Jornadas Nacionales de Robótica
Spanish Robotics Conference
14-15 Junio 2018 http://jnr2018.eii.uva.es

Desarrollo de un vehículo eléctrico autónomo de código abierto para personas mayores

Luís M. Bergasa[a], R. Sanz[b*], J. López[b], E. Paz[b], C. Otero[b], P. Sánchez[b], R. Barea[a], E. López-Guillén[a], P. Revenga[a], E. Molinos[a], E. Romera[a]

[a] *Departamento de Electrónica, Universidad de Alcalá, Alcalá de Henares, España.*
[b] *Departamento de Ingeniería de Sistemas y Automática, Universidad de Vigo, Campus Universitario Lagoas-Marcosende, 36310, Vigo, España.*

Resumen

Este artículo presenta algunos de los principales resultados de un proyecto de investigación en tecnologías para el desarrollo de un coche eléctrico autónomo que asista al sector de población de personas mayores en entornos fundamentalmente urbanos. La arquitectura del software empleada se basa en el sistema operativo ROS (*Robot Operating System*). Se describe nuestra propuesta de arquitectura ROS para la percepción del entorno, la navegación y la planificación en un campus universitario. Para evaluar el sistema, se han llevado a cabo diferentes pruebas en un simulador basado en V-REP que incluyen escenarios con diferentes comportamientos, tales como: mantenimiento de carriles, intersección de carriles y parada en un cruce de peatones. La arquitectura desarrollada se está aplicando a un coche eléctrico de código abierto orientado a un sector de la población con necesidades crecientes.

Palabras Clave: Conducción Autónoma, Lidar 3D, Cámaras, Fusión Sensorial, Mapeado 3D, Control y Planificación.
Datos del Proyecto:
Denominación del proyecto: Vehículo inteligente para personas mayores
Referencia: TRA2015-70501-C2-1-R, TRA2015-70501-C2-2-R
Investigador/es responsable/es: Luís Miguel Bergasa Pascual (UAH) y Rafael Sanz Domínguez (UVigo)
Tipo de proyecto (internacional, nacional, autonómico, transferencia): Nacional
Entidad/es financiadora/s: Mineco y fondos FEDER
Fecha de inicio/fin: 1 enero 2016/ 31 diciembre 2018

Introducción

La conducción autónoma es considerada como una de las soluciones a los problemas planteados y uno de los grandes retos de la industria de la automoción en nuestros días. La existencia de vehículos autónomos fiables y asequibles económicamente creará un gran impacto en la sociedad afectando a aspectos medioambientales, demográficos, sociales y económicos. En particular, se estima que provoque una reducción de las muertes en carretera, una mejora en el flujo de tráfico, una reducción del consumo de combustible y emisiones nocivas asociadas, así como una mejora en el confort del conductor en general, y en la movilidad de colectivos con facultades deterioradas, como pueden ser las personas mayores o los discapacitados.

La conducción autónoma ha atraído gran atención recientemente en los grupos de investigación y en la industria, debido a los espectaculares anuncios de varias empresas sobre previsiones de entrada al mercado. Sin embargo, sus predicciones interesadas parecen muy optimistas. Un organismo más científico, como es IEEE, ha predicho recientemente que para el año 2040 la mayoría de vehículos que circulen por autopistas serán autónomos. La conducción en entornos urbanos tardará más en llegar, debido a su mayor complejidad e incertidumbre.

Este artículo presenta los principales resultados de un proyecto de investigación en técnicas autónomas de navegación en entornos urbanos llevados a cabo para implementarlas en un vehículo eléctrico de código abierto. Además, se describen los métodos de percepción, navegación y planificación, basados en GPS, cámaras y LIDAR, y llevados a cabo sobre una arquitectura basada en ROS. Esta arquitectura permite la coordinación de controladores y middleware, lo que simplifica la compleja tarea de adquisición de datos globales y la sincronización de sensores. El sistema propuesto permite la comunicación entre procesos de forma independiente y modular, permitiendo ejecutar algoritmos múltiples y paralelos para alcanzar objetivos de bajo nivel (adquisición de datos de sensores y control de actuadores) y objetivos de alto nivel (toma de decisiones, planificación de trayectorias y navegación). Finalmente, presentamos la etapa de simulación llevada a cabo para validar la propuesta de arquitectura y algunos resultados experimentales de diferentes conductas de conducción en simulación.

