera parte del problema, este involucra la implementación de algoritmos para detección de movimiento, un campo bien estudiado en caso de cámara estáticas, pero que presenta inconvenientes si la cámara está en movimiento, por lo que el problema requiere separarse en dos partes:

* Detección de objetos
* Rastreo de los objetivos

Sobre el primer punto, detección de objetos, se tiene que lidiar con múltiples dificultades como son vehículos a altas velocidades y distancias lejanas, bajo condiciones diferentes como el clima y la iluminación, llamada oclusión. Para ello soluciones o técnicas de aprendizaje de máquina son elegidas, gracias a que pueden manejar grandes volúmenes de datos.

Tomando en cuenta las capacidades del modelo a escala AutoModelCar se busca implementar un sistema de detección que funcione en tiempo real, y que sea ligero computacionalmente, para ello se plantea una combinación entre filtros de cascada (Haar) y máquinas de soporte vectorial con histograma orientado por gradiente (HOG+svm). Aun cuando, gracias al avance en la potencia computacional, el enfoque principal son las redes neuronales convolucionales, con la elección correcta de parámetros hog+svm puede tener un desempeño comparable.

En el rastreo de los objetivos, el enfoque dominante en la mayoría de las aplicaciones y mismo que se utiliza en el presente trabajo, es tomarlo como un problema de inferencia bayesiana, el cual estima la posición siguiente de un estado basado en la observación actual y estados previos, el estado posterior es actualizado siguiendo un modelo de movimiento, junto a una etapa de corrección [Janai et al. 2021] [2]. Uno de estos modelos es el filtro de Kalman, donde los principales problemas aquí son poder identificar que un objeto es el mismo entre dos observaciones distintas, junto con lidiar con detecciones incorrectas u observaciones faltantes que van acarreando errores en la predicción.

Por si solo un enfoque individual no es suficiente, se requiere de un ambiente cooperativo a través de comunicaciones, los vehículos autónomos requieren comunicarse entre sí, con la infraestructura, con la nube y otros dispositivos, llamado CAV (vehículos autónomos conectados). Esto requerirá del establecimiento de redes de comunicaciones robustas, seguras y confiables, capaces de transmitir grandes volúmenes de datos a alta velocidad y con bajas latencias. Aquí se encuentran dos principales tendencias el estándar 802.11p o WAVE (“Wireless Access in Vehicular Environments”) diseñado por la IEEE y 5G para redes móviles. [Martínez-Díaz y Soriguera 2018] [3]

Esto lleva finalmente a la etapa de toma de decisiones, que involucra la predicción, planeación de rutas y evasión de obstáculos, basado en las importantes tareas previas de percepción y cooperación con otros vehículos.

## Objetivos

El objetivo del trabajo es presentar la implementación sobre el modelo de carro autónomo AutoModelCar de un conjunto de algoritmos que le permitan la detección y rastreo de otros autos, así como simular una red vehicular vanet que les permitan en un primer nivel notificar la presencia de posibles obstáculos para poder reducir y prevenir accidentes de tránsito, por ejemplo, mediante la oportuna detección de otros vehículos, mismos que sean comunicados a otros autos mediante una red vehicular, pues dichos obstáculos pueden encontrarse en puntos ciegos para el otro vehículo. Además, el uso del rastreo nos puede ayudar a determinar si son o no potenciales riesgos según la trayectoria que estén siguiendo. En un segundo nivel, contar con una vanet, rastreo y seguimiento de otros vehículos pueden permitir llegar a la negociación entre vehículos para la planificación de rutas.

A continuación, se desarrollan los temas a tratar en este trabajo:

* En el capítulo 2 se presenta el estado del arte en que se encuentran los sistemas de detección, junto las investigaciones que se llevan a cabo para la creación de vanets, temas que sirven de introducción a las técnicas que se abordarán con mayor detalle en los siguientes capítulos. Si bien en el campo de la detección está dominado por las redes neuronales, hay otras alternativas bien exploradas con la característica de tener mayor velocidad a costa de sacrificar un poco de precisión, mismos que son parte fundamental y requisito del trabajo. Mientras que en las redes vanet, el uso de tecnologías celulares ha ganado un fuerte impulso gracias a 5G, sin embargo, el uso de DSRC con el estándar 802.11p aún presenta ventajas y sobre todo mayor tiempo de desarrollo. En el capítulo se presentan también el alcance del trabajo y las limitaciones dadas principalmente por la plataforma sobre la que se trabaja.
* En los capítulos 3 y 4 se desarrollan los temas de detección y rastreo respectivamente. Se entra en mayor detalle en qué consisten estos problemas, se describen distintas técnicas, junto a sus fundamentos matemáticos, tanto las utilizadas para este trabajo, como otras alternativas.
* En el capítulo 5 nos centramos en la plataforma sobre la que se desarrolla el trabajo, el AutoModelcar, para hablar un poco de ROS y OpenCV. Aquí se presenta un tutorial de ROS para la programación sobre el carrito, con el fin de que otros puedan retomar y avanzar sobre el proyecto.
* El capítulo 6 abarca la implementación de nuestros algoritmos para rastreo y detección, se describe con detalle desde el entrenamiento de los modelos, los pasos de preprocesamiento de imágenes, los algoritmos en sí, tanto para usarse fuera como dentro del carrito con ROS.
* A partir del capítulo 7 se desarrolla la segunda parte de este trabajo, las redes vanet. Aquí se plantean las problemáticas principales para el uso de redes vehiculares como la falta de infraestructura y la seguridad, se repasan los modelos principales, las tecnologías facilitadoras, junto a técnicas de enrutamiento.
* Para el capítulo 8, en base a la investigación se presenta la arquitectura de una red vanet basada en DSRC, junto con la estructura y formato de los mensajes que se transmitirían, así mismo se realiza la simulación de la red.
* En el capítulo 9 se repasan y comentan los resultados obtenidos, se presentan áreas de oportunidades para mejorar, y cómo se puede avanzar en la investigación, tanto del coche autónomo, y cómo se podría implementar físicamente en el AutoModelCar la red vanet.
* Por último, en el apéndice se agrega la documentación realizada del coche autónomo, junto con la configuración y cambios realizados para su funcionamiento.

## Metodología a usar

El trabajo cuenta con una parte inicial de investigación y se sustenta de forma teórica metodológicamente en información propuesta en *papers* como “Linear SVM classifier based HOG car detection” de [Bougharriou, et al, 2017] [4], ó “A Two-Stage Approach to People and Vehicle Detection With HOG-Based SVM” de [Feng Han et al. 2006][5], que proponen el uso de hog con svm pensado principalmente como un algoritmo ligero y rápido.

Posteriormente, se aplica de forma experimental sobre el simulador realizado por el profesor Marco Morales junto con el equipo de Eagle Knights del ITAM; y en el modelo a escala desarrollado por el profesor Raúl Rojas en Freie Universität Berlin, sobre este último se presenta la documentación y una guía para el uso del vehículo.

# Capítulo 2

## Estado del arte

En la investigación sobre coches autónomos, una de las áreas de gran relevancia y que ha tenido grandes avances en los últimos años es la visión por computadora. En este campo se tienen dos aproximaciones: enfoque modular y monolítico de extremo a extremo. El primero y más apoyado consiste en partir un modelo complejo de altas dimensiones a variables controlables de bajas dimensiones separables en módulos que pueden entrenarse y probarse de forma independiente, estos módulos normalmente se dividen en: percepción, análisis del entorno, planificación de trayectorias y control del vehículo. Típicamente con técnicas de aprendizaje de máquina como redes neuronales en los campos de percepción y análisis, mientras que para planificación y control se usan enfoques tradicionales como máquinas de estado, algoritmos de búsqueda y modelos de control. Por un lado, un enfoque modular permite un resultado más fácil e interpretable para las personas, sin embargo, no necesariamente más eficiente, o rápido que el enfoque monolítico, considerando que cada problema se ve de forma independiente.

Si bien el crecimiento en aplicaciones de aprendizaje de máquina ha sido grande, como se menciona anteriormente, no se sufre de problemas como asignación de pesos, sobreajuste, o en el caso de redes neuronales de una difícil interpretación. Acompañado de estas técnicas, los coches autónomos van acompañados de una gran variedad de sensores, como cámaras, odometría de las ruedas y de rango (LiDAR y sonar), de estos las cámaras se han beneficiado mucho gracias a ser baratas y fáciles de usar, [Janai et al. 2021][6] como el coche de Intel Mobileye demos equipado solo con ellas.

Cuando se utiliza el enfoque basado en aprendizaje de máquina se requieren de grandes *datasets* para el entrenamiento de los modelos, por lo que, para avanzar en las investigaciones, se han puesto de forma libre diversos *datasets* para la detección, rastreo, estimación de movimiento, como KITTI uno de los *benchmarks* más famosos, u otros proyectos públicos como ApolloScape que proporcionan imágenes y anotaciones.

Para la clasificación, hay enfoques tradicionales de aprendizaje de máquina, en los que se proponen el uso de un conjunto de clasificadores de cascada, o HOG+SVM, los cuales funcionan relativamente bien con pequeños *datasets*, son baratos computacionalmente y fáciles de interpretar. Aun así, el enfoque dominante son las redes neuronales convolucionales, usados para realizar tanto detección completa como por partes, gracias a que son más adaptables, transferibles, de mayor precisión y escalan con los datos, un enfoque que aún parece ser insuficiente. De modo que, la investigación alrededor de coches autónomos está lejos de acabar, pues estos modelos no son suficientes al actuar en entornos nuevos, como diferentes condiciones de clima.

Con el objetivo también de mejorar la seguridad, el campo de las comunicaciones entre vehículos también ha tenido gran avance, desde nuevos protocolos, tecnologías, y técnicas para tener mayores tasas de transmisión, a menores latencias y más confiables, gracias en parte a 5G, *fog computing*, y antes 802.11p, han permitido pensar en nuevas aplicaciones, como su uso para redes vanet, donde se tienen dos tendencias vehículo a infraestructura y vehículo a vehículo.

En vehículo a vehículo se ha explorado desde la tercera generación de telefonía celular con la 3GPP, como servicios que permiten a dispositivos comunicarse entre sí cuando están en proximidad [Tariq Islam y Cheolhyeon Kwon s. f., 2022] [7]. El debate aquí viene desde usar bandas licenciadas (asignar nuevos espacios o reusarlas), o no licenciadas, que técnicas de acceso y asignación de recursos usar, hay enfoques tradicionales como basadas en aprendizaje de máquina.

Entre los avances se ha logrado la estandarización de algunos protocolos como DSRC, en el que 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 MHz de los 75 en los 5.8 GHz para usuarios sin licencia.

Las investigaciones se centran desde el tipo de tecnología a usar como celulares con amplia cobertura y seguridad, pero de alto costo y latencia, contra el estándar IEEE 802.11p sobre la banda 5.8- 5.9 GHz, promovido por industria automotriz de Europa y USA. Técnicas de *broadcasting*, tanto de banda estrecha similar a radio FM, como banda ancha, junto a técnicas basadas en localización y *clustering* que se limiten a solo un sitio de interés. Protocolos de enrutamiento, con métodos como reenvió oportunista, basado en trayectoria o geográfico, en él se deben lidiar con problemas como el cambio en la topología y escalabilidad. La potencia de transmisión, ajuste de la potencia, problemas de interferencias. Y, por último, pero no menos importante, la seguridad y privacidad, pues pueden suponer grandes peligros la transmisión falsa de mensajes, debe tener confianza, resiliencia.

Se tiene múltiples programas orientados a la cooperación entre vehículos, como “Inter-Vehicle Hazard Warning” en Europa, conducción cooperativa como CarTALK 2000, PROMOTE-Chauffeur e INVENT VLA. Igualmente, consorcios como Car2Car también en Europa o “Vehicle Safety Communication” en USA [Jakubiak y Koucheryavy 2008] [8].

Igualmente, ya hay vehículos en el mercado con capacidades para comunicarse de la forma V2X (vehículos a todo), como General Motors que usa DSRC, Audi usa una combinación entre DSRC y Celular, Ford y BMW.

## Alcance

Para el trabajo se presenta la implementación en el modelo a escala del carro autónomo de los algoritmos para la detección y el rastreo de vehículos, junto a la simulación para una red vanet.

El primer paso fue aprender a usar la plataforma del carro autónomo para la cual se generó su documentación, junto con tutoriales, con el objetivo de que otros alumnos puedan retomar y avanzar en este proyecto, de forma más amigable. En la documentación se incluyen la configuración del carro, los nodos y tópicos principales, se comentan algunos cambios hechos, junto con una guía para crear nuevos proyectos, hacer uso de los sensores y transformaciones, basado en el libro “A gentle introduction to ROS” de Jason M. Kane, aplicado para el carro.

El siguiente paso es la programación de los nodos para detección y rastreo, los cuales se hicieron por aparte como un proyecto en visual studio y se validaron los resultados con un video, para posteriormente traducirlo como nodo de ROS y probar los resultados primero en el simulador y después en el modelo. Para la programación se usa Opencv con los cuales se realizó una fase para el entrenamiento de los modelos usados para cascada y hog+svm, principalmente para el caso del segundo involucra encontrar los parámetros adecuados que mantengan buena precisión junto con velocidad en la predicción.

Con el objetivo de que funcione en tiempo real, la detección con ventana deslizante usando un modelo entrenado con hog+svm resulta ser muy lento, por ello se agrega como paso anterior cascada que logra una detección multiescala rápida en la que se logren identificar todos los vehículos, aunque con muchos falsos positivos. Para el segundo paso de la detección, se agrega la máquina de soporte vectorial para la predicción, ahora visto como un problema de clasificación para filtrar los falsos positivos sobre las ventanas elegidas por cascada.

Para reducir el tamaño de la base de datos, el tiempo de entrenamiento, al igual que mejorar ligeramente la precisión del modelo, se separó el problema en dos máquinas de soporte vectorial asemejándolo a un problema de dos clases, si el coche está de frente o de lado, a costa de un incremento en el tiempo para la detección. Igualmente, sería útil probar a dividir los datos con más clases.

Como base de datos se usó “GTI vehicle image database” y “KITTI vision benchmark suite”. Con cascada se usaron un set de positivos y negativos del tamaño de 5000 imágenes, para svm se usaron dos *datasets* positvos de 1500 imágenes positivas con coches de frente o de lado, y uno negativo con 3000 imágenes, que corresponden a la carretera, árboles y señales. Es recomendable usar *datasets* más grandes.

El vehículo autónomo cuenta con sensores como un LIDAR y una cámara de profundidad, de los cuales para la primera parte del trabajo en detección y rastreo se usará la cámara únicamente como imagen en 2D, posteriormente puede integrarse el uso del LIDAR o la profundidad para estimar la distancia a los objetivos, tomando una ROI que corresponde a la ventana del objeto detectado.

Para la simulación de una red vanet, se realiza primero una comparación e investigación de los modelos en los que se centran las investigaciones actualmente y las tecnologías que lo permitirían, considerando que se trata de una red simple y de baja latencia para la comunicación ad-hoc entre vehículos cercanos, con usos como alertar de posibles obstáculos y prevenir accidentes, por lo que no se considera, por ejemplo, la necesidad de cómputo en la nube para la planeación de rutas.

## Requerimientos funcionales

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, sin embargo, de estos dos lenguajes de programación se eligió C++ ya que es mucho más rápido y no se requiere del intérprete de python.

En cuanto a los algoritmos de Kalman, svm y cascada, así como para el preprocesamiento de imágenes se usan las librerías de OpenCV 2.4, debido a que ya vienen incluidas en el carro. Además, es open source y cuenta con implementaciones rápidas y optimizadas para visión por computadora en tiempo real. También se debe tomar en cuenta que, para los métodos de aprendizaje de máquina usados, se requiere de entrenar los modelos desde cero.

# Capítulo 3: Detección de objetos

## Descripción

Como se ha mencionado anteriormente, en este capítulo nos centramos en el problema de la detección de objetos, una parte muy importante de la conducción autónoma, pues nuestro vehículo se encontrará en ambientes con otros participantes del tráfico, desde vehículos, personas, hasta animales. Para el objetivo del trabajo, nos centramos únicamente en vehículos, pero las mismas técnicas se pueden replicar a otras clases.

Dada la problemática y limitaciones del problema, desde que se requiere realizar la detección desde una cámara en movimiento de objetos móviles, con formas complejas, el enfoque dominante se centra en técnicas de aprendizaje de máquina, y más específicamente en aprendizaje profundo con redes neuronales. Aun así, se tienen otros enfoques como el presentado que combina máquinas de soporte vectorial y clasificadores de cascada, u otras ideas como el uso de modelos que compensen el movimiento de las cámaras para usar técnicas tradicionales como optical flow. 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, si bien ambos tipos de errores son importantes, nuestra prioridad sería reducir los falsos positivos.

A diferencia de los humanos que somos capaces de reconocer objetos parcialmente visibles, gracias a que nuestra mente compensa las partes faltantes e invisibles, esto se convierten en problemas complejos para la computadora que no solo debe lidiar con aprender a identificar una clase, si no que se presentan problemas como la iluminación, esto se le conoce como oclusión y ocurre cuando el objeto está oculto ya sea por otro objeto del mismo tipo “intra-class occlusion” o, por otro objeto o elemento “inter-class occlusion”.

Con la aparición de oclusión, el rendimiento de clasificadores empeora drásticamente, una solución parecería ser crear diferentes clasificadores que lidien con la oclusión, pero aparecen nuevos problemas como la falta de datasets, para poder identificar la presencia de oclusión [Saleh, Szénási, y Vámossy, s. f.]. Por lo que, para este trabajo nos centramos en entrenar clasificadores para la detección de vehículos de día, que traten de ser lo más generales para lidiar con problemas de oclusión como sombras, más no se separa el problema, además, en casos como en la noche se pueden mejorar los resultados con el uso de otros tipos de cámaras.

El camino clásico para la detección de objetos consiste primero en una etapa de preprocesamiento de la imagen, la extracción de una región de interés, la clasificación del objeto y una etapa de verificación o refinamiento.

Para el preprocesamiento, 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, adicionalmente, según el problema se puede agregar el uso de filtros de bordes o técnicas de compresión de imagen.

Para nuestro caso, la imagen primero se reescala para usarla en un clasificar, y se pasa a escala de grises, esto ayuda a simplificar los algoritmos y reducir la complejidad, ya que no se requiere que el modelo aprenda ciertos colores, esto en cierta forma comprime la imagen al reducir los 3 canales a 1.

Además, se incluye el uso de filtros, los cuales ayudan a introducir imperfecciones a los datasets, lo que permite que los modelos sean más robustos al mundo real, el uso de filtros de difuminado ayuda a esto, degradando la calidad de imagen y reduce también la necesidad de contar con datasets más grandes.

El difuminado en imágenes consiste en tomar píxeles vecinos y sacar un promedio de ellos, lo que reduce el detalle. [Nelson J. 2020] En opencv se presentan distintas técnicas para aplicar blur, algunas mejores que otras según sus aplicaciones.

El suavizado de imagen resulta de convolucionar la imagen con un filtro pasa bajas, esto elimina las altas frecuencias que consisten en ruido y bordes de la imagen, lo que produce el suavizado, el filtro usado en el trabajo es:

* Blur gaussiano

En lugar de usar una matriz, se usa un kernel gaussiano en el que se especifican el ancho y altura del kernel, junto a la desviación estándar. A diferencia del otro que da un promedio a todo, el pixel original recibe el mayor peso, mientras los pixeles reciben pesos más pequeños al alejarse del centro.

Alternativamente se puede implementar aplicando dos veces el filtro gaussiano de una dimensión. [«OpenCV: Smoothing Images» s. f.]

Después del preprocesamiento se debe 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, lo que puede resultar en un proceso muy caro según qué tan exhaustiva sea la búsqueda. Por lo que, igual que en este trabajo, se deben asumir algunas cosas, como el tamaño de la ventana, la posición de los candidatos, la proporción con la cual se incrementa el tamaño de la ventana, lo que reduce el número de comparaciones.