1. Arquitectura software

El sistema desarrollado, que hemos denominado *SmartCar* (figura 1), es una arquitectura jerarquizada y modular, en la que los módulos individuales procesan la información de forma asíncrona. Estos módulos son procesos independientes que se comunican entre sí utilizando el sistema de comunicación entre procesos ROS. En particular, el paradigma de publicación/suscripción se utiliza para proporcionar comunicaciones no bloqueantes.

Cada módulo corresponde a un proceso de Linux independiente que se ejecuta en el ordenador de a bordo. Los diferentes módulos están organizados en cuatro conjuntos:

- La *capa hardware/simulación* es la de nivel más bajo. Se trata del vehículo real, en el caso de la capa de hardware, o bien del modelo simulado, en el caso de la capa de simulación.
- La *capa de control de dispositivos hardware* incluye un conjunto de programas que controlan diferentes dispositivos de hardware, entre ellos los sensores y actuadores.
- La *capa de control* implementa la funcionalidad básica de control y navegación. Incluye el control reactivo (*local navigator*), la localización (*localization*), la planificación de rutas (*Map manager*) y un programa que procesa la mayoría de los sensores para detectar eventos relevantes (*Event monitor*).
- La *capa ejecutiva* coordina la secuencia de acciones que necesitan ser ejecutadas por otros módulos para llevar a cabo el comportamiento actual.
- La *capa de interfaz* consiste en un conjunto de procesos para interactuar con los usuarios y conectarse a otros procesos para aplicaciones multi-robot.

Algunos módulos de navegación incluidos en el control reactivo son críticos para la seguridad de la aplicación. Estos módulos importantes se encuentran principalmente en las capas inferiores: desde la capa de hardware, los actuadores que controlan el automóvil y los sensores que detectan los obstáculos; desde la capa de controladores de hardware, los controladores que controlan esos dispositivos y desde la capa de control, el navegador local. A diferencia de otras arquitecturas como (Urmson *et al.*, 2008 y Montemerlo *et al.*, 2003) que incluyen la planificación en el control de movimiento, en la solución propuesta aquí, el control de movimientos se divide en planificación de alto nivel y control reactivo de bajo nivel. La planificación de alto nivel produce en el primer término una ruta que consiste en una secuencia de *lanelets* (Bender *et al.*, 2014). El objetivo del sistema de navegación local presentado aquí es seguir el camino de forma segura, manteniendo el automóvil dentro del carril de conducción.

Los módulos de control integran la información de los sensores y del movimiento para proporcionar una mejor odometría de los sensores, capacidades básicas de navegación (localización, planificación de ruta y seguimiento de trayectorias) y funciones específicas básicas (aparcamiento, etc.).

El gestor de mapas carga el mapa y se encarga de planificar una nueva ruta como una secuencia de carriles. Cada carril está definido por sus bordes llamados *ways* (caminos) en Urmson *et al.* (2008). Además de la ruta, este módulo atiende consultas de otros módulos relacionados con el mapa. Por ejemplo, debe proporcionar los carriles contiguos para la maniobra de adelantamiento, los carriles de una intersección con el monitor de eventos para la maniobra de intersecciones, y también debe proporcionar la posición de los elementos reguladores.

El entorno en esta aplicación es dinámico y el vehículo puede encontrar obstáculos estáticos (un automóvil, un árbol caído, etc.) y obstáculos en movimiento como personas y otros vehículos. Si el obstáculo estático no bloquea completamente el carril, el módulo de navegación local lo evitará. Si el obstáculo bloquea el carril, la capa ejecutiva podría decidir adelantarlo si es posible.

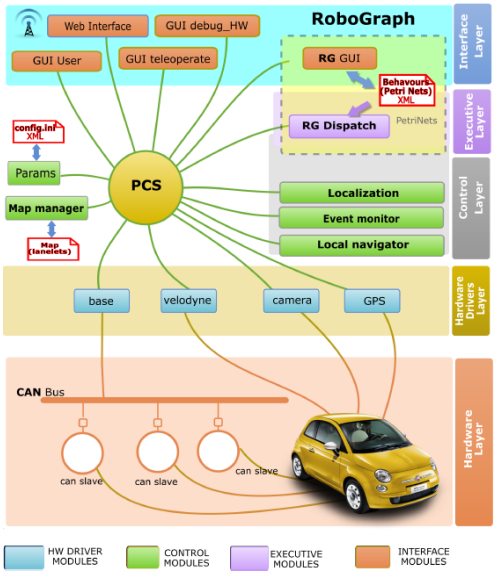

Figura 1: Arquitectura software del proyecto.

2. Capas de interfaz y ejecutivo

Una interfaz principal de usuario (*GUI user*) le permite al usuario seleccionar el destino y ver la ruta en una pantalla como un navegador de automóviles. Además, un módulo de interfaz a bordo permite a los usuarios interactuar directamente con el vehículo para probar algunos actuadores a nivel hardware y verificar las lecturas de los sensores. También hay varios módulos de interfaz que el programador puede usar para depurar y rastrear los programas de control. Un módulo de interfaz web se puede utilizar para conectar el vehículo de forma remota. Finalmente, una interfaz gráfica de usuario externa permite conectar y operar el vehículo de forma remota utilizando esta interfaz.

La capa ejecutiva (*Executive layer*) implementa diferentes comportamientos necesarios para conducir de forma autónoma el vehículo. En el contexto de este proyecto, un comportamiento corresponde a una situación de tráfico, tal como un semáforo, un paso de peatones, una intersección con ceda el paso, una intersección con un stop, incorporación a una autopista, etc. Cada uno de estos comportamientos se implementa en una o más redes de Petri. De alguna manera, las redes de Petri que implementan un comportamiento definen las reglas de tráfico para manejar la situación del tráfico correspondiente. El sistema resultante es muy flexible, ya que una adaptación a una norma de tráfico se implementa fácilmente con cambios menores en las redes de Petri correspondientes.

El papel de la capa ejecutiva es coordinar y sincronizar otros módulos para llevar a cabo la conducción autónoma en las diferentes situaciones. Cada comportamiento se modela como una secuencia de eventos discretos usando las redes de Petri. Mientras se ejecuta, otros módulos comunican eventos a esta capa (cruce de peatones, próximo a una intersección, etc.) a través de mensajes. De acuerdo con la receptividad de la red de Petri, en cada situación (comportamiento) el módulo ejecutivo está suscrito a los mensajes que pueden hacer evolucionar el estado de la red de Petri. Por otro lado, el módulo ejecutivo puede solicitar la ejecución de algunos comandos (reducir la velocidad, buscar peatones, etc.), publicando mensajes para otros módulos que pueden llevar a cabo estos comandos.

3. Robograph

La secuencia de acciones y eventos que forman parte de un comportamiento se implementa mediante redes de Petri usando una herramienta de la Universidad de Vigo llamada *RoboGraph* (Fernández *et al.*, 2008). Esta herramienta ha sido utilizada en otros proyectos con robots móviles en aplicaciones que usan IPC y JIPC (López *et al.*, 2011) y que se ha ampliado para funcionar dentro de ROS.

La Figura 1 muestra los programas que forman RoboGraph. *RG GUI* (*Robograph Graphical User Interface*) es una herramienta de desarrollo que permite crear, editar y monitorizar la ejecución de las diferentes tareas, mientras que *RG Dispatch* está a cargo de ejecutar esas tareas.

El programa *RG GUI* puede funcionar en tres modos diferentes, siendo posible cambiar en cualquier momento de uno a otro: *Editor*, *Monitor* y *Play Logger*. En modo editor, el usuario puede crear nuevas tareas utilizando un editor gráfico de redes de Petri simple e intuitivo. Una vez que la red de Petri ha sido construida, seleccionando y arrastrando diferentes elementos (lugares, transiciones, arcos y marcas), las acciones, asociadas a los lugares y las transiciones, y las condiciones, asociadas a las transiciones, deben definirse para la red de Petri interpretada.

Las acciones pueden ser comandos implementados en cualquier módulo en la arquitectura de control de la figura 1. Estos comandos se pueden seleccionar de una lista generada automáticamente por la GUI (figura 2). Cada comando es un mensaje ROS y el usuario debe definir los parámetros del mensaje que aparecerán automáticamente en una nueva ventana cuando ese comando se seleccione en el editor.