Otras alternativas son búsquedas selectivas, en lugar de recorrer toda la imagen, se seleccionan solo algunas localizaciones aproximadas [Janai et al. 2021], esto se comenta en el trabajo, pues permitiría dividir la imagen en regiones y según cada una aplicar un clasificador u otro.

## Técnicas de detección

Algunos métodos populares experimentados desde el punto de vista clásico son los usados en el trabajo: clasificadores de cascada y máquinas de soporte vectorial con histograma orientado por gradiente. En ellas, se ha observado que, por un lado, SVM con HOG trabaja mejor a más altas resoluciones, mientras presenta un mayor tiempo de procesamiento. Por otro lado, con cascada funciona mejor a bajas resoluciones, alcanzando resultado prácticamente en tiempo real. Otros métodos son “Scale-Invariant Feature Transform” y “Speeded Up Robust Feature”, todos ellos dependen de extraer manualmente la información de las imágenes, difíciles de diseñar y limitados en sus capacidades de representación. [Janai et al. 2021] Por otro lado, está el uso de redes neuronales convolucionales como YOLO, MobileNet, Fast R-CNN.

### Clasificador de cascada

Se trata de un método propuesto por Paul Viola y Michael Jones en "Rapid Object Detection using a Boosted Cascade of Simple Features" del 2001. Se trata de un enfoque donde una función en cascada, en este caso un grupo de funciones que en su conjunto componen el clasificador las cuales se revisan de forma secuencial que cumplan las condiciones, en este caso evalúan algunas características que determinan si puede o no pertenecer a una clase el objeto, estas son entrenadas por grandes sets de imágenes positivas y negativas.

Primero se requiere preparar de sets de imágenes positivas (ejemplos representativos de la clase u objeto que se quiere detectar) e imágenes negativas. El siguiente paso es extraer las características de la imagen, aquí se pueden tener diferentes métodos como haar o lbp para extraer la información de la imagen, de forma que en lugar de trabajar con la imagen completa, se reduzca la complejidad del problema obteniendo solo información relevante como de los bordes, por ejemplo, en haar las características se extraen substrayendo a la suma de pixeles del rectángulo blanco, los pixeles en el rectángulo negro [Shetty et al. 2021] [«OpenCV: Cascade Classifier» s. f.], como se observa en la imagen 3-1, se tratan de kernels que se aplican sobre la imagen para extraer los bordes o líneas, según el tipo.

Imagen que contiene Diagrama

Descripción generada automáticamente

Figura ‑ Diferentes kernel aplicables para la extracción de características haar [shetty et al. 2021]

A continuación, se describe de forma más exhausta como obtener las características haar, el primer paso es formar los kernel mostrados en la imagen según las características que se quieren extraer, estos se deben aplicar sobre la imagen variando el tamaño y ubicación sobre la imagen para calcular la mayor cantidad de características posibles. De nuevo, la operación consiste en sumar los pixeles que caen en los rectángulos blancos y restarles los negros, para simplificar el problema se usa la integral de una imagen, esto es, cada píxel es la suma de los píxeles arriba y a la izquierda. [Shetty et al. 2021]

Otro método usado para extraer las características es LBP o Local Binary Patterns, mismo que se usa en el proyecto, con la ventaja de que su entrenamiento es mucho más rápido que haar, pero menos preciso. Presentado por Ojala et al. en 1996, en LBP, para cada píxel, se toma una celda alrededor de él de 9 píxeles (3x3), estos píxeles se comparan con el centro, y a los vecinos se les asigna un valor de 1 si es mayor o igual al centro, esto nos da 8 píxeles alrededor con 256 combinaciones posibles, lo que nos permite promediar los alrededores asignando un valor entre 0-255.

Xc, yc es el píxel del centro, ic es el brillo del centro, ip el brillo de un píxel vecino, se asigna 1 si es mayor o igual, de lo contrario es 0, para calcular el valor del centro, 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]

Calculadas las características, la mayoría resultan ser irrelevantes, para elegir las mejores características se usa Adaboost, mismo que le da el nombre de cascada, son un grupo de predictores sencillos que se colocan en secuencia. 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 alcanzar cierto número de características.

Finalmente, cada característica evaluada corresponde a un clasificador débil, que por sí solos no pueden decidir sobre la imagen completa, si no que se realiza la suma ponderada de los clasificadores, a diferencia de SVM que es un clasificador fuerte. Cuenta con 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.]

### Hog+svm

Se trata de la combinación de dos técnicas, máquinas de soporte vectorial o svm que es un algoritmo de aprendizaje supervisado, y de histograma orientado por gradiente que permite extraer las características de la imagen.

Para empezar, en histograma orientado por gradiente, 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 Gx y Gy para cada píxel, con la fórmula:

Lo que hace es tomar las intensidades de los píxeles anteriores y posteriores. Estos gradientes horizontales y vertical se calculan multiplicando por el kernel:

Un conjunto de letras negras en un fondo blanco

Descripción generada automáticamente con confianza media

El siguiente paso es formar los histogramas, para eso se calcula la magnitud y ángulo de cada píxel:

La imagen que fue dividida en celdas, son agrupadas para formar un bloque al que se le asocia un histograma con n valores diferentes llamados bins. Cada punto del histograma toma valores entre 0 a 360 grados, con 9 bins se tiene un salto de 20 grados (toma valores de 0, 20, 40 … 360), esta aproximación en celdas y bins 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 hyperplano óptimo (figura 3-4) que separe los datos en dos clases representadas como -1 y 1. Este método es fácil de entrenar y permite una clasificación relativamente rápida.

Gráfico

Descripción generada automáticamente

Figura 3‑4 Ejemplo de la separación de dos varíables por una frontera, se muestra la separación óptima que logra el mayor márgen entre las dos variables [«opencv: introduction to support vector machines» s. f.]

Donde a son los pesos del vector, b el sesgo, y x los datos. Además por convención el hiperplano se escala de forma:

Los ejemplos de entrenamiento más cercanos al hiperplano son los que definen el margen de separación con la frontera y son llamados vectores de soporte.

En el entrenamiento se usa una función de pérdida o hinge loss, si la predicción es incorrecta la penalización es grande cuanto más lejos de la frontera está. Si la predicción es correcta y está cerca de la frontera, recibe una pequeña penalización, si es lejos el hinge es de 0

Para clasificar se aplica la función signo:

La idea es tener el mayor margen posible a la frontera, de tal manera que se reduzca la probabilidad de error.

Para hallar a y b se tienen dos funciones una de error de entrenamiento y un término de regularización para evitar grandes escalas, función de costo:

EL primer término es la distancia de los datos a la frontera, el segundo la regularización. Función que se minimiza por el método de descenso por gradiente para hallar los valores de a y b, en este método se define aleatoriamente un tamaño de paso y una dirección, tal que minimice la función de costo:

Las máquinas de soporte vectorial usan un hiperplano lineal para separar las clases, para casos donde los objetos no se pueden separar de forma lineal, primero se realiza un paso para poner las características en un espacio de mayor dimensión, la transformación se puede realizar por diferentes mapeos no lineales (RBF es el usado en este trabajo el cual mapea la función radialmente) [«OpenCV: Introduction to Support Vector Machines» s. f.] [Kecman 2005]

Texto

Descripción generada automáticamente

Figura ‑ tabla de diferentes kernels para linealizar un problema [kecman 2005]

### Comparación con redes neuronales convolucionales

Las redes neuronales convolucionales se especializan en el tratamiento de imágenes y audio, compuestas de 3 capas principales: capa convolucional que analiza las imágenes y detecta la presencia de características, capa de pool que reduce el tamaño y preserva solo las características importantes, alguna capa de activación con ReLU para devolver valores como 0 o 1, y finalmente la capa “Fully Connected” que conecta con otras neuronas. Con cada capa apilada permite identificar características más complejas, hasta llegar al objeto. [«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 región proposal network, que permite clasificar regiones y aplicarla dentro de alguna categoría. Region proposal toma una imagen y como salida devuelve sets de propuestas de objetos en 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.

Suposiciones que pueden ser inapropiadas por ejemplo en problemas de altas dimensiones donde el mapeo a una forma lineal no es adecuado (maldición de la dimensionalidad), o no sigue una distribución normal por lo que se requiere construir de otro algoritmo más efectivo. [Kecman 2005]

Al final el enfoque de redes neuronales convolucionales recientemente ha alcanzado una eficiencia casi en tiempo real, en parte gracias a la paralelización en la GPU. Algunas muestras, por ejemplo, en Fast R-CNN que ha alcanzado una frecuencia de 0.5 Hz, Faster RCNN con región proposal alcanza 17 Hz, en cuanto a YOLO9000 alcanza 90 Hz en imágenes de 288 x 288 píxeles, y 40 Hz en imágenes de 544 x 544. [Janai et al. 2021]

Por lo que, los métodos tradicionales, aunque puedan alcanzar niveles de precisión similares como SVM, su uso va a ser limitado únicamente en casos con poco poder de cómputo, en aplicaciones cercanas a tiempo real.

# Capítulo 4: Rastreo de objetos

## Descripción

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 frame, esto es asociar dos imágenes en el tiempo, como dice en el libro Joel Janai, se diferencia de la detección de objetos, donde los frames se procesan de forma independiente. Esto permite, por ejemplo, detectar 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 frames cuando se trata de la misma clase. [Janai et al. 2021]

## Técnicas

Entre las técnicas disponibles, el enfoque principal ha sido tomarlo como problemas de inferencia bayesiana, donde 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.

Algunos de estos métodos son el filtro de Kalman y filtro de partículas, de los cuales ambos 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 cuantas más muestras se tengan, 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:

Donde w y v son ruido.

O para el caso discreto:

La formulación del problema es similar al observador de Luenberger, 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 coches sobre una imagen 2D, considerando la posición xy y las velocidades por lo que el filtro de Kalman es 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.

El método alternativo es el filtro de partículas, donde el rastreo se hace mediante un método secuencial de Monte Carlo. La diferencia con el filtro de Kalman, es que mientras el filtro de Kalman parte de asumir que un problema es lineal y normal (gaussiano) y el filtro extendido sirve para problemas no lineales, el filtro de partículas se aplica en problemas no lineales y no gaussianos, donde 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, y 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. Como se implementa en el proyecto, también mediante la predicción se facilita la detección de objetos, tomando regiones de interés según la posición del objeto estimado, aun así, cada x tiempo se debe realizar la detección de un frame completo, no solo para detectar nuevos objetos, también porque se van acumulando errores.

## Filtro de Kalman

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

Donde x es la posición del objeto, x0 la posición inicial, v0 la velocidad inicial, a la aceleración y 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.

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:

Caso con velocidad constante

El filtro de Kalman parte de 5 ecuaciones:

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

Donde Kn es la ganancia de Kalman.

1. Ecuación de ganancia del filtro
2. La actualización de incertidumbre estimada
3. Extrapolación de la incertidumbre estimada para sistemas estáticos

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

### 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:

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

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

Se traducen en la matriz de transición F como:

Si se incluye la aceleración queda como:

[Becker s. f.]

### Forma general

Para el caso de movimiento tenemos de forma directa la matriz de transición F, en otros casos cuando no tenemos las matrices de transición F y de control G, se debe resolver el sistema dinámico, lo que involucra primero transformar la ecuación a su forma canónica de controlabilidad:

Lo que nos da las matrices A, B, C, D:

Entonces para obtener la respuesta en tiempo, se resuelve la ecuación:

Entonces nuestras ecuaciones principales quedan como:

1. Ecuación de predicción
2. Ecuación de extrapolación de la covarianza

En donde P se refiere a la incertidumbre de la estimación, F es nuestra matriz de transición ya calculada, y Q la matriz de ruido.

Ambas corresponden a la predicción, mientras que para los pasos de corrección:

1. Ecuación de actualización de estado

La ecuación actualiza el estado actual usando la predicción anterior junto al medición obtenida por el observador. En lugar de simplemente reemplazar con el valor medido, se usa una ganancia de Kalman que se actualiza con cada paso (Kn), de forma que agrega solo una parte del error entre la medida y el valor predicho.

Zn es una medición del valor obtenida con el observador mediante la ecuación, que considera la posibilidad de tener ruido aleatorio:

1. Ecuación para actualizar la covarianza de forma simplificada
2. Ganancia de Kalman

[Becker s. f.]

# Capítulo 5

## Implementación

### 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] [12] y GTI [Arróspide et al. 2012] [13], se partieron en dos sets de aproximadamente 7000 vehículos, y 8000 de no vehículos, que correspondían a imágenes como pedazos de la carretera, sombras, árboles y señalizaciones como se ve en la figura 5-2.

Imagen de la pantalla de un celular con texto e imágenes

Descripción generada automáticamente con confianza media

Figura 5‑1 Ejemplos de vehículos

Captura de pantalla de un celular con texto e imágenes

Descripción generada automáticamente

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.) [10] se debe crear un archivo .dat que sirve como descripción de las imágenes positivas, en él se incluyen el nombre de la imagen, número de objetos presentes, coordenadas, y dimensiones del objeto.

1070.png 1 0 0 64 64

Mientras que para las imágenes negativas se crea a un archivo .txt que contiene únicamente su nombre.

El módulo traincascade se trata de un ejecutable incluido solo hasta la versión de Opencv3.4, 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

- 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.

### 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, 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.

Una captura de pantalla de un celular con texto e imágenes

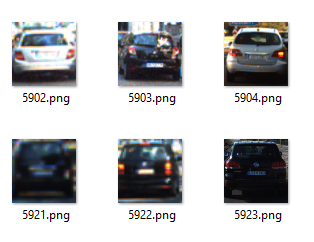
Descripción generada automáticamente con confianza bajaPara descartar los falsos positivos, se usan máquinas de soporte vectorial, aquí con el fin de lograr una mejor precisión, reducir el tiempo de entrenamiento y el número de muestras requeridas, se partió el problema en dos máquinas, una para identificar coches 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 vehículos de lado

Figura 5‑4 imágenes de la parte trasera de vehículos

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. Es importante mencionar que entre la versión de OpenCV 2.4 (la del carrito) y la más nueva OpenCV 4 hay bastantes diferencias, y no es posible usar el mismo modelo.

Primero se usa un método *load\_images*, en el que se realiza un pequeño preprocesamiento para las imágenes, se pasan a blanco y negro, se les aplica un filtro gaussiano, con el fin de remover el ruido y detalles insignificantes, lo que ayuda a mejorar la precisión. Otros tipos de filtros que se pueden usar son por ejemplo el *motion* *blur*, para un efecto de velocidad, o un filtro de *sobel* para resaltar solo los bordes.

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

Las características que se definen son:

* winSize
* blockSize
* cellSize
* blockStride
* nbins
* derivAperture
* winSigma

De ellas, las más importantes son:

* BlockSize se 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 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] [11] nos permite ver las características extraídas por HOG. Varios ejemplos se observan en las figuras de la 5-5 a 5-11, variando parámetros de HOG, como incluyendo filtros.

Diagrama

Descripción generada automáticamente

Figura 5‑5 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.

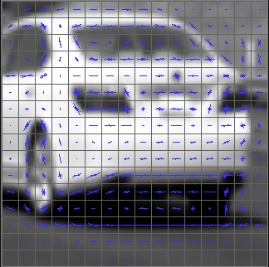
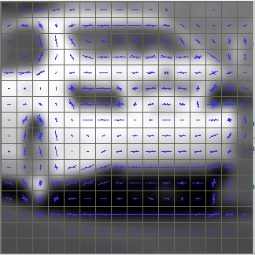


Figura ‑ Con blocksize (64,64), cellsize y blockstride de (4,4).

Figura 5‑7 Con filtro gaussiano (5,5)

Imagen que contiene interior, con baldosas, pequeño, tabla

Descripción generada automáticamente

Figura 5‑9 Imagen del vehículo del simulador.

Imagen que contiene Patrón de fondo

Descripción generada automáticamenteImagen que contiene interior, ventana, cama, grupo

Descripción generada automáticamenteEjemplo de imágenes negativas

Figura 5‑10 hog features de la carretera

Figura 5‑11 HOG features de un árbol

Aumentar el número de características tiene un impacto muy alto en el tamaño de la máquina de soporte vectorial, junto al tiempo de procesamiento, se eligió, 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.

Para extraer HOG se usa la función computeHOGs, 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:

convert\_to\_ml(gradient\_lstC, train\_dataC);

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. 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 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(TermCriteria::MAX\_ITER,

(int)1e8, 1e-6));

Opcionalmente se pueden poner class weights, los cuales son auxiliares a C para penalizar el error.

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, fue posible reutilizarlos cuando se incrementó el tamaño de los datasets y para el entrenamiento en OpenCV 2.4, sin embargo, puede que haya otros óptimos.

svmL->train(ml::TrainData::create(train\_dataL,

ml::ROW\_SAMPLE, labelsL));

Para los coches de lado, los valores obtenidos fueron de 2.25x10^-3 para gamma y 12.5 para C. Para los coches 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 carrito.

### Detección y nodo para el carrito

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 frames sin aparecer, en el caso del carrito se usaron 15 frames, que, usando una frecuencia de 5 Hz, corresponde a 3 segundos. Se trata de un vector de rectángulos que guardan la posición, el último frame en que apareció y de filtros de Kalman. Con ello nos permite seguir múltiples objetos sin que tengan que aparecer en el frame actual.

Una vez todo inicializado entonces dentro de un ciclo se recibe la imagen, para las pruebas se leen imágenes extraídas de un video. Para el carrito es necesario un paso extra para extraer la imagen de la cámara:

Al inicio del main se define:

ros::NodeHandle nh("~");

ros::Subscriber camara\_sub =

nh.subscribe("/app/camera/rgb/image\_raw", 10,

camaraRGBCallback);

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

Esta función de callback se entra siempre que se reciba una nueva imagen y la 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);

Figura 5‑13 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 frame 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 frames realiza la detección completa del frame. Dentro, como no se usa la detección multiescala de HOG, si no que solo la 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 = 2.0000000000000001e-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), se aplica un filtro gaussiano y se extraen sus características:

resize(imgEsquina, imgSVM, Size(64, 64), 0, 0, cv::INTER\_AREA);

vector< float > descriptorsSVM;

GaussianBlur(imgSVM, imgSVMBLUR, Size(5, 5), 0);

imgSVMBLUR.convertTo(imgSVMF, CV\_8UC3);

hog.compute(imgSVMF, descriptorsSVM, Size(0, 0), Size(0, 0));

En compute el primer size(0,0) significan que se toman 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();

Finalmente, para hacer la predicción:

float result1 = svm->predict(descriptorsSVM);

En Opencv2.4:

float result1 = svm->CvSVM::predict(fm);

El problema se puede visualizar en la figura 5-11, donde a la hora de detectar autos que van apareciendo en las esquinas, pues ninguno de los dos detectores lo ven hasta que el auto no esté completo.

Imagen en blanco y negro de un carro

Descripción generada automáticamente Imagen en blanco y negro de un carro

Descripción generada automáticamente Imagen en blanco y negro de un carro

Descripción generada automáticamente

Figura 5‑11 Ejemplos de Falsos Negativos

Hecha la predicción, si el resultado es positivo (1), entonces se guardan dos veces para poder usar groupRectangles, que toma el promedio de los rectángulos sobrepuestos, 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 agregando al inicio:

ros::NodeHandle nh("~");

detec\_publisher =

nh.advertise<sensor\_msgs::Image>("/detec",1);

El mensaje es bajo el tópico /detec.

Al final de la detección:

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, y se publica, se puede visualizar las detecciones en RVIZ.

* Detección sobre solo una región:

En este caso se llama a la función:

detecRegion(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 lo 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));

### Rastreo

El siguiente paso es el rastreo, para cada detección se revisa si es el mismo vehículo entre dos frames, 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.