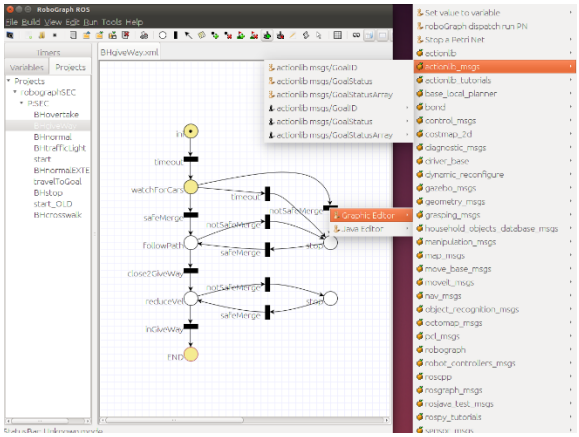

Figura 2: Edición del comportamiento *BHgiveWay* con Robograph.

Cuando *RG Dispatch* ejecuta la red de Petri, los mensajes de ROS asignados a lugares y transiciones se publicarán a medida que la red evoluciona.

4. Percepción del entorno

La percepción del entorno se basa en la fusión de la nube de puntos obtenida con un LIDAR 3D y la segmentación semántica calculada desde una cámara. La segmentación semántica se lleva a cabo en las imágenes RGB tomadas para la cámara derecha del par de cámaras estéreo para detectar, de forma unificada, el área de conducción y los diferentes obstáculos en el campo de visión de la cámara. Para hacer eso, utilizamos una Red Neural Convolucional (CNN), llamada *ERFNet* (ConvNet Factorizado Residual Eficiente para la Segmentación Semántica en Tiempo Real) (Romera *et al.*, 2018) desarrollada por el grupo de la Universidad de Alcalá. Después de eso, los píxeles segmentados en 2D se fusionan con la nube de puntos 3D proporcionados por el sensor LIDAR. De esta forma, se obtiene una nube de puntos 3D coloreada, en el campo de visión de la cámara, donde cada color representa una clase de obstáculo (vehículos, peatones, árboles, etc.). La nube de puntos fuera de este campo de visión se mantiene sin segmentar. Los obstáculos que pueden interferir con la navegación del vehículo se obtienen a partir la nube de puntos usando la agrupación de colores. La velocidad y la dirección de cada obstáculo se obtienen utilizando un evaluador de *Precision Tracker* (Held *et al.*, 2013). Esta información es publicada por el módulo de monitorización de eventos.

5. Planificación de trayectorias

La generación de mapas se ha realizado utilizando el enfoque *lanelet* presentado en Bender *et al.* (2014). Usando la herramienta JOSM de código abierto, se ha obtenido un mapa del campus externo de la Universidad de Alcalá, dado que las pruebas reales con el prototipo de vehículo se están llevando a cabo en dicho campus. Los carriles y las conexiones entre ellos se han delimitado manualmente, incluida la información de elementos regulatorios de tráfico (p. ej. puntos de parada, intersecciones, rotondas, etc.), para generar un mapa enriquecido útil para la navegación. La estructura generada ha sido procesada por un nodo ROS denominado *Map Manager*, que permite planificar una nueva trayectoria como una secuencia de *lanelets*. Se crea una estructura de datos que incluye la ruta más corta entre dos puntos cualesquiera del mapa, y se la envía al planificador de trayectorias. Además de la trayectoria, este módulo atiende consultas de otros módulos relacionados con el mapa. Por ejemplo, proporciona los carriles contiguos para la maniobra de adelantamiento, los carriles de una intersección con el monitor de eventos para la maniobra de intersección cruzada y también proporciona la posición de los elementos reguladores. Para la navegación, se aplican dos planificadores diferentes. En primer lugar, se obtiene una ruta de *lanelets* utilizando un algoritmo A* de los mapas de *lanelets*. Posteriormente, el sistema de navegación local obtiene primero la curvatura para guiar al robot desde la posición actual hacia una posición de avance en el centro del carril usando el enfoque de búsqueda pura (Coulter, 1992). Esta curvatura se usa como referencia para un método de evitación de obstáculos basado en el método BCM (*Beam Curvature Method*) (Fernández *et al.*, 2004.

6. Navegación local

El módulo de navegación local debe garantizar las operaciones de seguridad del vehículo independientemente de la decisión tomada por la capa ejecutiva. Así, cuando se detecta un obstáculo el sistema de control en tiempo real tiene que evitarlo y, por lo tanto, tiene que tener una respuesta muy rápida. La navegación local está basada en dos métodos simples y rápidos (*Pure pursuit* y BCM) que se han implementado de manera eficiente en ordenadores con bajas capacidades de proceso.