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:

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

Las ecuaciones son las siguientes:

x = x0 + vx0\*Δt

y = y0 + vy0\*Δt

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 e 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, 0, 1);

La matriz de medición queda como:

1 0 0 0 0 0

0 1 0 0 0 0

0 0 0 0 1 0

0 0 0 0 0 1

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));

Imagen en blanco y negro

Descripción generada automáticamente con confianza bajaImagen en blanco y negro

Descripción generada automáticamente con confianza bajaPodemos 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 5-12 y 5-13, 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, como se ve con mayor ruido “measurementNoiseCov” sigue mucho peor la señal cuando deja de tener nuevas mediciones, además de requerir de más para un mejor seguimiento, en el ejemplo deja moverse en el eje x, pero con ruido alto la predicción preserva todavía movimiento en ese eje.

Figura 5‑13 measurementnoisecov de 0.01

Figura 5‑12 mesasurementnoisecov de 0.1

Imagen en blanco y negro

Descripción generada automáticamente con confianza baja

Figura ‑14 measurementnoisecov de 0.001

Imagen en blanco y negro

Descripción generada automáticamente con confianza bajaImagen en blanco y negro

Descripción generada automáticamente con confianza bajaEn cambio, como se ve en las figuras 5-15 y 5-16, 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 0.5, los puntos azules reaccionan más rápido y como consecuencia se disparan

Figura 5‑15 Con ProcessNoiseCov de 0.00005, Los puntos azules reaccionan más lento a los cambios

**Actualización del filtro**

Para la actualización, se tiene la siguiente función, que recibe el filtro y lo devuelve actualizado, recibe la imagen, si hubo una detección en el frame 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:

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, primero porque se necesita para actualizar al siguiente estado, segundo para 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 frames.

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

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 ve la operación que realiza es: statePre = TransitionMatrix \* statePost

## Resultados

Tras el entrenamiento del filtro de cascada y las máquinas de soporte vectorial se probaron los detectores con un set de validación de unas 16000 muestras aproximadamente. Inicialmente con un set de entrenamiento de cerca de 400 muestras, y al ir variando los parámetros de HOG, se obtuvieron resultados lo suficientemente buenos. Es importante mencionar antes, que para el set de pruebas se combinaban ambos tipos de imágenes tanto coches de lado como de la parte trasera, y estaba algo desbalanceado, pues se tenían más coches vistos desde atrás.

Usando únicamente la máquina entrenada con la parte trasera de coches se consiguieron detectar correctamente el 77.8% de los coches, con un 0.1% de falsos positivos:

Tabla ‑ Matriz de confusión para svm con imágenes de la parte trasera de coches

|  |  |  |
| --- | --- | --- |
| Realidad/predicción | 0 | 1 |
| 0 | 0.99855 (TN) | 0.00144976 (FP) |
| 1 | 0.241294 (FN) | 0.758706 (TP) |

Precisión: TP/(TP+FP) = 99.81%

Exactitud: (TP+TN)/(TP+FN+TN+FP) = 87.86%

Por otro lado, la máquina entrenada con coches de lado alcanzó a detectar correctamente solo el 45.1% de coches correctamente, con un 0.8% de falsos positivos:

Tabla ‑ Matriz de confusión de svm con coches de lado

|  |  |  |
| --- | --- | --- |
| Realidad/predicción | 0 | 1 |
| 0 | 0.991859 | 0.00814096 |
| 1 | 0.548336 | 0.451664 |

Precisión: 98.2%

Exactitud: 72.17%

Por ultimo, con la idea original de juntar ambas máquinas, se alcanzó a detectar correctamente el 87.8% de los coches, con un 0.9% de falsos negativos.

Tabla ‑ Matriz de confusión resultante de juntar ambas svm

|  |  |  |
| --- | --- | --- |
| Realidad/predicción | 0 | 1 |
| 0 | 0.985632 | 0.01436768 |
| 1 | 0.14102 | 0.85898 |

Precisión: 98.92%

Exactitud: 92.23%

Posteriormente se aumentó el tamaño del set de entrenamiento a 1500 imágenes positivas, lo que mejoró los resultados, claro que a costa de aumentar considerablemente el tiempo de entrenamiento y para cargar los detectores, el tiempo para la detección también aumento, pero en menor medida.

Máquina 1 (vista de la parte trasera de los coches):

TN: 0.999888 FP: 0.00011152 FN: 0.130529 TP: 0.869471

Tabla ‑ Matriz de confusión de svm completa con la parte trasera de coches

|  |  |  |
| --- | --- | --- |
| Realidad/predicción | 0 | 1 |
| 0 | 0.994888 | 0.00511152 |
| 1 | 0.160529 | 0.839471 |

Precisión: 99.39%

Exactitud: 91.71%

Máquina 2 (vista de lado de los coches):

Tabla ‑ Matriz de confusión de svm completa con coches de lado

|  |  |  |
| --- | --- | --- |
| Realidad/predicción | 0 | 1 |
| 0 | 0.993773 | 0.00622672 |
| 1 | 0.586227 | 0.413773 |

Precisión: 98.51%

Exactitud: 70.37%

Ambas máquinas:

Tabla ‑ Matriz de confusión de ambas svm completas

|  |  |  |
| --- | --- | --- |
| Realidad/predicción | 0 | 1 |
| 0 | 0.997888 | 0.00211152 |
| 1 | 0.140529 | 0.859471 |

Precisión: 99.75%

Exactitud: 92.86%

Como se observa, las mejoras entre usar solo la máquina 1 y ambas son ligeras, aún así, en las pruebas en un escenario real los resultados empeoran, siendo el mayor problema la oclusión ambiental, con la cual, con pequeños cambios en la iluminación o sombras, una detección puede no aparecer en el siguiente frame. 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 frames, lo que entorpece principalmente al filtro de kalman.

Para empezar, en la prueba de un caso real con un video, el filtro de cascada arroja por frame entre 7 y 11 detecciones o candidatos para SVM, como se ven en la figura 6-27, pese a tener una tasa muy alta de falsos positivos, la idea es que encuentre la mayoría de los vehículos. En la figura 6-28 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 frames 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 6-29 se observa la predicción kalman.



Figura ‑ Ejemplos de detecciones sobre frames, en la primer imagen se indentifican correctamente los dos vehículos, en el segundo por el cambio de iluminación no detecta uno de los vehículos, en la tercer imagen se muestra un falso positivo



Figura ‑ 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

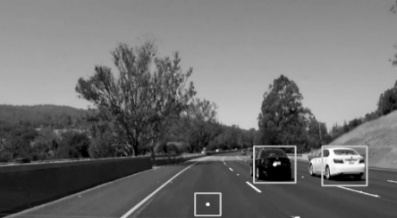


Figura ‑ Candidatos arrojados por el filtro de cascada

La solución presentada no es óptima, ni se puede trasladar a cualquier 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 carrito es de 640x480 pixeles, y parte del área de la imagen es ocupada por la parte frontal del vehículo.
* El número máximo de frames 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 taza u otra para hacer el procesamiento de las imágenes, por lo que entre frames se pueden esperar mayores o menores cambios según la frecuencia.
* Mismo razonamiento, para el máximo movimiento esperado de un vehículo entre dos frames, para poder asociar una detección como correspondiente al filtro de kalman
* Según la taza, e igualmente se puede limitar el problema si se considera el entorno sobre el cual el vehículo se mueve, junto a u 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 6-30), el cual se ha modificado para que aparezcan dos carritos, 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 coches, con los cuales se ajustaron los valores de la covarianza para el filtro de kalman, algunos resultados se observan en la figura 6-31.

Patrón de fondo

Descripción generada automáticamente

Figura ‑ Imagen del simulador desde gazebo, mapa con dos coches

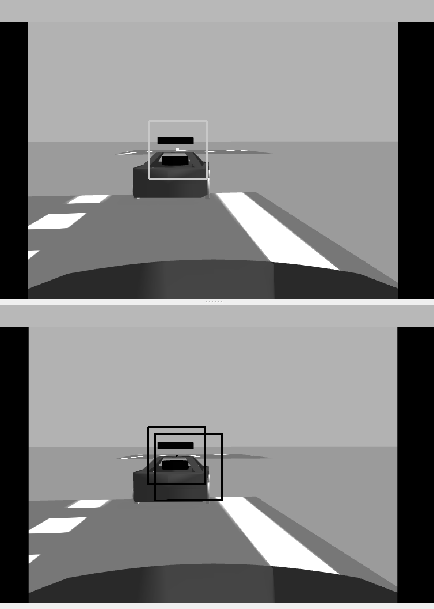
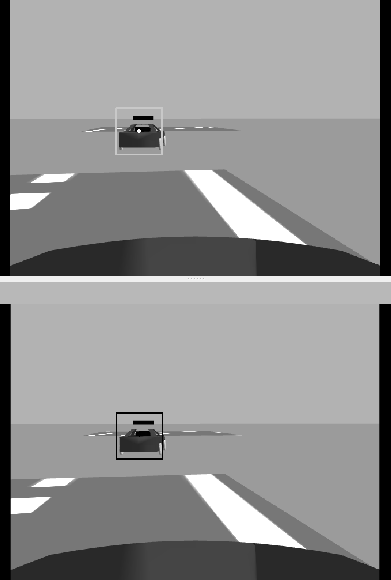


Figura ‑ Ejemplo de detecciones en el simulador, arriba se muestr la detección, abajo la predicción

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

* roslaunch manual\_control manual\_odroid.launch
* rosrun vision\_camara detecRastreoC

En el caso de los tiempos:

Tabla ‑ Tiempos medidos de los detectores en los tres ambientes de pruebas

|  |  |  |  |
| --- | --- | --- | --- |
|  | En Visual Studio | En el simulador | En el carrito |
| Tiempo para cargar las dos máquinas de soporte vectorial | 967.38 segundos | 10.156 segundos |  |
| Tiempo para la detección de un frame completo | 1700-2200 ms | 100-110 ms |  |
| Tiempo para la detección sobre una región | 60-90 ms | 50-70 ms |  |
| Tiempo para la predicción | 30-60 ms | 40-60 ms (predicción y corrección) |  |
| Tiempo total en detección de un frame completo | 1800-2300 ms | 140-180 ms |  |
| Tiempo total en detección solo de regiones en un frame | 150-200 ms | 110-140 ms |  |

Es importante mencionar que si bien el procedimiento en visual studio es lento, en especial a la hora de cargar las máquinas de soporte vectorial, donde demora hasta 17 minutos, se debe a que únicamente aprovecha el 13% del CPU, que en nuestro se corrió en un procesador intel i3 6100u. Por otro lado, tanto el simulador como el carrito 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 frame, en el simulador se pudo haber tomado una taza ligeramente más alta que 5 Hz.

En el caso para cargar el filtro de cascada, al ser muy ligero, prácticamente no demora tiempo, apenas un segundo para el simulador y 103 ms en visual studio. 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.

# Capítulo 6 Redes vanet

## Descripción

Dentro de las redes móviles ad hoc llamadas Manet, un tipo específico son las redes vehiculares ad hoc donde los nodos son vehículos con módulos de comunicación inalámbrica. Dentro se ha realizado u na amplia investigación y experimentación, sin embargo, no solo no hay un modelo definido o preferido, si no que dependiendo de los requisitos en la comunicación se requerirían de distintas tecnologías. En la tabla 6-1 se inicia mostrando una comparativa contra redes móviles y para vehículos aéreos no tripulados (fanet), que nos sirve para presentar la problemática inicial en una red vanet.

Tabla 6‑ 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 | No necesita | Consumo eficiente para mini UAV’s |
| Poder computacional | Limitado | Alto | Alto |
| Localización | GPS | GPS, A-GPS, DGPS | GPS, A-GPS, DGPS |

En el caso de redes vanet se debe lidiar con una cantidad variable de nodos, según el tráfico, que se pueden mover a gran velocidad, y que en muchos casos no hay línea de visión, aunque como ventaja su movimiento es conocido y está limitado a los caminos existentes, lo que facilitaría el uso de modelos que usan infraestructura para su conexión.

Entre las tareas de seguridad que se plantean, se incluye el avisar si tanto el vehículo usuario como otro dentro de su rango de visión se detiene, da la vuelta, cambia de carril, si desacelera, o realiza alguna acción anormal, con el fin de evitar colisiones.

Para estas tareas se requiere tanto de modelos que sean seguros, confiables y de baja latencia. Para nuestras tareas además se requieren de aplicaciones de localización con buena precisión, los ideales para esta tarea son los sistemas de navegación satelital, aquí se tiene GPS y GPS asistido (A-GPS), los cuales tienen una precisión de 10-20 metros, siendo insuficientes más que para planificación de rutas. Mientras que GPS diferencial (DGPS) usa estaciones en la Tierra para calcular y realizar correcciones (son estaciones fijas con posición conocida, que reciben su ubicación del satélite, ven el error y lo comunican a usuarios) con precisión de cerca de 1 metro. Otras ideas para mejorar la precisión incluyen el uso de unidades de medición inercial (IMU) para mantener registro de la posición entre dos mediciones con el GPS. Se pone énfasis en la localización, por que, para nuestro trabajo corresponde al principal dato que se quiere comunicar, pero en el caso del modelo de coche autónomo no se cuenta con un módulo incluido. [Hartenstein y Laberteaux 2010]

## 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 aquí, los nodos pueden reenviar la información (multihop), con el fin de abarcar una mayor área, aquí se puede apoyar de infraestructura en la carretera para darle mayor robustez. Dentro del broadcasting se puede hacer mediante distintos métodos:

* Inundación: Todo nodo que recibe la información lo reenvía, para evitar duplicación se puede implementar contadores (Time to Live ttl) de cuántas veces se debe reenviarse.
* Beaconing: Se considera como el envío periódico de información, 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 métodos de control para evitar problemas de congestión. Otras alternativas para controlar 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; o según que tanto se aleje del nodo original, los nodos retransmitan con mayor probabilidad. [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 o clustering, aquí hay nodos especiales que se dedican a retransmitir la información dentro de su grupo, y nodos repetidores (relay) que pertenecen a varios clusters, en general estos clusters se hacen de forma geográfica, y los repetidores son solo los nodos donde hay overlapping entre clusters.

Por último, hay propuestas basadas en unicast, que pueden funcionar según algunas aplicaciones, que requieran de que dos nodos se conecten, sin embargo, para aplicaciones de seguridad, no es ideal ya que es muy lento, dificultades escalabilidad, no útil para nuestro objetivo.

Uno de los protocolos de enrutamiento preferidos o más explorados es “Ad-hoc on demand distance vector routing” o AODV, gracias a que se trata de un protocolo ya existente usado por ejemplo en Zigbee, es un protocolo bajo demanda que soporta tanto unicast como multicast, en el que cuando se requiere transmitir un mensaje, se establece una ruta, 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, que es lo que nos interesa, 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]

Pese a ser el más usado, AODV presenta algunos problemas de escalabilidad con redes grandes, donde se vuelve caro pues los nodos requieren de almacenamiento para mantener las tablas de ruteo. Además, puede ocurrir congestión, pues los nodos están enviando constantemente paquetes 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.

## Modelos v2x, v2i y v2v

Dentro de la comunicación vehicular se presentan diferentes modelos con diferentes aplicaciones, el primero es v2i o comunicación de vehículos con infraestructura, la cual desempeña el rol de coordinador, donde la preocupación principal está en la latencia, como en la cobertura. Mientras v2v se refiere a la comunicación entre vehículos, aquí no hay un nodo central, con una funcionalidad similar a Bluetooth. Por último, el objetivo final v2x o comunicación entre vehículos contra todo, que no solo abarcaría los anteriores modelos, pero también espera comunicarse con los celulares de los peatones. En la figura 6-1 se presentan algunos ejemplos de las aplicaciones que se pueden usar y el tipo de comunicación que requerirían.

Tabla

Descripción generada automáticamente con confianza media

Figura ‑ [Hartenstein y Laberteaux 2010] Algunas aplicaciones y su modelo sugerido

Aquí nos centraremos en el modelo v2v donde hay dos principales estándares dominantes: DSRC basado en IEEE 802.11p (contempla capas física y MAC), y c-v2x de la 3GPP que usa tecnología celular basada en 4G LTE y 5G.

## DSRC

Diagrama

Descripción generada automáticamente

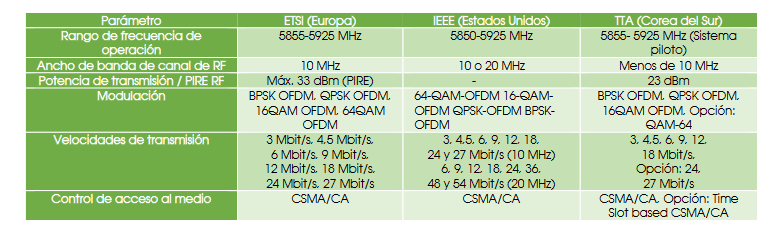
Figura ‑ [Hartenstein y Laberteaux 2010] Capas del estándar dsrc, se compone de dos partes principales, el estándar IEEE 802.11p e IEEE 1609 o wave

### 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 servició que incluye comunicaciones V2V y V2I, con fines de seguridad, como advertir posibles colisiones.

Como se observa en la figura 6-3, 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 de acceso al medio. [«IDENTIFICACIÓN DE NECESIDADES DE ESPECTRO PARA SISTEMAS DE TRANSPORTE INTELIGENTE EN LA BANDA 5850-5925 MHz» 2021]

Figura ‑ [«IDENTIFICACIÓN DE NECESIDADES DE ESPECTRO PARA SISTEMAS DE TRANSPORTE INTELIGENTE EN LA BANDA 5850-5925 MHz» 2021]



Dentro de la capa física se trata la modulación que corresponde a la representación de datos binarios para su transmisión en un medio físico, la información es colocada sobre una señal portadora en la banda correspondiente, existen distintos métodos, desde los más sencillos como modulación por amplitud o en frecuencia, a otros más complejos como QAM y OFDM que permiten transmitir una mayor cantidad de datos, pero son más vulnerables al ruido.

La modulación OFDM o multiplexación por división de frecuencia ortogonal se basa en la suma de N modulaciones lineales tipo QAM u otra modulación, con N portadoras diferentes. De forma que la señal queda modulada en fase y amplitud si se usa QAM, pero dispersa a lo largo de todo el ancho de banda, con las diferentes portadoras queda también modulada en frecuencia. El ancho de banda completo se divide en intervalos, al que cada uno se le asigna una modulación.

En la imagen 6-4 se observa la división del ancho de banda en subportadoras, contra una sola portadora, el número de divisiones se determina según el ancho de banda, la velocidad de los datos y la duración del símbolo, quedan separadas como: N=1/T.

Diagrama

Descripción generada automáticamente

Figura ‑ OFDM en el dominio de la frecuencia [Cabrera y Tarrés, s. f.]

OFDM permite un mejor aprovechamiento del espacio al permitir que se sobrelapen sus portadoras, siempre que le frecuencia central quede libre sin interferencias, cada una con a forma de una función sinc. Para su demodulación es sencillo con la transformada rápida de Fourier. [Cabrera y Tarrés, s. f.]

### 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.

Encabezado MAC como se ve en la figura 6-5 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]

Tabla

Descripción generada automáticamente

Figura ‑ [Kenney 2011] encabezados de capa 2

CSMA/CA en su forma básica, un nodo que quiere transmitir empieza mandando un mensaje tipo request to send (RTS) que incluye datos como el total de datos a transmitir, los nodos objetivo responden mandando un mensaje clear to send (CTS) en caso de estar libres que incluye también el tamaño de la respuesta, de lo contrario mandan un mensaje de ocupado. El resto de los nodos, usando los mensajes RTS y CTS, calculan el network allocation vector (NAV) que es el tiempo en el que no transmitirán datos. A diferencia de solo CSMA sin CA, donde los nodos transmiten con cierta probabilidad, mediante el sensado del canal y el cálculo del NAV permite reducir las colisiones.

### Capa lógica

DSRC hace uso del estándar IEEE 802.2 control de enlace lógico, y se apoya del protocolo 802.3. Como se observa en la figura 6-6 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]

Diagrama

Descripción generada automáticamente

Figura ‑ [Kenney 2011] Encabezados de la subcapa llc y snap

### Capas superiores

El estándar “wireless access in vehicular environments” (WAVE) o IEEE 1609 son 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 el overhead de UDP/IPv6, con mensajes cortos llamados WSMP que permitan transmisiones de un solo salto eficientes, el contenido se observa en la figura 6-7, 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.