La Figura 3 muestra los componentes del módulo de navegación local. Las entradas y salidas del módulo corresponden a mensajes con otros módulos de la arquitectura del software (figura 1). Cada componente del módulo de navegación local (cuadros verdes en la figura 2) tiene sus propias entradas y salidas. En las siguientes subsecciones, describimos cada uno de estos componentes.

La ruta es proporcionada en un mensaje enviado por el módulo gestor de mapas como una secuencia de *lanelets* a seguir. La Figura 4 muestra un ejemplo de *lanelets* en la intersección de una rotonda (líneas rojas). El gestor de rutas procesa esta información y extrae una lista de obstáculos virtuales creados a partir de las líneas limitadoras de los carriles. El objetivo de estos obstáculos virtuales es garantizar que el módulo reactivo no obligue a cruzar las líneas limitadoras y que el vehículo se mantenga en medio del carril de circulación. La trayectoria es la línea media del *lanelet* que el vehículo debería tratar de seguir.

Para la gestión de trayectorias se ha empleado el algoritmo de seguimiento de trayectoria denominado *Pure Pursuit* (Coulter, 1992). Este algoritmo calcula la curvatura que conduce al vehículo a un punto elegido en la ruta que se encuentra a una determinada distancia de anticipación desde la posición actual del vehículo. Esta curvatura es enviada al módulo reactivo.

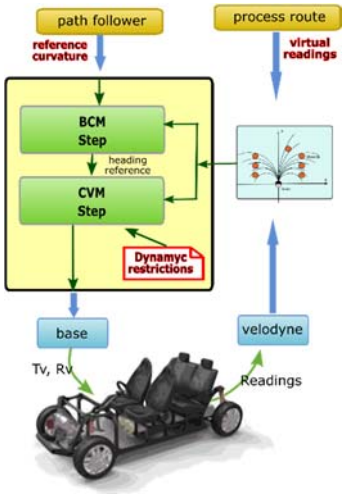

Figura 3: Componentes del módulo de navegación local.

El módulo de evitación de obstáculos es responsable del bucle de control que evita los obstáculos mientras sigue las referencias dadas por *follow path*. El método utilizado para evitar los obstáculos es una adaptación del método BCM. Las entradas al lazo de control son la referencia de curvatura y las lecturas de los sensores. Las lecturas de los sensores integran las lecturas del sensor virtual, obtenidas por el componente de ruta del proceso, y las lecturas del sensor *Velodyne*. Las salidas de este componente son las velocidades de translación y de rotación que se envían en un mensaje ROS al módulo base para obtener la velocidad y la aceleración de vehículo de acuerdo con el modelo de Ackerman.

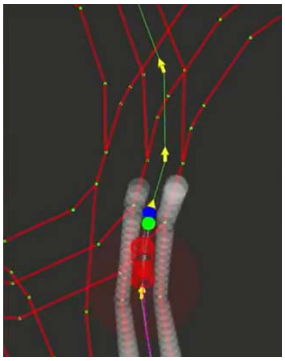

Figura 4: Intersección en una rotonda, mostrando los obstáculos de la frontera (círculos blancos) y trayectoria a seguir (línea verde con flechas amarillas).

7. Simulación del vehículo

Para la simulación del comportamiento del vehículo con los módulos desarrollados en el proyecto se ha utilizado el simulador multiplataforma V-REP, desarrollado por Coppelia Robotics GmbH. V-REP fue elegido por su facilidad para crear nuevos modelos o modificar los ya existentes. Aunque no soporta nativamente la comunicación con nodos ROS, el *plugin RosInterface* permite esta funcionalidad, replicando de manera natural la API de la librería *roscpp* de ROS.

Este simulador permite la personalización completa de la simulación mediante diferentes enfoques (complementos, *plugins*, *scripts*, etc.) y soporta cuatro motores dinámicos o físicos. Además, es posible crear modelos personalizados utilizando una amplia variedad de actuadores y sensores incorporados, articulaciones, formas y mallas, *scripts*, etc.

Se ha modelado el entorno del campus externo de la UAH, donde se probará el vehículo real, utilizando el servicio abierto *OpenStreetMap*. En este servicio los usuarios pueden mapear una zona y subirla a los servidores para uso público. Para importar los *lanelets* en V-REP se ha desarrollado un programa que convierte los carriles en mallas. V-REP permite importar fácilmente diversos formatos de mallas, por lo que las carreteras y los edificios se han cargado mediante archivos OBJ. Para importar los edificios se ha descargado el archivo OSM correspondiente al campus de la UAH, directamente de la web de *OpenStreetMap*. Posteriormente, este archivo se ha convertido a formato OBJ con la herramienta de código abierto *OSM2World*. Para realizar pruebas de percepción con los sensores de visión, se han añadido manualmente elementos comunes encontrados habitualmente en el campus de la UAH (señales, árboles, farolas, etc.). Estos elementos han sido obtenidos de repositorios de archivos CAD libres y convertidos a formato OBJ.

La parte visual del vehículo ha sido importada de un repositorio de CAD, dándole el aspecto de un vehículo comercial. La parte con propiedades dinámicas está formada por *cuboids* (paralelepípedos rectos) a los que se les ha dado una masa para simular la inercia del vehículo. Además, está dotado

de cuatro ruedas con amortiguación, motor propulsor, frenos, dirección y sensores. Para simular el sistema de dirección del vehículo real con aceleraciones finitas, éste se ha simulado en V-REP como un sistema dinámico (con inercia y fricción) actuado por un motor servo-controlado en posición. Esto quiere decir que la posición del volante es controlada internamente por V-REP mediante un PID. El simulador recibe consignas de posición del volante de la misma manera que el vehículo real.

8. Descripción del vehículo eléctrico

Para las pruebas reales se está utilizando un vehículo de código abierto basado en el chasis del modelo TABBY EVO de la compañía Open Motors (Tabby Evo, 2018). Este chasis ha sido equipado con un paquete de baterías, una carrocería tubular, algunos sensores (GPS, LIDAR y cámaras) (Figura 5) y ha sido modificado para lograr una navegación autónoma.

Se ha retirado el volante manual para instalar una dirección asistida eléctrica comercial (modelo Opel Corsa) con un codificador, implementado en la UAH, para controlar electrónicamente la dirección del vehículo, como se muestra en la figura 5. Para hacer eso, diseñamos una ECU basada en una placa de desarrollo de código abierto olimexino-STM32, que recibe los comandos de ángulo y genera una señal PWM a un puente completo mediante el uso de un control de circuito cerrado PID. En el proceso de configuración, se ha calculado la relación entre el ángulo de las ruedas y el ángulo deseado. También se lee un sensor de par incluido en la dirección asistida, para cambiar a control manual cuando el conductor pone su mano en el volante. Además, la señal generada por el acelerador se conmuta mediante una señal generada por la ECU para obtener la aceleración deseada. Si el conductor pone su pie en el pedal, el sistema cambia automáticamente al modo manual. Las entradas de la ECU son la velocidad y el ángulo deseados de las ruedas, y estos comandos se envían a través del bus CAN desde el control de alto nivel. El frenado mecánico no se ha automatizado porque se realiza actualmente utilizando el frenado regenerativo del motor.

Figura 5: Chasis del vehículo y detalle de la dirección electrónica del volante.

Para la percepción del entorno, el vehículo ha sido equipado con un LIDAR *Velodyne* (VLP-16) colocado en la parte superior del vehículo y en el centro del mismo. El dispositivo proporciona 16 canales de FOV horizontal de 360 ° y FOV vertical de + - 15 °. Además, el vehículo ha sido equipado con una cámara en color de visión estereoscópica de 3 sensores y multi-línea base (*Bumblebee* XB3 1394b). La cámara tiene una resolución máxima de 1280x960 píxeles a 16 fotogramas por segundo. Se ha montado en el parabrisas delantero del vehículo a 160 cm de altura sobre el suelo y orientado a la carretera. Finalmente, se ha instalado un DGPS-RTK GPS *TopCon HiperPro* en la parte superior central del vehículo. Todos estos dispositivos están conectados a un ordenador embebido, que funciona bajo el sistema operativo Linux, donde se ejecuta la arquitectura de software.

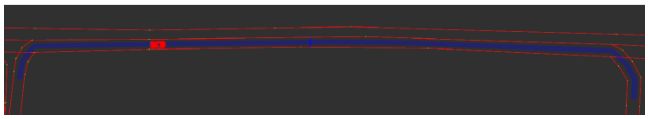

Figura 6: Trayectoria seguida por el vehículo.