Diagrama

Descripción generada automáticamente

Figura ‑ [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]

## Contenido de los mensajes

Con los motivos de seguridad que se plantean en el proyecto se requiere comunicar la ubicación tanto del vehículo dada por GPS, como la localización aproximada de las detecciones, la comunicación de la velocidad calculada y el uso de bits para indicar las acciones suyas y de otros, como frenado y cambio de carril. Siguiendo un acercamiento de agregación, es decir, cada vehículo en lugar de retransmitir inmediatamente los mensajes debe modificarlos, agregar y comparar con su propia información, de forma que el área de datos sería de tamaño variable. Igualmente considerando un contexto como en geocasting no toda la información es útil, de forma que, al alejarse de cierta región, hay datos como localizaciones de otros vehículos que deben descartarse.

# Capítulo 7 Implementación de una red vanet

## Arquitectura y requisitos de la red vanet

Como se repasó en el capítulo anterior, no hay un modelo bien definido de comunicación colaborativa entre vehículos, por el contrario, cada compañía lleva a cabo pruebas e investigaciones sobre sus propios modelos, como facilitadores, tanto la IEEE y el 3GPP han sacado estándares que abarcan algunas de las necesidades que se deben de cumplir, además, de momento la tendencia ha sido reaprovechar tecnologías y métodos ya existentes como AODV o WiFi (IEEE 802.11). En base a esto, nuestro modelo está enfocado enteramente en aplicaciones de seguridad, los mensajes que serían de tamaño variable comparten información del vehículo propio, como de las detecciones, pensando en que no todos los nodos estén conectados.

El contenido de los mensajes incluiría: 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 y su velocidad; en el caso de las detecciones comunicar su posición actual, velocidad estimada, junto a sus próximos movimientos.

Esta información plantea 3 casos, el primero una actualización periódica de los datos, que los coches procesarían y compararían con su información, para después reenviar el mensaje en forma de un resumen actualizado. En segundo, el envío de mensajes de emergencia como que un coche frene o cambie de carril. Estos dos primeros son mensajes de tipo broadcast, para los que se considera que solo se retransmitan una vez. Por último, la posibilidad de conectarse de forma directa con otros coches (unicast) para la verificación de información.

En base a esto y como la prioridad es tener la menor latencia posible, se escoge un mecanismo la red vanet está basada en comunicación vehículo a vehículo (v2v), pero se puede incluir modelos v2i o v2x, para otros servicios como planeación de rutas. Dentro del modelo v2v, se escoge como protocolo de enrutamiento para la comunicación uno a uno AODV, mientras que se realiza la simulación sobre el estándar 802.11p (primera parte de DSRC), con soporte de CSMA/CA.

## Implementación

Para el diseño de la simulación 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.

Para la simulación de 802.11p, aparte de los ya mencionados, se usa SUMO, un paquete de simulación de tráfico que incluye el modelado de vehículos, transporte público y peatones, lo que nos permite diseñar nuestros escenarios, que incluye la creación de rutas, agregar y controlar nodos (vehículos). Para agregar este modelo se trabaja con VEINS, un framework de código abierto para ejecutar simulaciones de redes VANET, basado en dos simuladores: OMNeT++ y SUMO, y que añade además un conjunto de protocolos para la comunicación entre vehículos.

### AODV

Para la primera simulación, el proyecto se compone de 3 archivos principales, el primero es un .NED, que corresponde a la descripción de la topología de la red. 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 y 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.

Una vez descrita la red, se escribe un archivo .xml que contiene los obstáculos:

<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 requieren de ejecutar una aplicación para comunicarse, en este caso se usa una ya definida “PingApp”, que indica al nodo source que mande un ping al nodo destino.

\*.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, esto valores de acuerdo al estándar de DSRC. Además, como se hará en los resultados, se puede experimentar con la mínima potencia recibida.

\*.\*.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 representó como:

\*.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

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

\*.\*.routingApp.typename = "Aodv"

\*.\*.routingApp.activeRouteTimeout = 0.5s

\*.\*.routingApp.deletePeriod = 0.1s

En la figura 7-1 se observa la ventana al ejecutar la simulación:

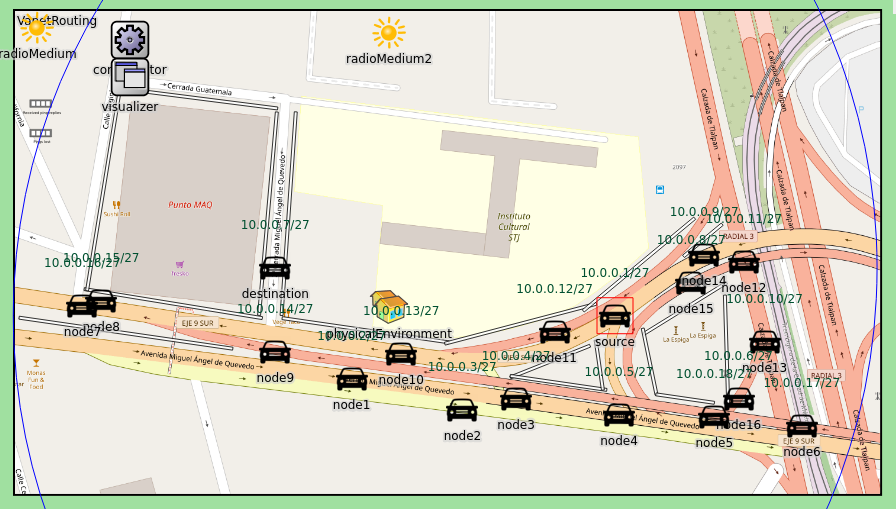


Figura ‑ Imagen de la simulación de aodv

### IEEE 802.11p

Aquí se comienza con el modelo en SUMO, aquí se trazan de forma gráfica las rutas, junto con los edificios. Sin embargo, no se pudieron agregar en omnett los obstáculos de esta forma, por lo que se usó el archivo .xml escrito en la simulación anterior.

Se escribe un archivo .launch con los programas de SUMO, mapa2.net.xml contiene los caminos, las intersecciones y la velocidad máxima en cada uno, rutas2.rou.xml son las rutas y los vehículos (para estos se define la hora de salida y la ruta que toman), walls.poly.xml son las construcciones, para cada una se debe definir que son de tipo “building”.Por último, el archivo hello2.sumocfg es creado por SUMO al ejecutarse y contiene la ubicación de los tres archivos anteriores.

<?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>

En el caso de los vehículos se asgregan en el archivos para las rutas:

<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 7-2 se observa la simulación corriendo en SUMO, los puntos amarillos son los vehículos.

Un dibujo de una cara con ojos y boca

Descripción generada automáticamente con confianza baja

Figura ‑ Imagen final en sumo

Una vez finalizada la simulación en simio, el siguiente paso es agregarla en omnet++, se puede partir del ejemplo incluido en veins\_inet/examples/veins\_inet. Se requieren agregar los archivos correspondientes a las rutas, la red, el config y el launch. En el caso de los obstáculos se requieren agregar módulos extras en el .ned para que funcionen, o incluirlo como en la primera simulación. El archivo .NED, al igual que el anterior incluye módulos para el radio, modelar el entorno y visualizadores, junto a un manager para controlar la comunicación.

En cuanto al archivo de configuración omnetpp.ini se agrega nuevamente el archivo .ned, y el tiempo de simulación:

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 el anterior se requiere de una aplicación que más adelante se va a explicar los cambios realizados, con una interfaz inalámbrica.

\*.node[\*].numApps = 1

\*.node[\*].app[0].typename =

"org.car2x.veins.subprojects.veins\_inet.VeinsInetSampleApplication"

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

A continuación, 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. Mientas 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[\*].ipv4.configurator.typename = "HostAutoConfigurator"

\*.node[\*].ipv4.configurator.interfaces = "wlan0"

\*.node[\*].ipv4.configurator.mcastGroups = "224.0.0.1"

El modulo “VeinsInetMobility” permite rastrear la posición del nodo, aquí se usa el manager para agregar los archivos de SUMO. Para los obstáculos se agregan con el modulo 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"

En el caso de las aplicaciones se tratan de módulos que implementan comportamientos a cada nodo, para generar patrones de tráfico, para nuestro modelo se tienen dos métodos principales: el método startApplication() con el cuál se controlan los nodos para el envío de un mensaje, para generar eventos y con veins controlar su comportamiento (velocidad, rutas, etc.); el otro método es processPacket para definir el comportamiento de los nodos cuando reciben un mensaje.

Se tienen dos casos, el primero representa el envío de mensajes de emergencia, aquí se representa un accidente, como que un coche se incorpore a otro carril, el segundo caso es el envío periódico de mensajes para mantener informados a los nodos. En ambos casos, los nodos para procesar el paquete deben de retransmitirlo una vez, para el segundo caso, como se reciben múltiples mensajes, debe de tener un buffer para saber qué mensajes ya recibió.

Para controlar el nodo se usa if (getParentModule()->getIndex() == 0) para definir qué acciones realizar, con traciVehicle se puede puede controlar el vehículo como cambiar la velocidad (traciVehicle->setSpeed(0);) o ruta. Para programar el envío de un mensaje, se tiene un módulo aparte que representa el mensaje junto con su contenido, se define el tamaño, se rellenan sus variables, luego se crea el paquete y se le agrega el payload, con sendPacket se envía a todos.

auto payload = makeShared<VeinsInetSampleMessage>();

payload->setChunkLength(B(100));

payload->setRoadId(traciVehicle->getRoadId().c\_str());

timestampPayload(payload);

auto packet = createPacket("ChangingLane");

packet->insertAtBack(payload);

sendPacket(std::move(packet));

Por último, con una función callback se puede definir cuando ocurren los eventos:

timerManager.create(veins::TimerSpecification(callback).onesh

otAt(SimTime(70150, SIMTIME\_MS)));

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

auto payload = pk->peekAtFront<VeinsInetSampleMessage>();

Se puede extraer la información del mensaje y según el contenido realizar alguna acción (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 retiran estos mensajes por su antigüedad,

En la figura 7-3 se muestra la simulación final, para ejecutarla debe estar corriendo veins (se abre la aplicación veins\_launchd) y se lanza el archivo omnetpp.ini.

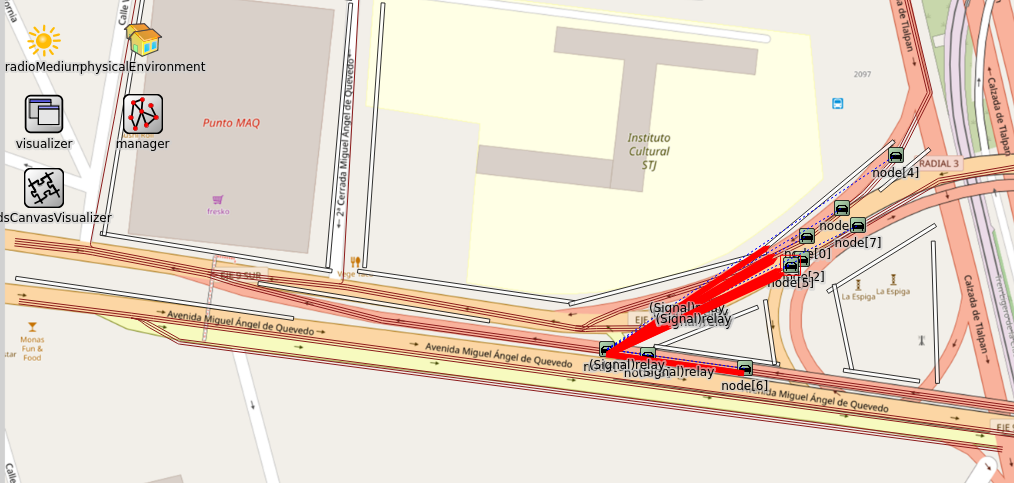


Figura ‑ Imagen de la simulación corriendo en omnet++, se observa un nodo retransmitiendo a todos un mensaje

## Resultados

Para la primera simulación se preparan dos escenarios principales con diferente densidad de nodos que se observan en las figuras 7-4, para los que se experimenta los resultados del protocolo AODV al intentar mandar un mensaje ping entre dos nodos, y se mide el porcentaje de mensajes recibidos, tasa de errores 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 ‑ Simulación 1 experimentando el desempeño con diferente densidad de nodos

Los resultados varían según una serie de parámetros que configuran tanto el medio físico como la capa MAC en la simulación. En el medio físico, como se parte del estándar 802.11p se tiene que la frecuencia de transmisión es en 5.9 GHz, y la potencia máxima de 200 mW (23 dBm), además, es importante la sensibilidad del receptor, para la que se eligió un valor normal que es de -85 dBm.

Con esto y tomando como ganancia de las antenas 0 dBm, con el modelo de free space path loss, se tiene una cobertura con línea de visión de 1 km de radio, distancia que parece ser más que suficiente para medidas de seguridad. 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, pues, un intervalo pequeño significa mayor inundación en la red de mensajes, mientras que por otro lado, un tiempo grande tiene el problema de que la topología cambia con los nodos moviéndose, 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.

En cuanto al interés de variar el intervalo de transmisión entre mensajes ping, es que AODV se trata de un protocolo de inundación reactivo, de forma que un intervalo alto siempre significará mandar mensajes para encontrar una ruta al nodo destino, pero un intervalo bajo puede o no llevar a la inundación de la red, con la topología cambiando según la velocidad y densidad de los nodos puede llevar a retrasos en la red si se necesitan buscar constantemente nuevas rutas, o se pueden usar las ya descubiertas. De forma que en estos casos conviene que el intervalo de envío de mensajes sea adaptable, según las características de la red.

En esta simulación 1 se representa al 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, con AODV un nodo empieza buscando una ruta en sus tablas al nodo destino, en caso de no tener una ruta fresca, manda mensaje por broadcast a los nodos alrededor que responden si conocen una ruta o por el contrario retransmiten a otros nodos para preguntar, una vez adquirida la ruta manda el mensaje y espera una respuesta.

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

En la figura 8-11 se observa las posiciones iniciales y finales de los nodos, a propósito, se diseñó el primer escenario para que hubiera un momento en que los nodos no tuvieran un camino para comunicarse, esto será importante cuando se reduzca el intervalo entre mensajes, pues deberán mandarse mensajes del protocolo AODV para buscar nuevas rutas. En la primera simulación que dura 12 segundos, el nodo fuente intenta enviar 12 mensajes, pero solo recibe respuesta de 8, lo que es una tasa de pérdida del 33.33% con un rtt promedio de 436.4 ms. En la gráfica 7-5 se observan los mensajes del protocolo AODV enviados por cada nodo, corresponden al descubrimiento de rutas e informes de errores cuando se rompe un link. 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, mientras cada nodo debe enviar entre 7 a 15 mensajes en la duración de la simulación. En la misma figura se observan los mensajes recibidos por cada nodo, en naranja los mensajes ping que se transmitieron correctamente (8), la cantidad de mensajes recibidos depende de la posición del nodo, pero los cercanos reciben y procesan alrededor de 80 mensajes cada uno.



Figura ‑ A la izquierda la posición inicial, a la derecha la posición final

Todos estos mensajes transmitiéndose en la red tiene dos impactos importantes, hay colisiones cuando dos nodos transmiten simultáneamente, pero también significa retrasos a la hora de 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 8-13), 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, el nodo fuente tarda 1.3 ms, mientras la segunda cuenta después de crear el paquete y que se transmita en el medio (junto a los tiempos de contención), que para el nodo fuente es 1 ms.

Figura 8‑13 a la izquierda "in progress frames" a la derecha "pending queue"