9. Algunos resultados de simulación

Para validar nuestra propuesta de arquitectura software basada en ROS se presentan algunos resultados experimentales en simulación de tres comportamientos representativos de entornos urbanos: mantenimiento de carril, ceda el paso y paso de peatones. Se ha diseñado un escenario específico para verificar estos comportamientos, como se puede ver en la figura 6.

Durante la prueba, el vehículo usa sus sensores de percepción (LIDAR, cámara y GPS). La capa de control toma esta información y los *lanelets* y elementos reguladores, proporcionados por el administrador del mapa, para generar los comandos de velocidad para el control de bajo nivel, siguiendo el comportamiento ordenado por la capa ejecutiva a través de un conjunto de redes de Petri jerárquicas. Los resultados han demostrado que los datos de percepción, los algoritmos de navegación y los comandos de movimiento se han sincronizado correctamente mediante el uso de la arquitectura propuesta basada en ROS.

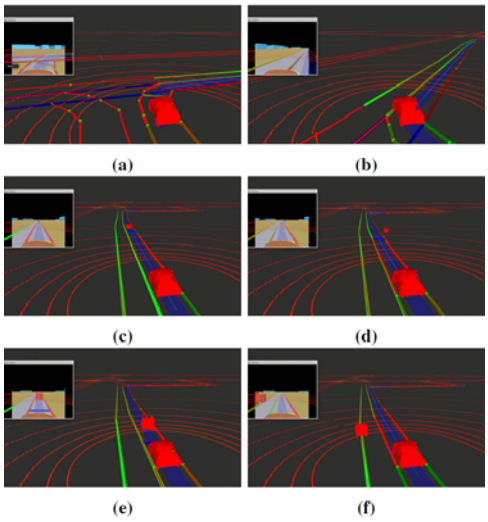

Figura 7: Estado de los diferentes comportamientos en la trayectoria seguida: (a) Ceda el paso (9.75 s); (b) Mantenimiento del carril (14.5 s); (c) Paso de peatones: Parada (29.0 s); (d) Paso de peatones: límitada máx. vel. (30.5 s); (e) Paso de peatones: parada (35.5 s); y (f) Mantenimiento del carril (38 s).

El cumplimiento de estos comportamientos se ilustra en la figura 7. En el comportamiento de mantenimiento del carril, el vehículo sigue la ruta global calculada dentro del carril, evitando colisionar con cualquier obstáculo. El control local reactivo se ejecuta teniendo en cuenta la velocidad máxima permitida para el carril por el que circula el vehículo. Los comportamientos del paso de peatones y ceda el paso funcionan de una manera similar. Se activan cuando el vehículo está cerca de la señal de tráfico (en esta simulación, la distancia se ha establecido en 30 m). Una vez que se activa el comportamiento, la velocidad máxima se reduce y el monitor evalúa si hay algún

obstáculo. El escenario de prueba (figura 6) muestra el vehículo viajando en una carretera recta con dos curvas cerradas (al principio y al final), un ceda el paso en la incorporación a la carretera principal y un paso de peatones, marcado por una línea azul transversal, ubicada a mitad de camino (figura 6). Un comportamiento de ceda el paso se desencadena a los 8 s., reduciendo la velocidad del automóvil, ya que debe detenerse antes de incorporarse a la carretera principal, esperando por cualquier vehículo con prioridad de paso (figura 7 (a)). Como en esta prueba simulada no se detectó ningún vehículo, entonces nuestro vehículo continúa girando a la derecha y siguiendo recto por la carretera principal (figura 7 (b)). Cuando se acerca al cruce de peatones, una persona cruza desde el carril izquierdo (moviéndose de izquierda a derecha). Se activa el comportamiento de paso de peatones para detener el vehículo antes del paso de peatones (figura 7 (c)). Cuando el peatón cruza la carretera, el vehículo reanuda su recorrido reduciendo su velocidad máxima porque se acerca al paso de peatones, y se pueden detectar nuevos peatones (figura 7 (d)) En esta simulación, un nuevo peatón aparece repentinamente desde la derecha cuando el automóvil está muy cerca del paso de peatones, el vehículo se detiene justo antes que el peatón sin colisionar con él (figura 7 (e)). Cuando el peatón cruza y no se detectan más peatones, el vehículo reanuda su camino comenzando por un estado de velocidad creciente y luego pasando al comportamiento de mantenimiento del carril, que se mantiene hasta que se alcanza la posición objetivo (figura 7 (f)).