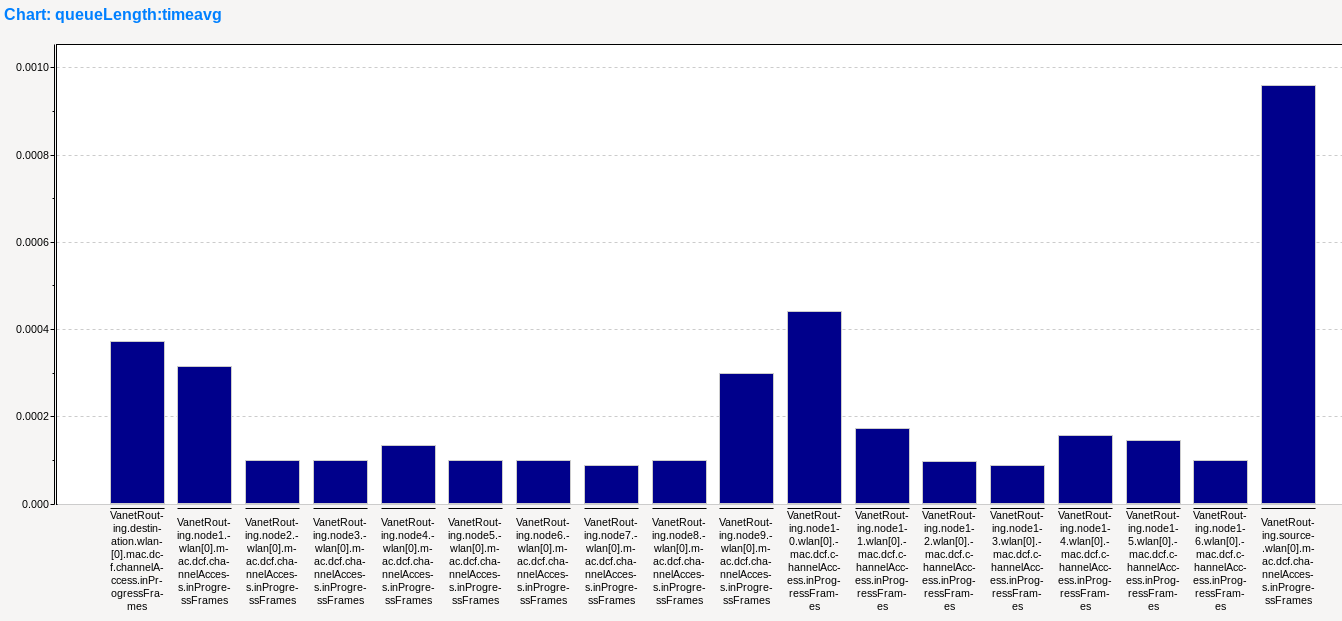
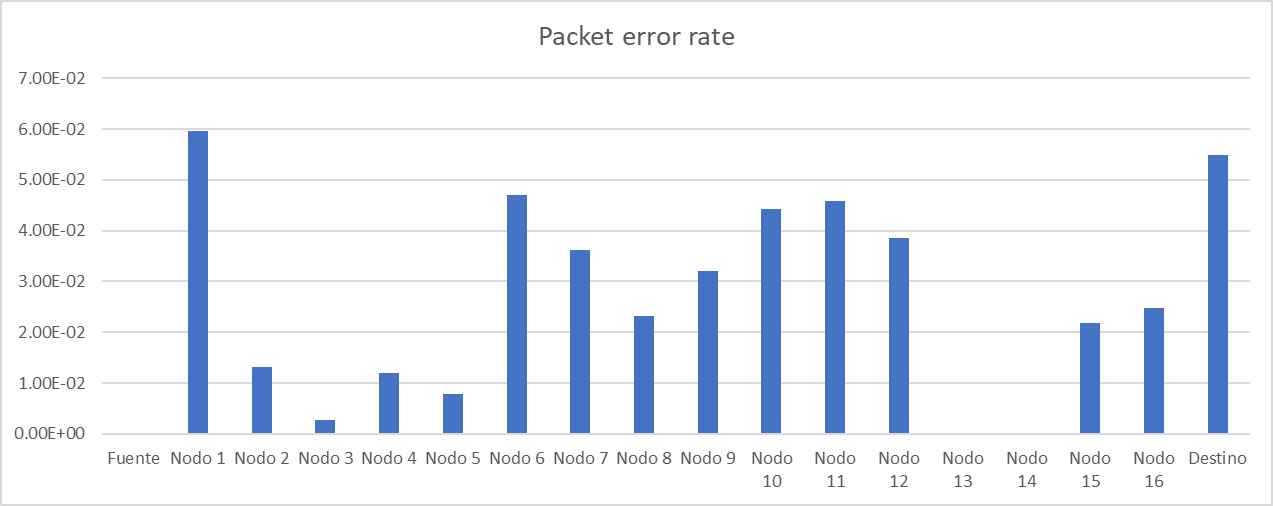
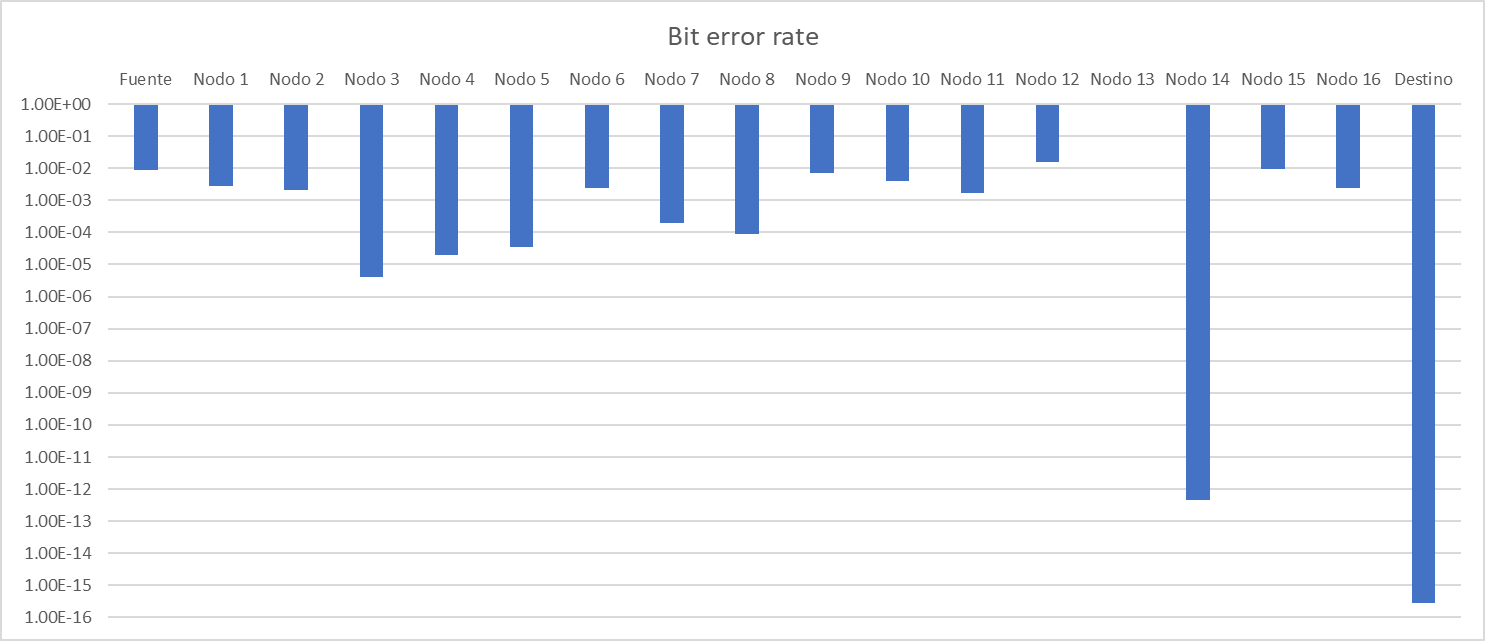
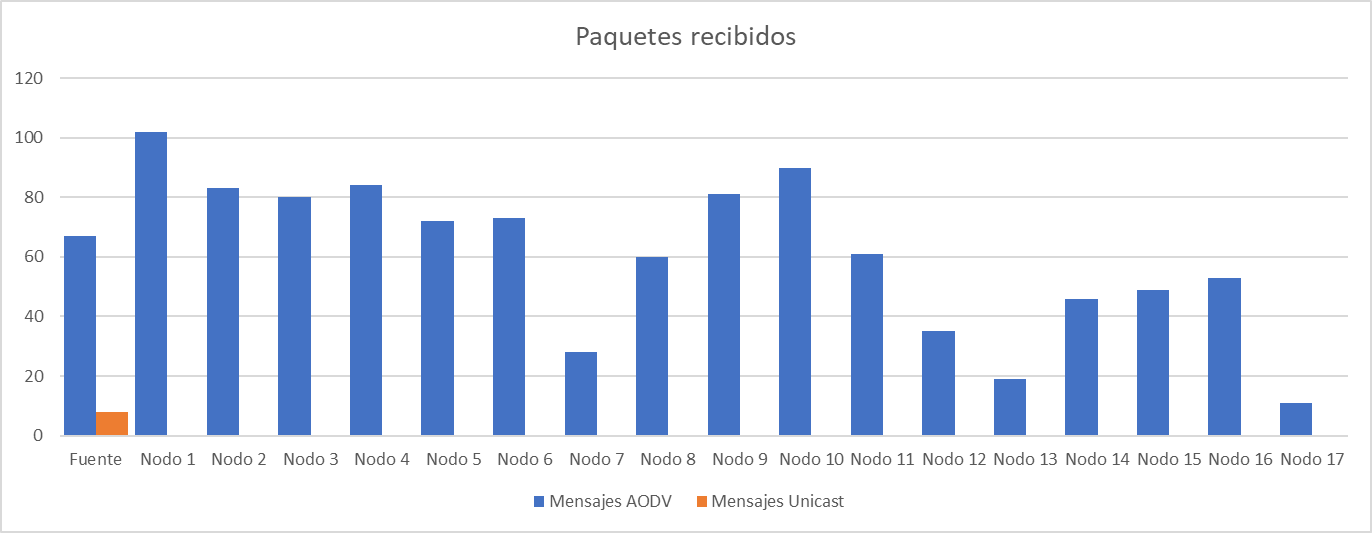
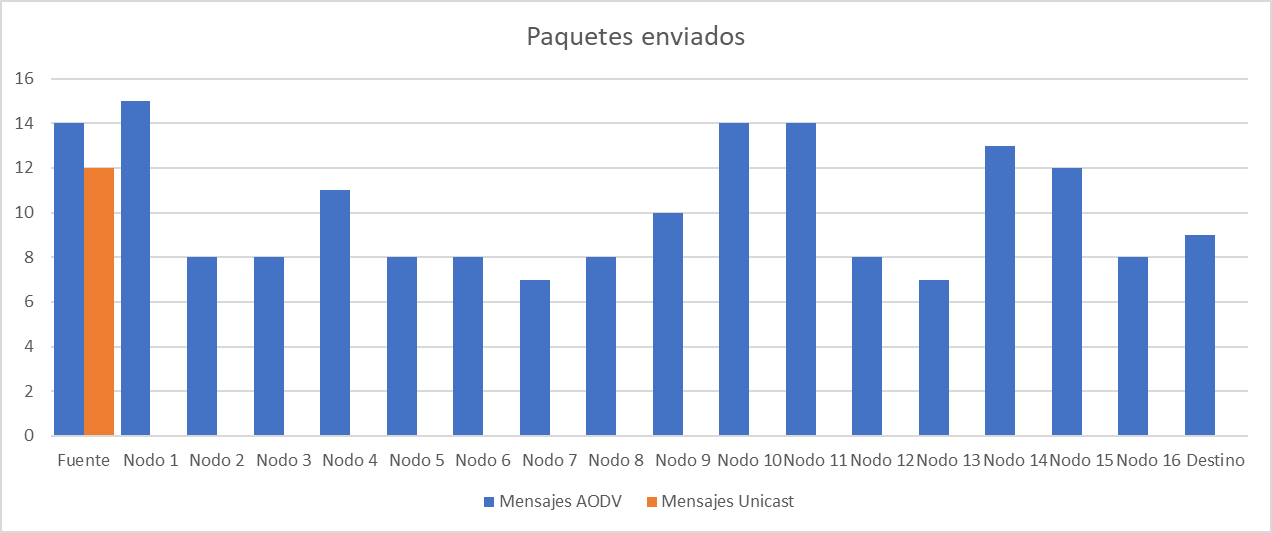
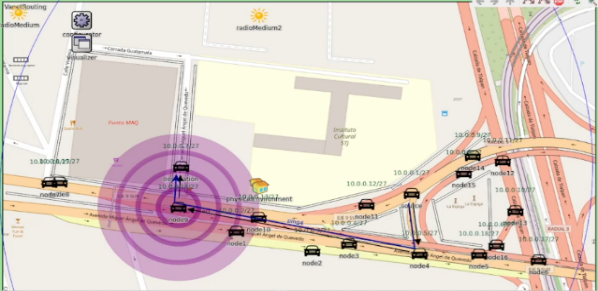


Figura 8‑12 resultados de la simulación 1



En la figura 8-14 se muestra visualmente la simulación.

Figura 8‑14 Rutas formadas por el protocolo aodv



A partir del primer caso analizamos y comparamos con las demás simulaciones y se presentan algunas gráficas interesantes, otros parámetros que se pueden variar y experimentar es por ejemplo con la potencia de transmisión o sensibilidad del receptor, lo que significa un menor radio de cobertura y los mensajes requieren más saltos para llegar al objetivo y las rutas son más fáciles de romperse lo que significa que menos mensajes se reciben, puede ser interesante observar el impacto de la inundación de la red.

Por otro lado, otro experimento puede ser subir la taza de transmisión, por ejemplo, a 24 mbps, se reciben 8 paquetes de 12 (se pierden el 33%) con un mismo rtt que en el caso 1, los cambios son que lleva a una menor inundación en la red de mensajes, al mismo tiempo que sube la tasa de errores, con mayores velocidades se esperaría ver que los tiempos de encolamiento crecieran y la llegada de paquetes bajara de forma importante. Es importante mencionar que, en las gráficas presentadas de paquetes enviados y recibidos, en color azul considera solo los mensajes del protocolo AODV, no cuenta como enviados los pings.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Velocidad de los nodos | Intervalo entre mensajes | Mensajes transmitidos | Mensajes recibidos | Porcentaje de pérdidas | RTT |
| 30 km/h | 1s | 12 | 8 | 33.3% | 436.4 ms |

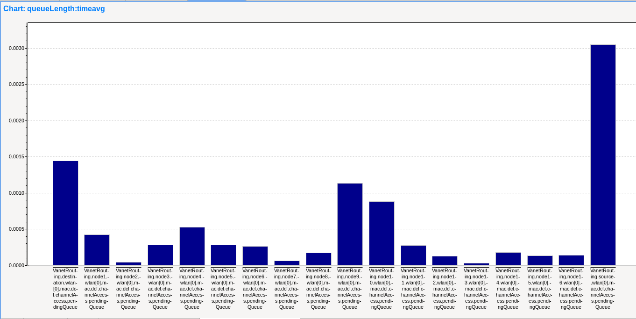
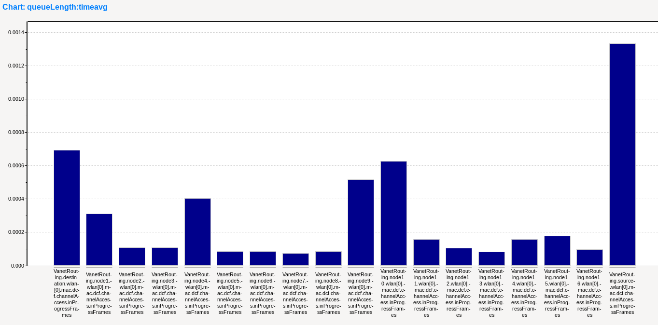
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 km/h | 0.5 s | 24 | 15 | 37.5% | 617.6ms |
| Reduciendo el intervalo entre mensajes a 0.5 s, con los cambios en la topología y el tiempo de vida de las rutas, los nodos deben buscar en ocasiones nuevas rutas mediante AODV, lo que no significa un aumento grande de mensajes en la red, únicamente 1 o 2 más por nodo, siendo los nodos 9 a 11 son los de mayor aumento en # mensajes, igualmente aumenta el número de colisiones y errores en paquetes, igualmente el tiempo en acceso al medio aumenta lo que lleva a que el RTT lo haga también. | | | | | |

Gráfico, Gráfico de barras

Descripción generada automáticamenteGráfico, Gráfico de barras

Descripción generada automáticamenteGráfico, Gráfico de barras

Descripción generada automáticamente



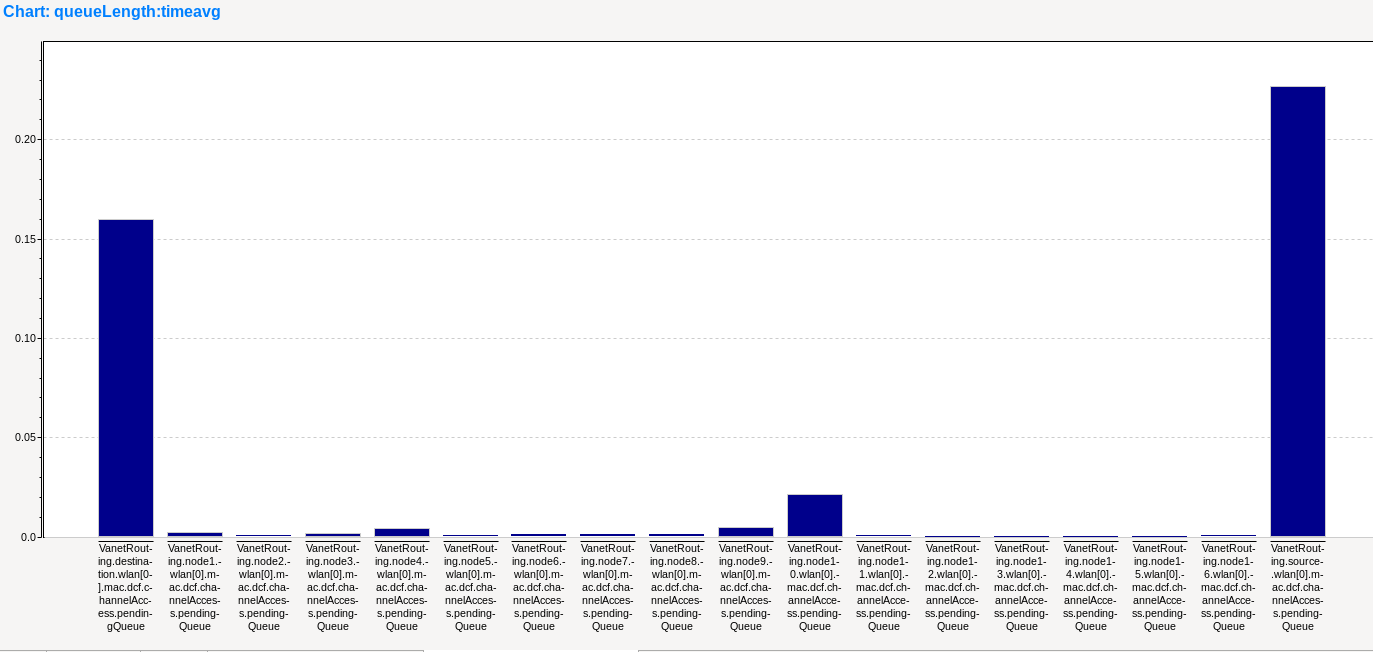
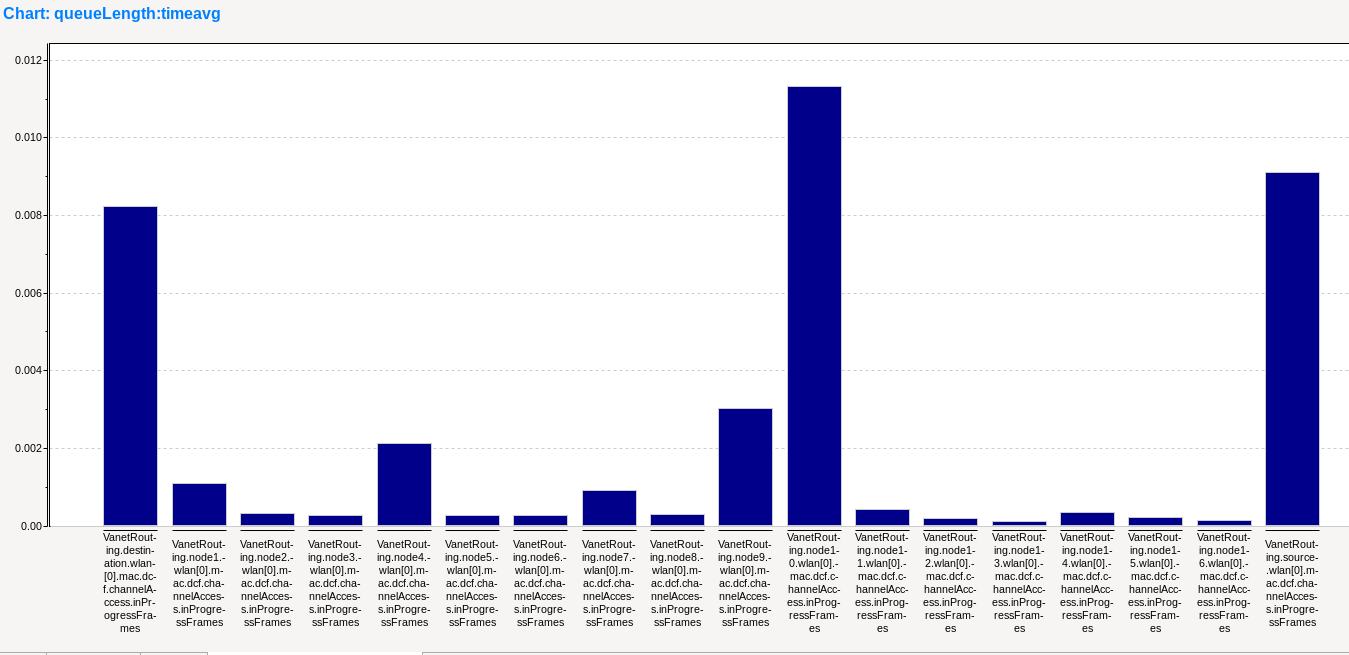
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 km/h | 0.2 s | 60 | 20 | 66.6% | 76.65 ms |
| Pese a la alta tasa de paquetes perdidos, al reducir nuevamente el intervalo entre mensajes, se logra una disminución en el número de mensajes en la red, debido a que el protocolo no requiere redescubrir nuevas rutas todo el tiempo, esto permite que el RTT y el tiempo de acceso al medio disminuyan bastante, para la fuente apenas 0.5 ms para acceder al medio. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 km/h | 0.1 s | 120 | 85 | 29.16% | 2844.43 ms |
| Si bien logra reducir la tasa de paquetes perdidos, la cantidad de paquetes en la red es muy alta, lo que lleva a que el RTT crezca considerablemente hasta 2.8 segundos, esto se debe y como se verá más adelante a que al haber una baja densidad en la red, los links se rompen y se requiere encontrar nuevas rutas, momento en que se debe inundar la red, causando altos retrasos en los paquetes, qué para motivos de seguridad no es adecuado. | | | | | |