10. Conclusiones y trabajos futuros

Como resultado de un proyecto de investigación, este artículo ha presentado los principales desarrollos en la arquitectura propuesta basada en ROS para nuestro vehículo autónomo en entornos urbanos. También se ha descrito el proceso de automatización de nuestro vehículo de código abierto.

El objetivo de esta arquitectura es la de proporcionar a nuestra plataforma las capacidades para ser utilizado como un vehículo funcional autónomo en el campus de la Universidad de Alcalá. El sistema de percepción se basa en la fusión de LIDAR, cámaras y GPS. La propuesta ha sido validada en simulación con algunos comportamientos típicos de conducción urbana como: mantenimiento de carriles, intersección de carriles y parada cuando un peatón está cruzando un paso de peatones. Como trabajo futuro, implementaremos los comportamientos presentados en el prototipo real, propondremos nuevos comportamientos tanto en simulación, como en condiciones reales y adaptaremos el HMI (*Human Machine Interface*) al sector de las personas mayores.

English Summary

Development of an open-source electric vehicle for elderly people.

Abstract

This paper presents the main results of a research project in technologies for the development of an autonomous electric car to be used by elderly people in urban environments. The software architecture is based on the operating system ROS (Robot Operating System). The proposed ROS architecture for the environment perception, navigation and planning in a university campus is described. To evaluate the system, different tests have been carried out in a simulator with some typical urban driving behaviors as lane keeping, lane intersection giving way and stop when a pedestrian is crossing a crosswalk. The developed architecture is being applied to an open source electric car aimed at a sector of the population with growing needs. As future work, we will implement the presented behaviors in the real vehicle and we will propose new behaviors for both simulation and real conditions.

Keywords:

Autonomous Driving, ADAS, 3D Lidar, Cameras, Sensor Fusion, 3D Mapping, Control and Planning, Human Factors.

Agradecimientos

La investigación presentada en este artículo ha sido financiada por los siguientes proyectos de investigación del Programa Estatal de I+D+i Orientada a los Retos de la Sociedad del Ministerio de Economía y Competitividad: Vehículo inteligente para personas mayores (TRA2015-70501-C2-1-R (MINECO/FEDER)) y *Smartelderlycar*: control y planificación de rutas (TRA2015-70501-C2-2-R (MINECO/FEDER)); y por el proyecto *RoboCity2030-III-CM* (S2013/MIT-2748), financiado por los Programas de actividades I+D (CAM) y cofinanciado con fondos estructurales de la UE.

Referencias

Bender, P., Ziegler, J., Stiller, C., 2014. Lanelets: Efficient map representation for autonomous driving. In IEEE Proc. Intelligent Vehicles Symposium, pp. 420–425.

Coulter, R., 1992. Implementation of the pure pursuit path-tracking algorithm, Carnegie Mellon University, the Robotics Institute, Pittsburgh, Pa., USA.

J. L. Fernández, J.L., Sanz, R., Benayas, J., Diéguez, A., 2004. Improving collision avoidance for mobile robots in partially known environments: the Beam Curvature Method. Robotics and Autonomous Systems, vol. 46, no. 4, pp. 205–219.

Fernández, J. L., Sanz, R., Paz, E., and Alonso, C., 2008. Using hierarchical binary Petri nets to build robust mobile robot applications: RoboGraph". In IEEE International Conference on Robotics and Automation (ICRA 2008), Vol. I, pp. 1372-1377.

Held, D., Levinson, J., Thrun, S., 2013. Precision tracking with sparse 3d and dense color 2d data. In IEEE International Conference on Robotics and Automation (ICRA). pp. 1138–1145.

López, J., Losada, D.P., Zalama, E., 2011. A framework for building mobile single and multi-robot applications. Robotics and Autonomous Systems, vol. 59, pp. 151-162.

Montemerlo, M., Roy, N., Thrun, S. 2003. Perspectives on standardization in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) Toolkit. In IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, (IROS 2003), Las Vegas, USA.

Romera, E., Alvarez, J. M., Bergasa, L. M., Arroyo, R., 2018. ERFNET: Efficient residual factorized convnet for real-time semantic segmentation. IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 1, pp. 263–272.

Tabby Evo chassis, 2018. Open motors. https://www.openmotors.co/product/tabbyevo/.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., 2008. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. Journal of Field Robotics, vol. 25, pp. 425-466.