Gráfico, Gráfico de barras

Descripción generada automáticamenteGráfico

Descripción generada automáticamente con confianza media



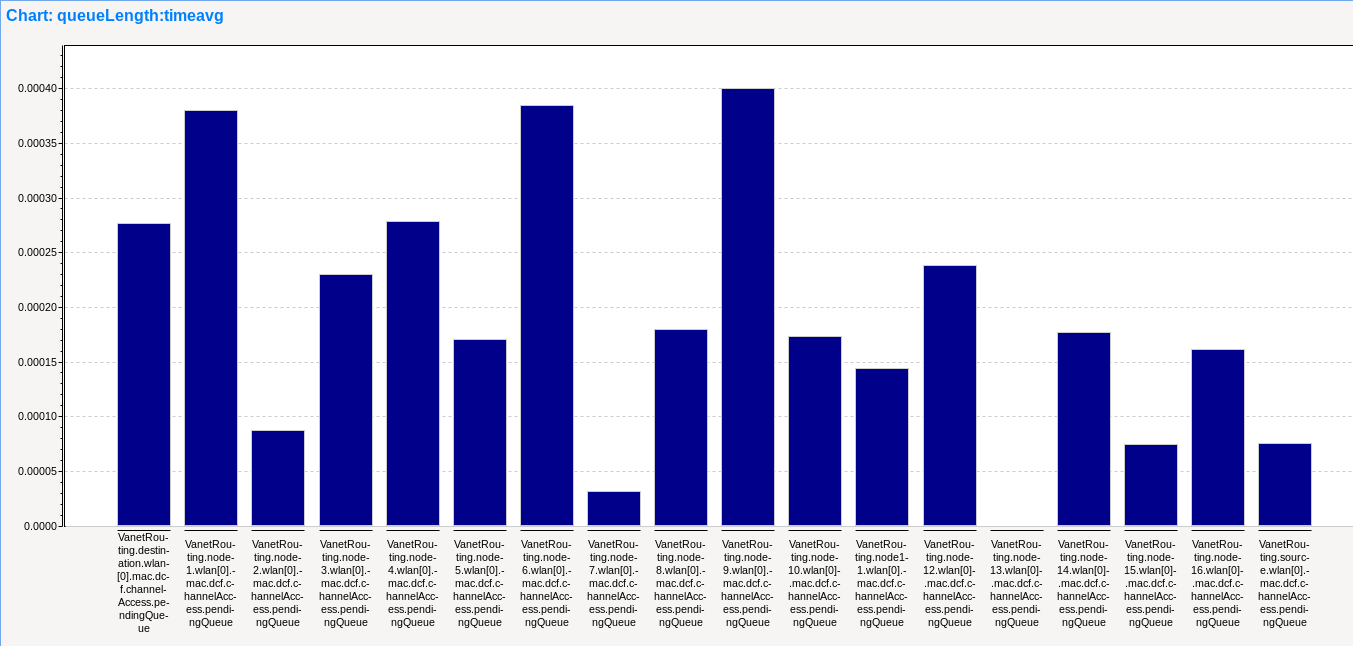
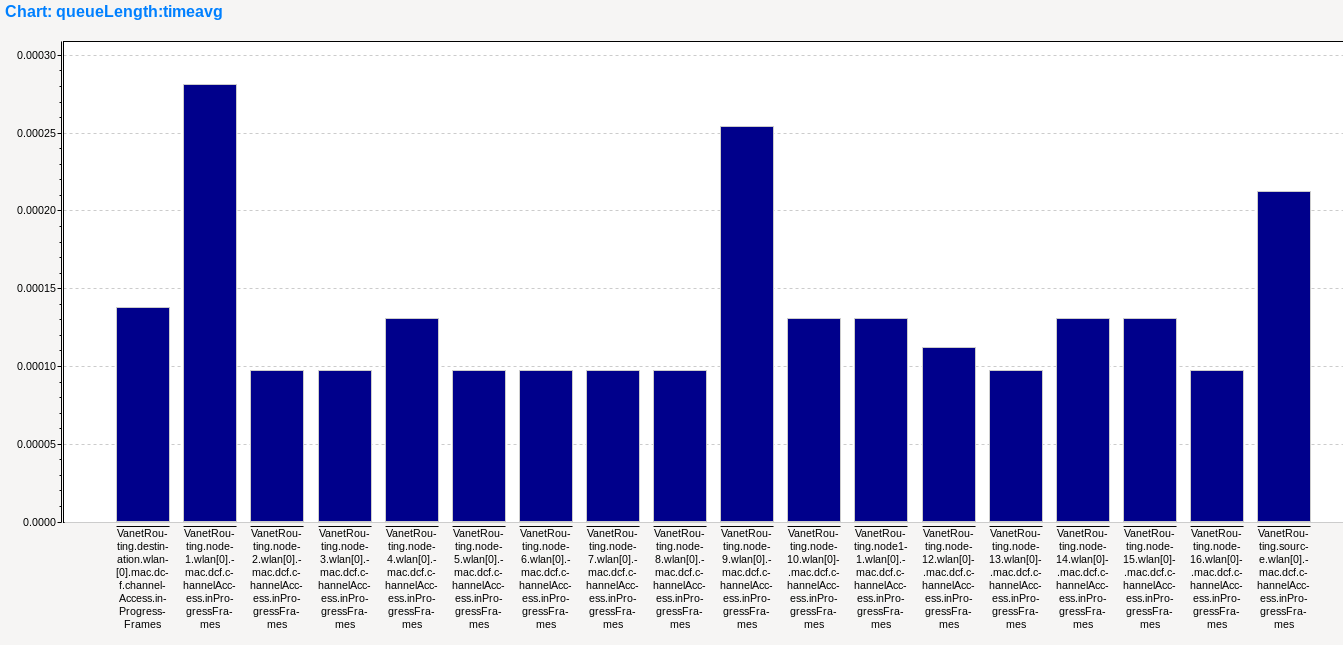
In progress frames Pending acces queue

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 1 s | 8 | 2 | 75% | 409.1ms |
| Al aumentar la velocidad en los nodos se observa una drástica caída en los mensajes recibidos comparado con los 8 que se transmitieron correctamente en el caso 1, el RTT se mantiene cercano. Con el cambio de la topología, la inundación de la red con mensajes del protocolo AODV aumenta considerablemente, mientras antes enviaban no más de 18 mensajes ahora envían entre 20 y 40, recibiendo cada nodo entre 150 y 200 mensajes, los tiempos de encolamiento se incrementan ligeramente entre 0.1 y 0.4 ms | | | | | |

Gráfico, Gráfico de barras

Descripción generada automáticamenteGráfico, Gráfico de barras

Descripción generada automáticamente



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 0.5 s | 16 | 5 | 68.75% | 135.57ms |
| Al reducir el intervalo entre mensajes a 0.5, las rutas, en algunos casos, duran lo suficiente para mandar más mensajes, esto permite reducir el número de mensajes AODV en la red, aquí se observa un aumento en las tasas de errores. | | | | | |

Gráfico, Gráfico de barras

Descripción generada automáticamenteGráfico, Gráfico de barras

Descripción generada automáticamenteGráfico, Gráfico en cascada

Descripción generada automáticamenteGráfico, Gráfico de barras

Descripción generada automáticamente

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 0.2 s | 40 | 12 | 70% | 70.75 ms |
| Con un intervalo de 0.5 y 0.2s las rutas AODV se mantienen vivas, por lo que no se requiere inundar la red, igual que el anterior, mantiene un RTT y tiempo de encolamientos bajos. Los nodos moviéndose a alta velocidad causan que se pierdan bastantes mensajes y tienen tasas de errores altas. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 0.1 s | 80 | 13 | 83.75% | 491.5 ms |
| Se repite el mismo caso con una velocidad de 30 km/h, aumentan las colisiones y errores en los mensajes, mientras que aumentan el tiempo RTT. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 1 s | 5 | 2 | 60% | 171.9 ms |
| A subir la velocidad a 80 km/h, nuevamente se observa un aumento en los mensajes inundando la red. | | | | | |

Gráfico, Gráfico de barras

Descripción generada automáticamenteGráfico, Gráfico de barras

Descripción generada automáticamente

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 0.5 s | 11 | 0 | 100% | - |
| En este caso no se lograron transmitir correctamente mensajes, probablemente por los cambios en la topología de la red y de mensajes, por lo que los tiempos de encolamiento debieron aumentar. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 0.2 s | 27 | 15 | 44.4% | 345.3 ms |
| Consigue la mejor tasa de envíos, gracias a que puede aprovechar las rutas existentes, se observa como impacta el aumento en la velocidad de los nodos, por ejemplo, en el RTT, mientras que a 30 y 50 km/h, el RTT promedio era de 70 ms, aquí sube a 345 ms. | | | | | |

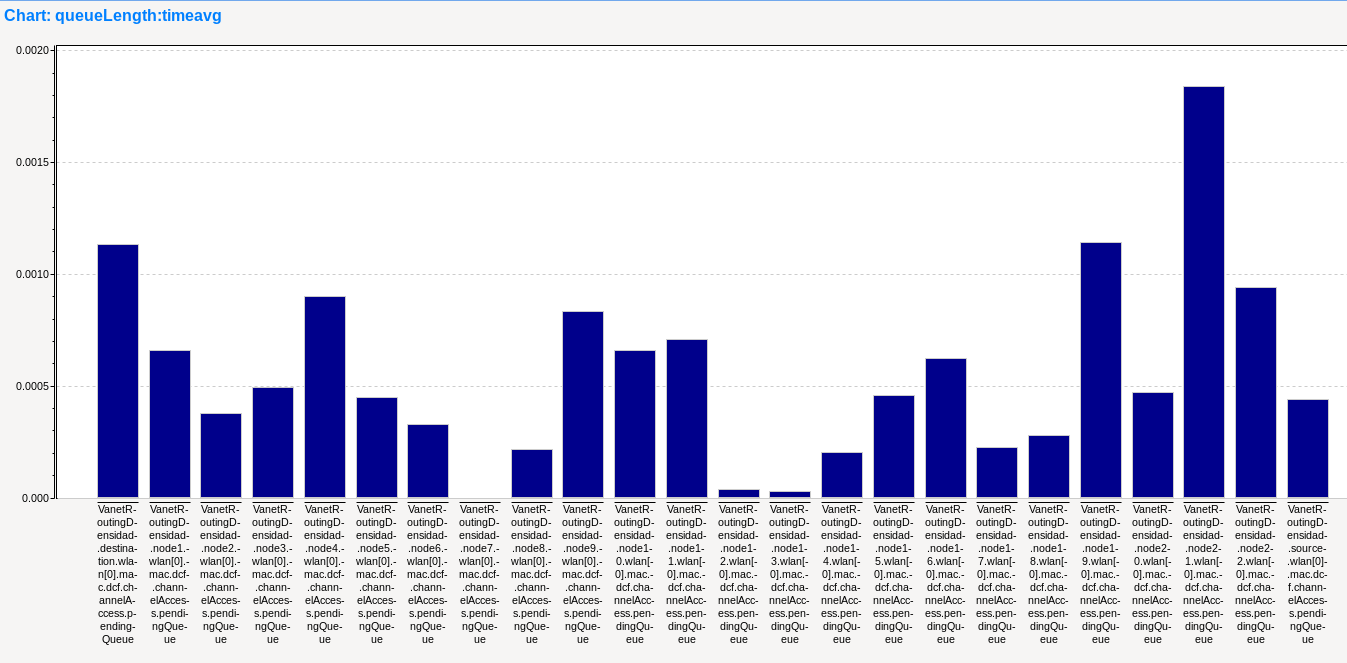
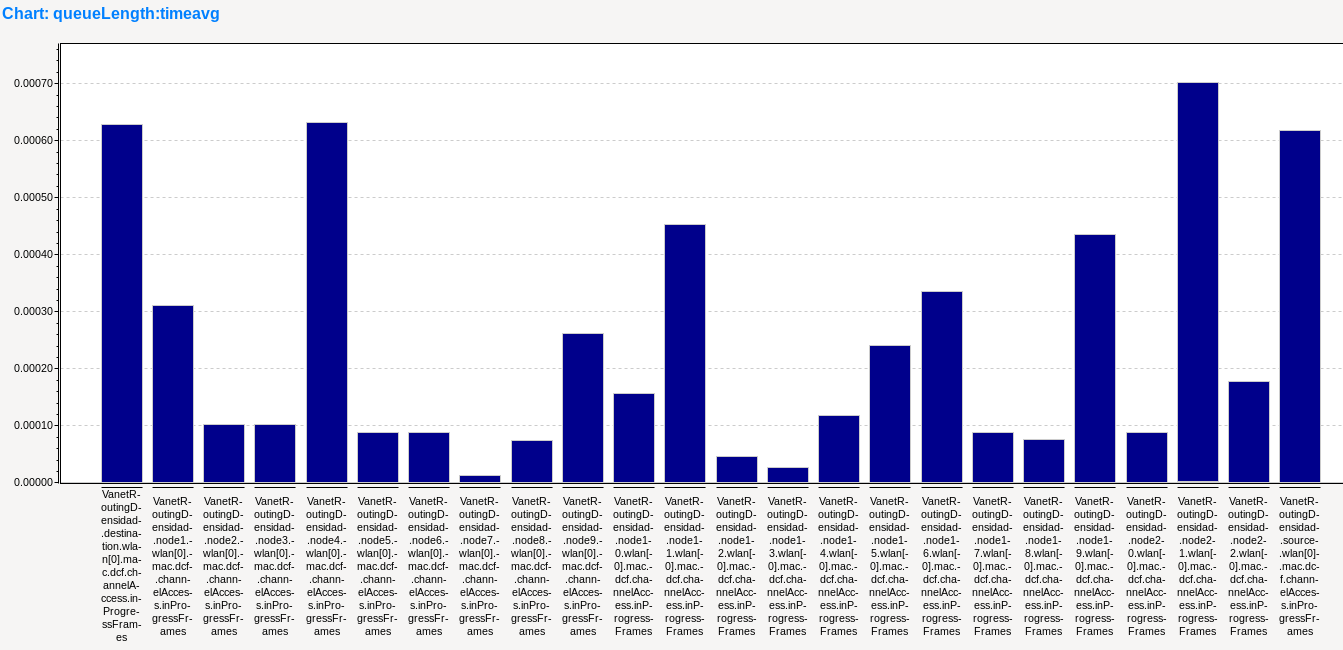
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 0.1 s | 55 | 5 | 90.9% | 321.1 ms |
| Repitiendo los casos anteriores, la tasa de pérdidas de mensajes aumentan de forma considerable. | | | | | |

Los resultados nos mostrarían que el escenario óptimo sería con un intervalo de 0.2 segundos entre mensaje, para aprovechar las rutas activas, sin inundar la red. Cambiando al escenario 2, en el que se aumenta la densidad de nodos de 16 a 22, esto significará no solo que siempre haya una ruta, sino que además puede ser óptima comparado con el escenario 1, la reducción en el intervalo entre mensajes a 0.1, puede ser posible debido a que es más fácil mantener las

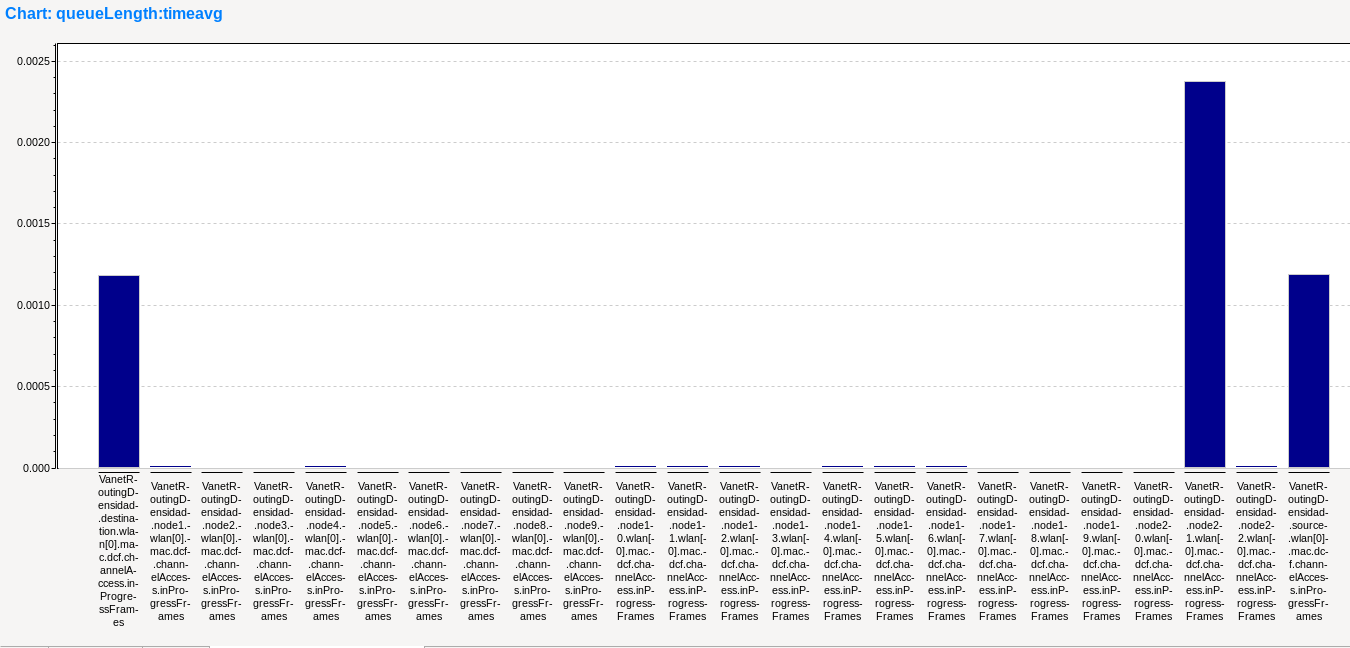
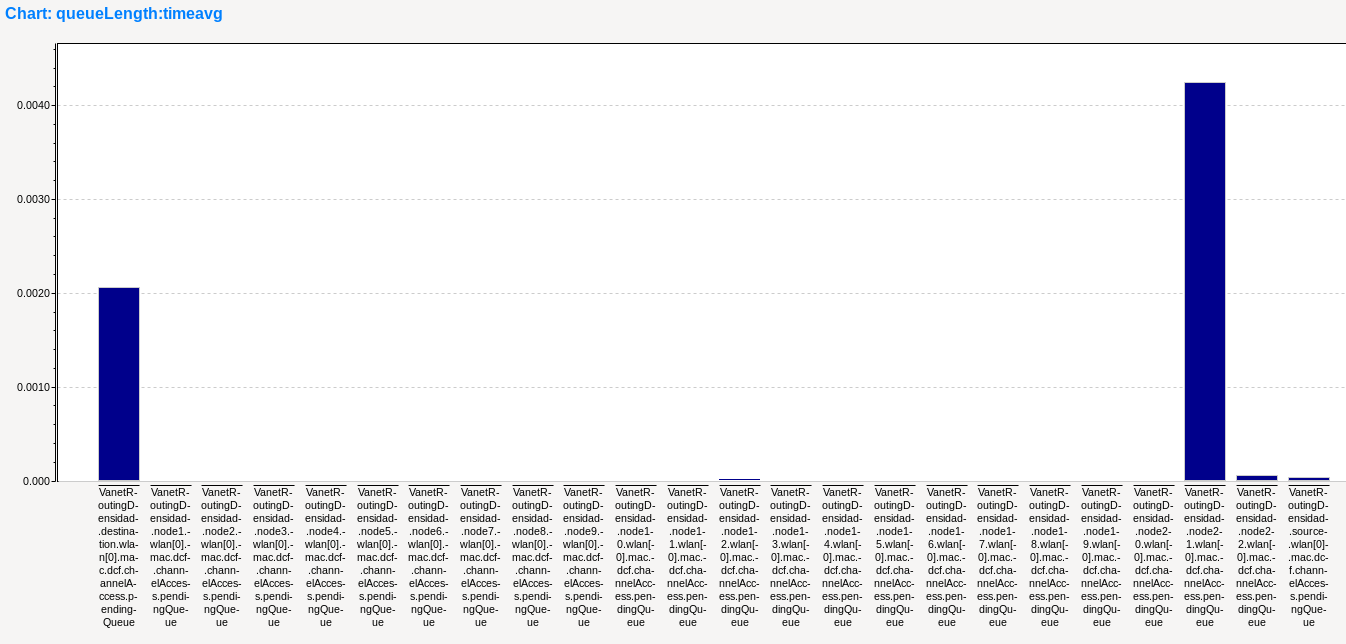
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Velocidad de los nodos | Intervalo entre mensajes | Mensajes transmitidos | Mensajes recibidos | Porcentaje de pérdidas | RTT |
| 30 km/h | 1s | 12 | 11 | 8.33% | 6.85 ms |
| Comparando con el caso 1, los nodos envían menos mensajes y participan en menor medida en la red, lo que lleva a que no aumenten los mensajes recibidos. Además, se observa una disminución en la tasa de errores. El mayor cambio se da en los tiempos de encolamiento que disminuyen de forma considerable, junto al RTT.  Gráfico, Gráfico de barras, Histograma  Descripción generada automáticamenteGráfico, Gráfico de barras, Histograma  Descripción generada automáticamenteGráfico, Gráfico en cascada  Descripción generada automáticamenteGráfico, Gráfico de barras  Descripción generada automáticamente | | | | | |

rutas, además, no todos serán partícipes en la comunicación y búsqueda de rutas.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 km/h | 0.5 s | 24 | 23 | 4.16% | 48.01ms |
| Igual que en el caso 1, las rutas no permanecen suficientemente tiempo activas, por lo que se requiere inundar la red con mensajes, lleva a una subida en los paquetes recibidos por nodo, un aumento en el RTT y que los tiempos promedios de encolamiento se dupliquen o hasta tripliquen.  Gráfico, Histograma  Descripción generada automáticamenteGráfico, Gráfico de barras, Histograma  Descripción generada automáticamente | | | | | |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 km/h | 0.2 s | 60 | 60 | 0% | 2.18 ms |
| Las rutas de los nodos se mantienen activas en todo momento, los nodos encuentran rápidamente una ruta optima, por lo que no requieren de transmitir muchos mensajes (algunos ni siquiera participan), las tasas de errores son bajas, y el tiempo de encolamiento es muy bajo.  Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico  Descripción generada automáticamenteGráfico  Descripción generada automáticamente | | | | | |



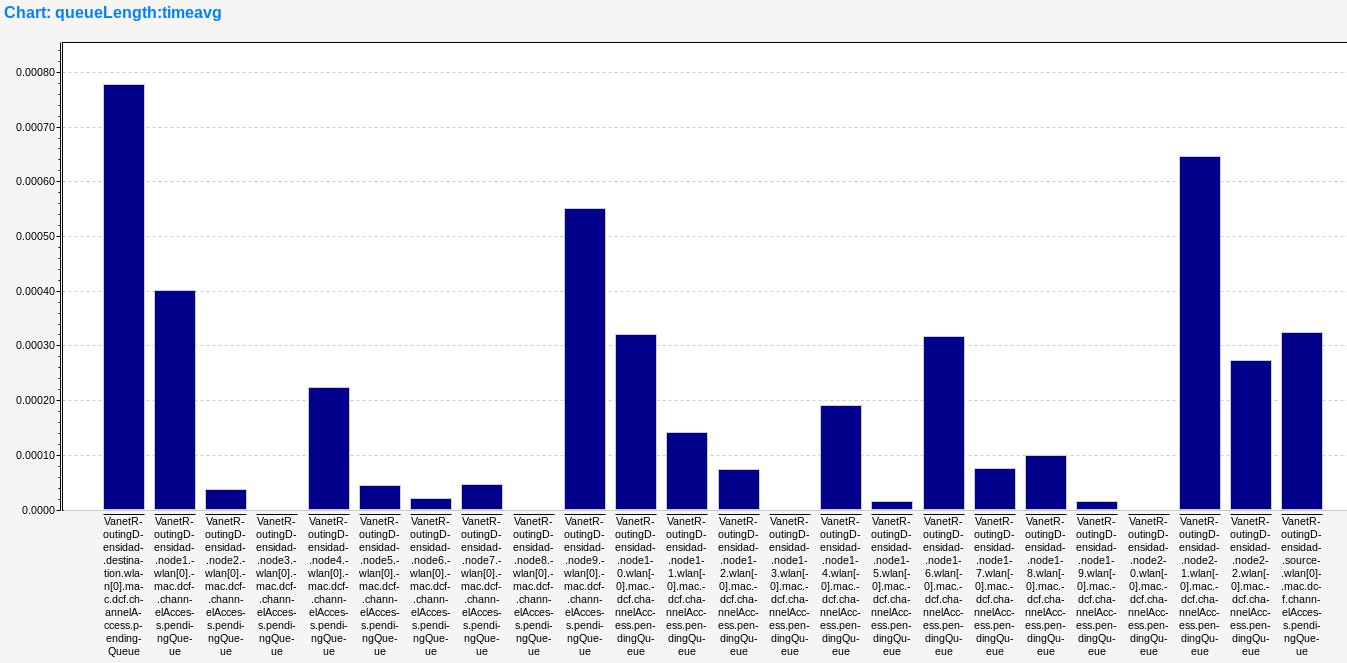
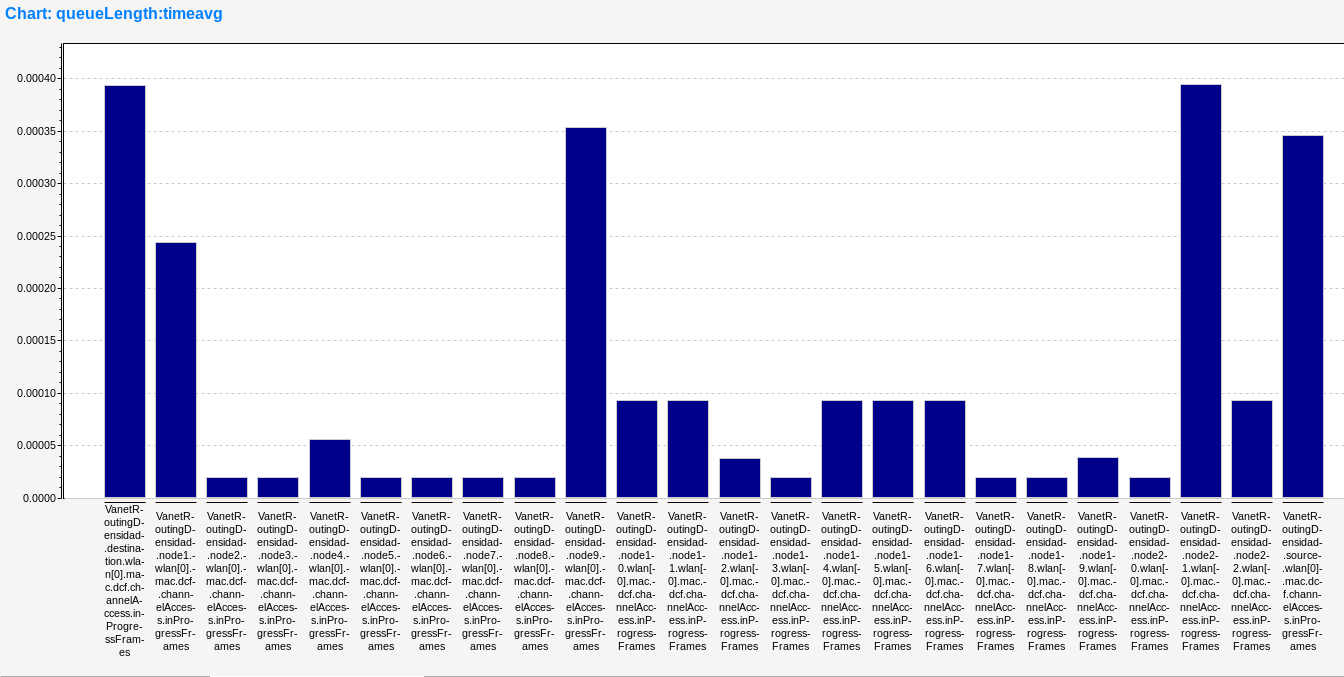
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 km/h | 0.1 s | 120 | 120 | 0% | 2.05 ms |
| Se repiten los resultados que el caso anterior, no aumenta el número de mensajes en la red, al mantenerse la ruta activa no hay colisiones ni altos tiempos. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 1 s | 8 | 8 | 0% | 48.49ms |
| El RTT aumenta al tener que buscar nuevas rutas y romperse los links, logra transmitir todos los mensajes. | | | | | |

Gráfico, Gráfico de barras, Histograma

Descripción generada automáticamenteGráfico, Histograma

Descripción generada automáticamente



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 0.5 s | 16 | 15 | 6.25% | 5.11 ms |
| Logra transmitir correctamente los mensajes, sin embargo, las rutas si se actualizan regularmente, comparado con 30 km/h donde solo requerían mandar 1 o 2 mensajes, se requieren mensajes de mantenimiento a la ruta, lo que aumenta muy ligeramente el RTT. | | | | | |

Gráfico

Descripción generada automáticamenteGráfico, Histograma

Descripción generada automáticamente

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 0.2 s | 40 | 40 | 0% | 2.29 ms |
| No presenta el problema que el anterior, 0.2 es un buen intervalo entre mensajes para mantener las rutas activas y no requerir inundar la red con mensajes del protocolo AODV. | | | | | |

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteGráfico

Descripción generada automáticamente

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 0.1 s | 80 | 80 | 0% | 2.11 ms |
| No hay diferencias con el caso anterior. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 1 s | 5 | 4 | 20% | 5.98 ms |
| A comparación del escenario 1, a alta velocidad los nodos consiguen transmitir correctamente la información. | | | | | |

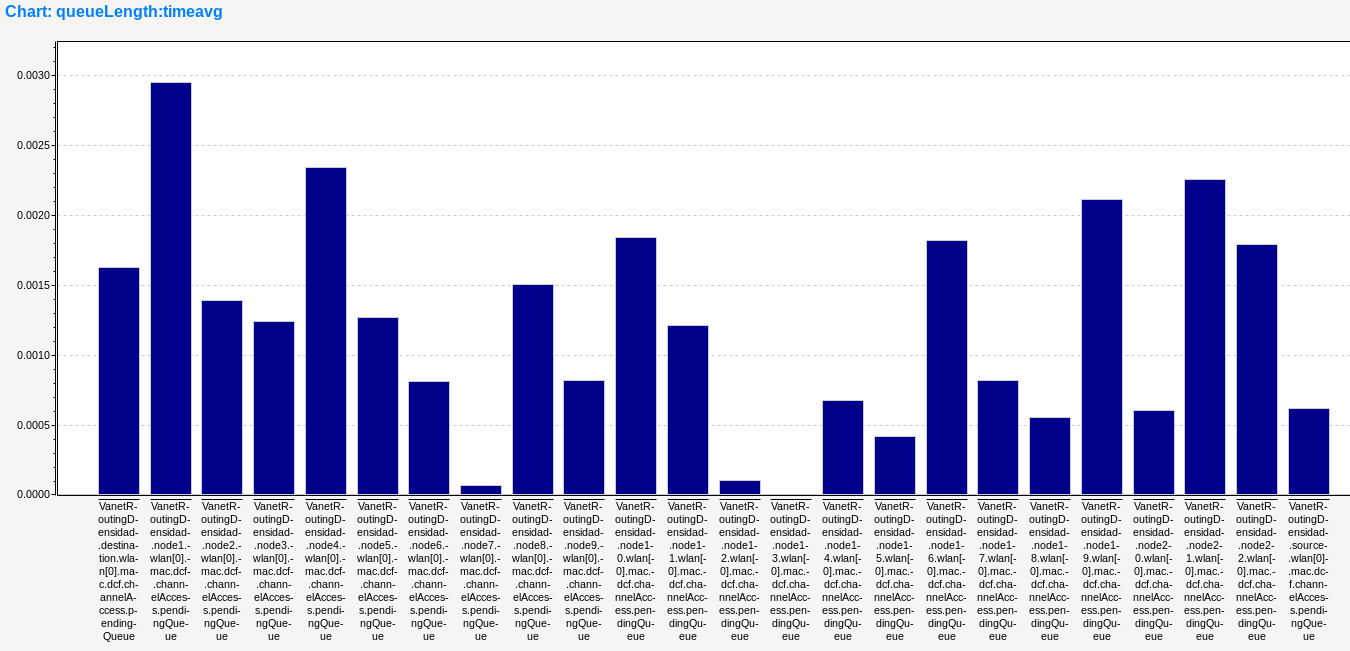
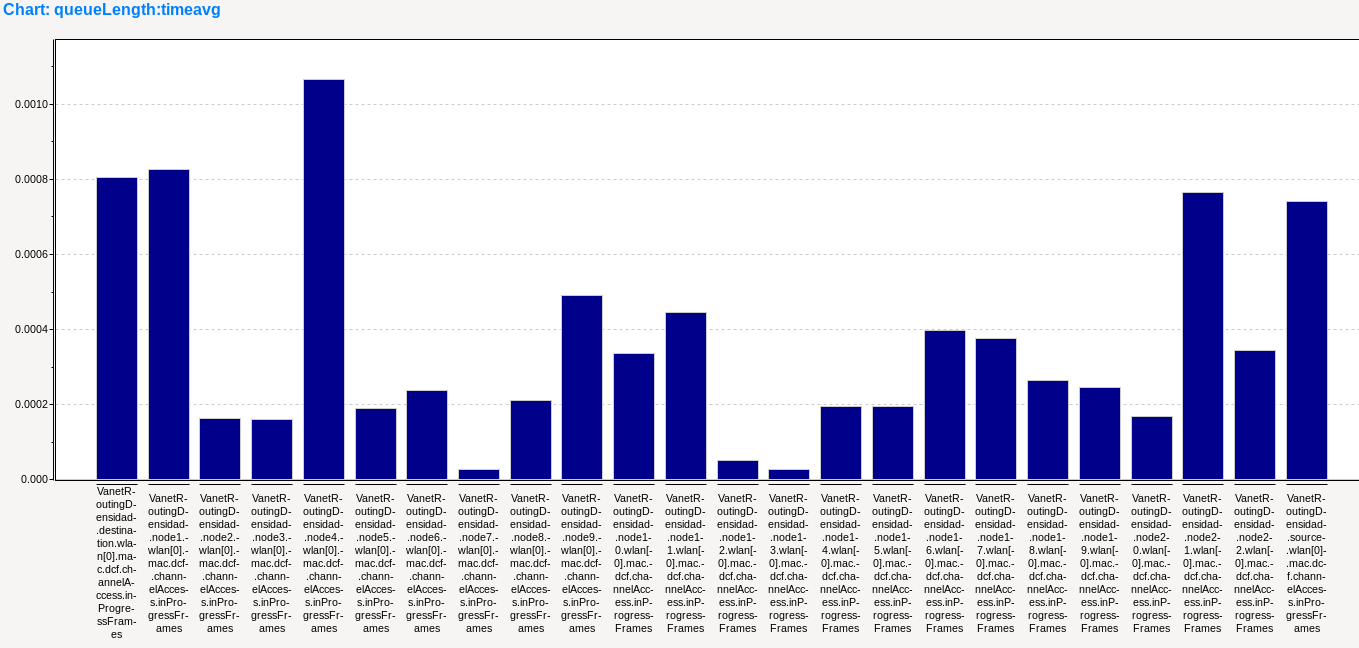
Gráfico, Histograma

Descripción generada automáticamenteGráfico, Gráfico de barras

Descripción generada automáticamente

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 0.5 s | 11 | 9 | 18.18% | 41.4 ms |
| Como con 50 km/h los cambios en la topología requiere de actualizar las rutas e inundar la red, aumenta el RTT y algunos mensajes se pierden.  Gráfico, Gráfico de barras  Descripción generada automáticamenteGráfico, Gráfico de barras, Histograma  Descripción generada automáticamente | | | | | |





|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 0.2 s | 27 | 27 | 0% | 2.35 ms |
| Logra transmitir los mensajes sin problema, a diferencia con 50 km/h se requieren más mensajes para mantener las rutas. | | | | | |

Gráfico, Histograma

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente con confianza media

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 0.1 s | 55 | 55 | 0% | 2.17 ms |

Los resultados nos muestran que conviene usar un intervalo adaptativo entre el envío de mensajes, según la densidad y velocidad del tráfico, para casos con baja densidad de nodos, no es posible mantener bajos intervalos entre mensajes. Mientras que, si bien puede mantener bajos intervalos a altas velocidades, se observa el aumento en el RTT y de mensajes requeridos para mantener las rutas, por lo que, al sumar más nodos transmitiendo y comunicándose, habrá un aumento en las colisiones y tiempos de encolamiento, por lo que deberá reducirse la frecuencia con la que actualizan los nodos. Además, falta considerar los tiempos de procesamiento de los mensajes.

Simulación 2

Para la segunda simulación consideramos el caso en el que todos los nodos intentan comunicarse entre sí (se compone de 9 nodos, 7 transmiten sus detecciones y todos retransmiten), con el objetivo de informar de las detecciones, los nodos al recibir un mensaje deben retransmitirlo con cierta probabilidad una vez. 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 únicamente se especifican los archivos de configuración 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.

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 7-9 a la 7-13.

Imagen que contiene Esquemático

Descripción generada automáticamente

Figura ‑ El nodo 6 en rojo quiere transmitir

Imagen que contiene Diagrama

Descripción generada automáticamente

Figura ‑ 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 retransmisiones quedan informados.

Esquemático

Descripción generada automáticamente con confianza media

Figura ‑ 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

Imagen que contiene Esquemático

Descripción generada automáticamente

Figura ‑ Posición final, el nodo 6 se incorpora

Para experimentar con la velocidad, y ver qué tantos mensajes pueden recibir al recorrer una misma distancia, como se observa en la figura 8-19, los nodos parten de una posición inicial a una final.

Cada nodo intenta transmitir de forma aleatoria dentro de un intervalo de 100 ms, de forma que podamos observar algunas colisiones, durante la simulación los nodos 3, 8 y 6 (abajo en la figura 8-15) no son alcanzados por quien transmite el mensaje original, pero si con las retransmisiones.

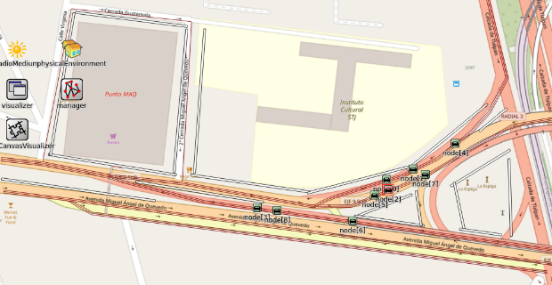
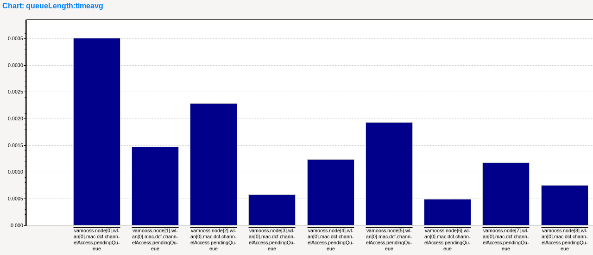
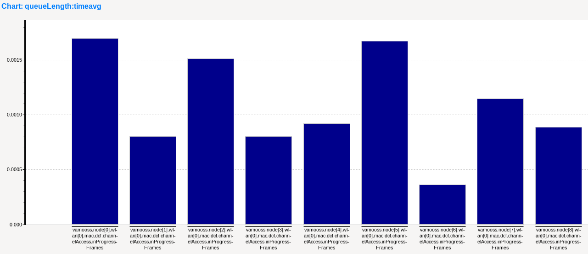


Figura ‑ A la izquierda se observa la posición de partida de los nodos, a la derecha la posición final

Como en la simulación 1, partimos del primer escenario para comparar, en este, todos transmiten sus detecciones cada segundo, a excepción de los nodos 1 y 6, se mueven a 30 km/h, en una simulación que dura 40 segundos, lo que corresponde a que cada nodo intentará transmitir 40 veces. En las gráficas se observan los mensajes enviados y recibidos por cada nodo, estos toman en cuenta tanto sus transmisiones como retransmisiones, el caso de un intervalo entre transmisiones de 0.1 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, y considerando que la red apenas está compuesta de 9 nodos, se esperaría que con el aumento del número de participantes intervalos para comunicarse como de 0.2 segundos también presenten problemas, que nos muestra la falta de un nodo central que coordine a los demás nodos para transmitir.

Los nodos 1 y 6 no transmiten, no se cuentan en el promedio de mensajes enviados.

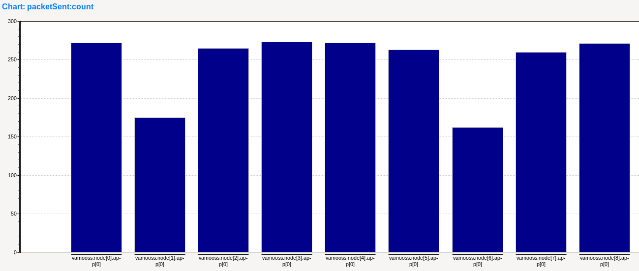
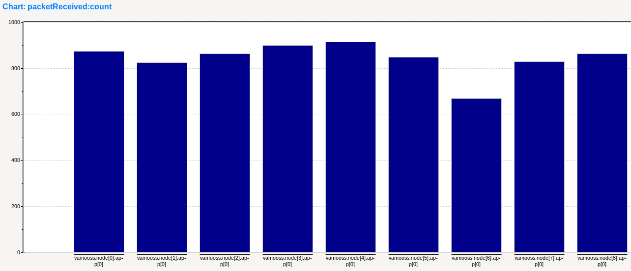
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Velocidad | Intervalo de transmisión | Duración de la simulación | Detecciones transmitidas | Mensajes enviados por nodo en promedio | Mensajes recibidos promedio |
| 30 km/h | 1s | 40 s | 40 | 169 | 520 |



Como en la simulación 1, los tiempos están divididos en “pending queue” e “in progress frames”, estando en promedio para acceder al medio en 1.5 ms, y para transmitir 1 ms.

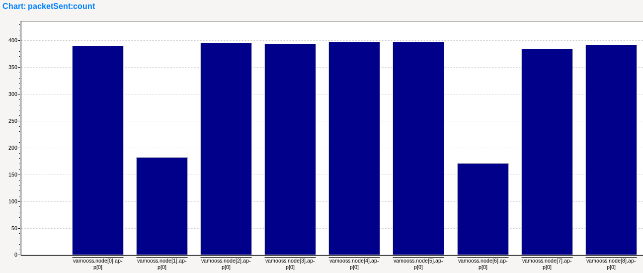
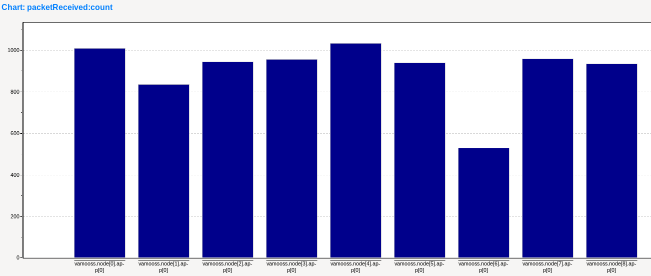
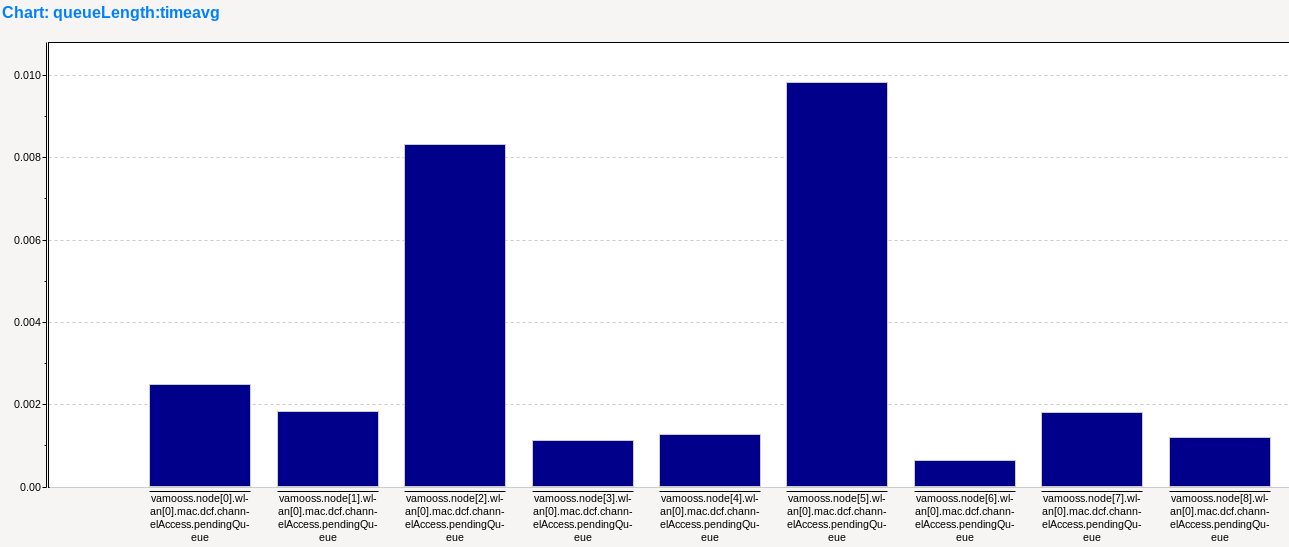
Gráfico, Gráfico de barras

Descripción generada automáticamente

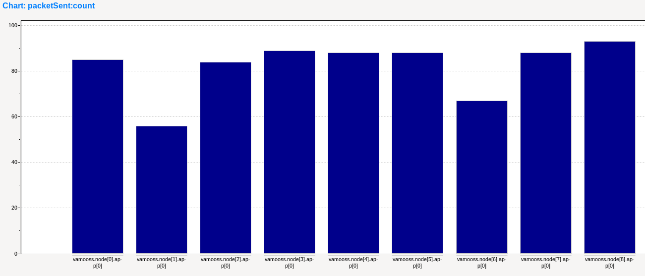
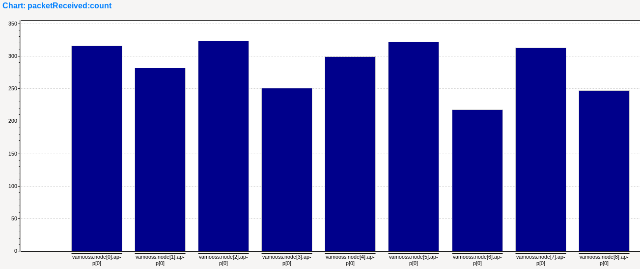


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 km/h | 0.5 s | 40 s | 80 | 264.2 | 840 |
| No se observan grandes cambios en la tasa de errores, ni tiempos de encolamiento. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 km/h | 0.2 s | 40 s | 200 | 388.2 | 920 |
| Como se observa en la baja cantidad de mensajes transmitidos por nodos, comparado a las detecciones que se requieren transmitir, en la red se pierden bastantes mensajes por colisiones, además, los tiempos de procesamiento en algunos casos se duplican. | | | | | |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 1 s | 24 s | 24 | 88.1 | 287 |
| Al aumentar la velocidad, los tiempos y la tasa de errores se mantiene, a diferencia de la simulación 1, aquí los nodos son alcanzables en todo momento. | | | | | |



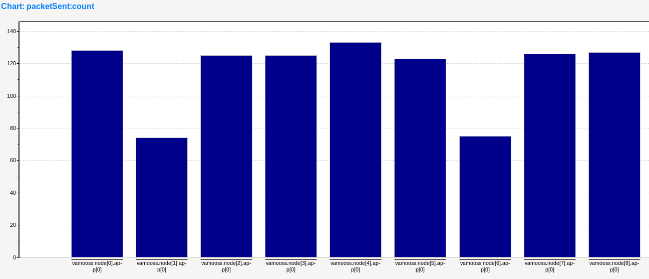
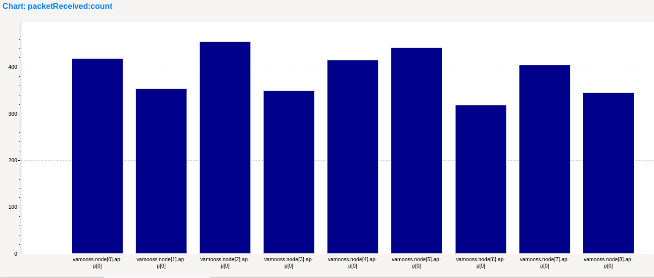
Gráfico, Gráfico de barras

Descripción generada automáticamente

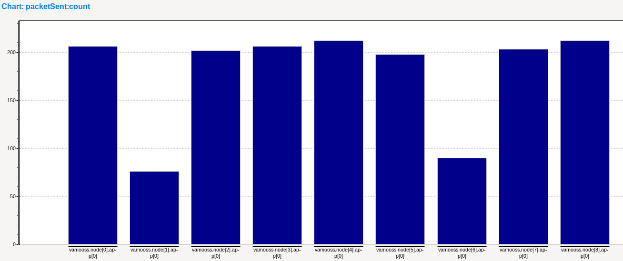
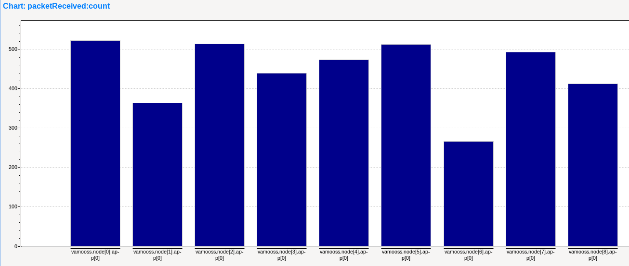
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 0.5 s | 24 s | 48 | 126.8 | 387 |

Gráfico, Gráfico de barras

Descripción generada automáticamente

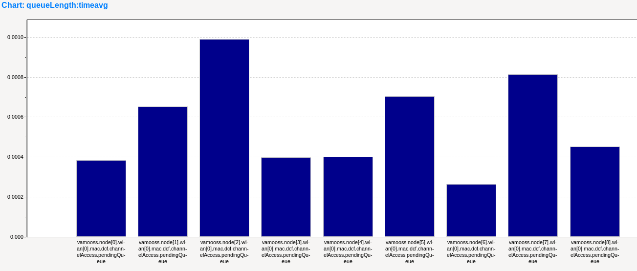


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 50 km/h | 0.2 s | 24 s | 120 | 216 | 447 |
| Igual que a 30 km/h, los tiempos no solo de acceso al medio, también de procesamiento aumentan, y de forma más considerables aquí, igualmente la tasa de errores presenta un cambio importante. | | | | | |



Gráfico, Gráfico de barras

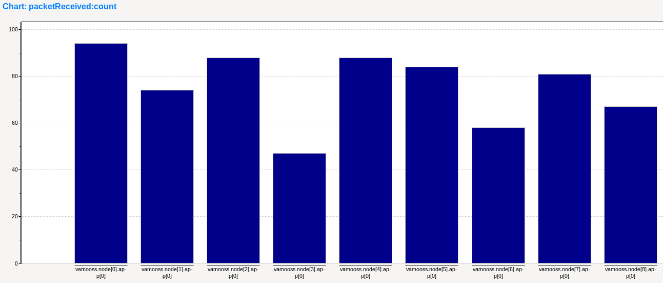
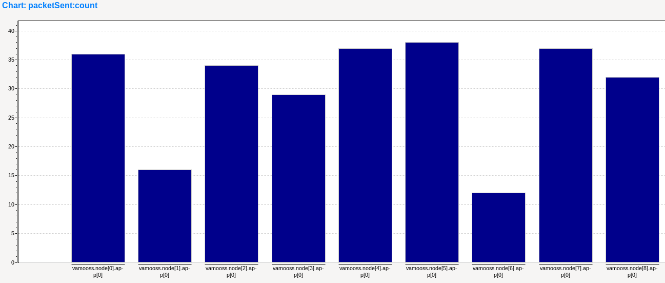
Descripción generada automáticamente



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 1 s | 15 s | 15 | 34.5 | 76 |
| Se observa el cambio considerable en el número de mensajes recibidos por los nodos a mayor velocidad, muchas transmisiones terminan perdiéndose. | | | | | |

Gráfico, Gráfico de barras

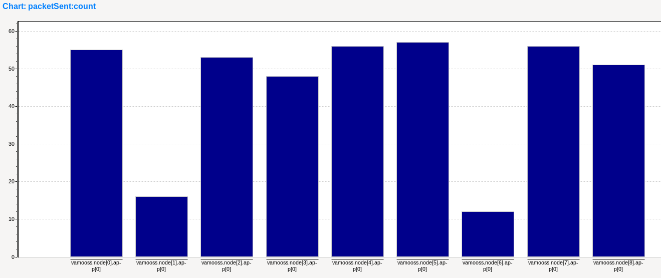
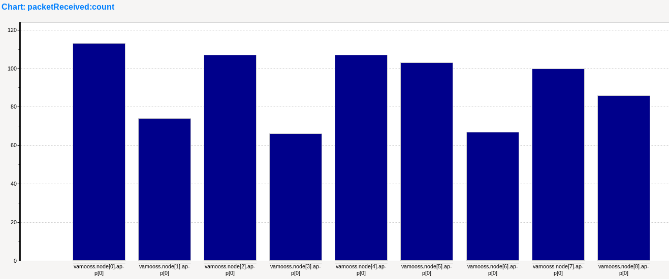
Descripción generada automáticamente



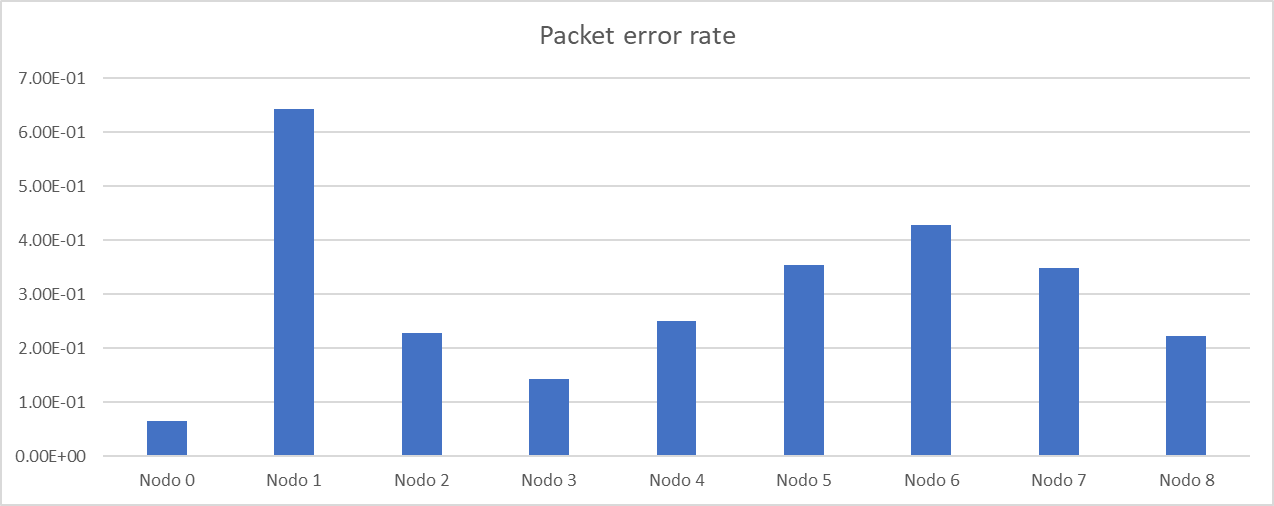
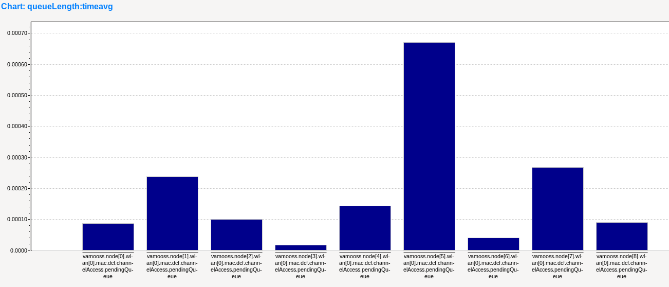
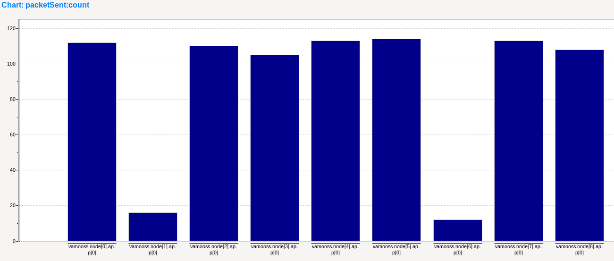
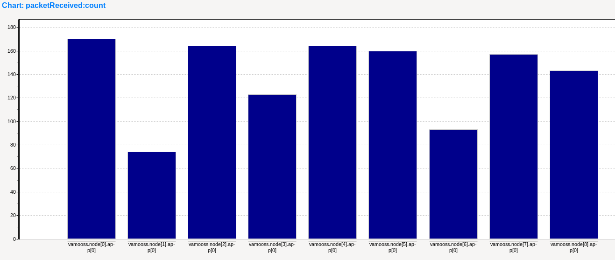
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 0.5 s | 15 s | 30 | 53.8 | 91.7 |
| Se mantiene el caso anterior, a altas velocidades se pierden muchos mensajes y la tasa de error es alta. | | | | | |

Gráfico, Gráfico de barras

Descripción generada automáticamente



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 80 km/h | 0.2 s | 15 s | 75 | 112.8 | 139 |
| El problema se acentúa aún más, los nodos apenas transmiten 113 mensajes en promedio, de los que 75 son suyos. En el caso de los tiempos de encolamiento, como es de esperarse al recibir pocos paquetes, disminuye. | | | | | |



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.

# Conclusiones

## Resultados

Precisiones, tiempos, frecuencia

## Discusión y mejoras

# 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>. [18]

Andreas Geiger, Philip Lenz, Christoph Stiller, y Raquel 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>. [12]

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 [13]

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>.

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. [4]

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%20codificaci%C3%B3n%20y%20modulaciones%20avanzadas_M%C3%B3dulo%205_Multiplexaci%C3%B3n%20por%20divisi%C3%B3n%20en%20frecuencias%20ortogonales%28OFDM%29.pdf>.

«catkin/workspaces - ROS Wiki». s. f. Accedido 12 de abril de 2023. <http://wiki.ros.org/catkin/workspaces>. [16]

«es - ROS Wiki». s. f. Accedido 12 de abril de 2023. <http://wiki.ros.org/es>. [9]

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>. [1]

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. [5]

Fernández Villaverde, Jesús. s. f. «Kalman and Particle Filtering». University of Pennsylvania. <https://www.sas.upenn.edu/~jesusfv/filters_format.pdf>.

Hartenstein, Hannes, y Kenneth P Laberteaux, eds. 2010. *VANET: Vehicular Applications and Inter‐Networking Technologies*. 1.a ed. Wiley. <https://doi.org/10.1002/9780470740637>.

«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/documentodereferenciaidentificaciondenecesidadesstien59ghz_0.pdf>.

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>. [8]

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. [2](http://arxiv.org/abs/1704.05519.%20%5b2)] [6] [B]

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>.

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>.

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>. [3]

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.[14]

«OpenCV: Cascade Classifier». s. f. Accedido 29 de mayo de 2023. <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>. [10]

«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>.

«OpenCV: Smoothing Images». s. f. Accedido 31 de mayo de 2023. <https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html>.

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. [17](https://pointclouds.org/documentation/structpcl_1_1_point_x_y_z_r_g_b_a.html.%20%5b17)]

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>.

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>. [19]

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.gltp.2021.08.044>.

«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/>. [15]

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.[7](https://doi.org/10.1016/j.adhoc.2022.102978.%5b7)]

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

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

Zhou, Zhi-Qiang. (2019) 2023. «HOG Visualization Using OpenCV(C++)». C++. <https://github.com/zhouzq-thu/HOGImage>. [11]