Diseño y construcción de un vehículo skid-steer compatible con ROS

Agustín Capovilla
sinc(i)
UNL-CONICET
Santa Fe, Argentina
acapovilla@sinc.unl.edu.ar

Guido Sánchez
sinc(i)
UNL-CONICET
Santa Fe, Argentina
gsanchez@sinc.unl.edu.ar

Marina Murillo
sinc(i)
UNL-CONICET
Santa Fe, Argentina
mmurillo@sinc.unl.edu.ar

Leonardo Giovanini
sinc(i)
UNL-CONICET
Santa Fe, Argentina
lgiovanini@sinc.unl.edu.ar

***Resumen*—Los vehículos *skid-steer* son una de las configuraciones más populares utilizadas en Vehículos Terrestres Autónomos (AGVs, del inglés *Autonomous Ground Vehicles*). La ausencia de un mecanismo de dirección explícito los convierte en vehículos compactos y livianos. La configuración *skid-steer* reduce significativamente la complejidad mecánica, haciendo que sean fáciles de construir y capaces de realizar tareas complicadas tanto en ambientes *outdoor* como *indoor*. Sin embargo, el proceso de construcción y puesta en marcha (hardware y software) de un vehículo que sea capaz de operar de forma autónoma generalmente requiere pasar mucho tiempo resolviendo problemas que ya han sido abordados. Este trabajo detalla el proceso requerido para construir *desde cero* un AGV que sea compatible con Sistema Operativo Robótico (ROS, del inglés *Robot Operating System*). Se describen *todas* las tareas de construcción, partiendo de los materiales, las partes mecánicas, los componentes electrónicos, el desarrollo del firmware que se ejecuta en el microcontrolador y el software que se ejecuta en la computadora de a bordo. El vehículo propuesto es un AGV construido con componentes que se encuentran disponibles en el mercado local o son fáciles de reemplazar, lo que proporciona un banco de pruebas estandarizado para la investigación y el desarrollo en el área de vehículos autónomos.**

***Keywords*—autonomous ground vehicle, robot operating system, skid-steer, raspberry pi, bluepill**

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
A. Capovilla, G. Sanchez, M. Murillo & L. Giovanini; "Diseño y construcción de un vehículo skid-steer compatible con ROS"
XI Jornadas Argentinas de Robótica, 2022.

I. Introducción

Los AGV con configuración *skid-steer* son mecánicamente simples, ágiles y poseen gran capacidad de tracción en distintos tipos de superficies. Son considerados la elección apropiada para desplazarse en terrenos difíciles, pero conservan una gran maniobrabilidad. Esto hace que este tipo de vehículos hayan ganado popularidad en distintas aplicaciones civiles y militares, tales como vigilancia, búsqueda y rescate, exploración, detección de explosivos, agricultura de precisión, aplicaciones industriales, vehículos de servicio, entre otros [1]-[3]. En los últimos años el interés en el desarrollo y construcción de AGVs se ha incrementado enormemente, sin embargo, al tratar de cumplir con la tarea de hacer que un AGV funcione realmente de forma autónoma, los desarrolladores e investigadores suelen dedicar mucho tiempo a resolver problemas que ya están resueltos, como el desarrollo de modelos cinemáticos de vehículos, control PID, integración de sensores, entre otros. Se han realizado muchos esfuerzos para crear vehículos terrestres [4]-[6], cuadricópteros [7], submarinos [8] y embarcaciones [9]. Hay un gran número de AGV comerciales y sus fabricantes incluyen a Clearpath Robotics [10], TurtleBot [11] y AgileX [12]. Existen proyectos *open-source* tales como Linorobot [13] y proyectos académicos [14], [15]. Sin embargo, las soluciones comerciales generalmente son demasiado costosas y hay casos en que no es sencillo reconfigurarlas o adaptarlas a otros requerimientos. La mayoría de los proyectos *open-source* proveen una guía, pero usualmente carecen de un instructivo completo y en muchos de los casos están dirigidos a la construcción de vehículos para operación en ambientes *indoor*. En este contexto, ROS [16], [17] se ha convertido en el estándar *de facto* para el desarrollo de software de robótica. Es un *framework* flexible para escribir software de robots, que posee una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear robots complejos y robustos para varios casos de uso. Además, tiene una amplia variedad de paquetes disponibles probados por la comunidad. El núcleo de ROS tiene una licencia abierta que permite su utilización en productos comerciales y de código cerrado.

El presente trabajo extiende los resultados obtenidos en «Hibachi: A ROS compatible skid-steer vehicle» [18], donde se hizo énfasis en el desarrollo del software necesario para que el vehículo sea compatible con ROS. El trabajo giró en torno a la utilización de una placa orientada al desarrollo de autopilotos ***Navio2*** [19], la cual fue útil para obtener un prototipo inicial de forma rápida, sin embargo, esto quitó flexibilidad de integración tanto de software como de hardware. Por estas razones, este trabajo hace énfasis no solo en disminuir los costos, sino minimizar la dependencia sobre distintos tipos de componentes con el fin de incrementar la capacidad de construcción, modificación, adaptación y reparación del vehículo.

Este trabajo detalla ***íntegramente*** el proceso constructivo de un AGV con configuración *skid-steer* compatible con ROS. Esto implica la selección de componentes mecánicos y electrónicos, el diseño del conexionado entre el microcontrolador y la computadora de a bordo (de ahora en más CPU), así como de la distribución de energía. Se detalla el proceso de desarrollo del firmware de bajo nivel, su comunicación con la CPU, el desarrollo de software que permita que el AGV se comporte como un nodo de ROS y finalmente, la configuración los paquetes necesarios que permitan que el

vehículo calcule su odometría y reciba acciones de control. Se pretende detallar de forma integral el ciclo de desarrollo entre hardware disponible en el mercado local y ROS, permitiendo concentrar los esfuerzos en resolver problemas interesantes específicos de los AGV, en lugar de *reinventar la rueda*.

II. Descripción del hardware

Con respecto al hardware, se espera construir un AGV que sea fácil de ensamblar, fácil de reparar y que utilice materiales disponibles en el mercado local. Por este motivo, la elección de la configuración *skid-steer* resultó ser la más apropiada desde el punto de vista mecánico y de prestaciones. La selección de los componentes se realizó de forma estratégica, en parte reciclando hardware preexistente en el sinc(*i*) y agregando hardware elegido cuidadosamente para cumplir los requerimientos. El AGV propuesto consta de los siguientes componentes electrónicos:

- **Raspberry Pi 3B+**: Es un ordenador de placa reducida que es capaz de correr Raspberry Pi OS versión 10 (Buster) y ROS Noetic. La Raspberry Pi está conectada a un microcontrolador para recibir los datos de odometría y enviar *setpoints* de velocidad angular.
- **BluePill** [20]: Este microcontrolador está basado en el STM32F103C8T6 ARM Cortex-M3. Posee una frecuencia de reloj de 72 MHz, 48 pines de entrada/salida, 64KiB de FLASH y 20KiB de ROM. Este microcontrolador se encarga de mantener la velocidad de los motores mediante control PID a partir de la información que intercambia con la Raspberry Pi.
- **Motores DC con encoders**: Dado que el AGV propuesto utiliza una configuración *skid-steer*, se deben controlar cuatro motores. Cada uno de ellos opera a 12V y cuentan con caja reductora y encoder de cuadratura de fábrica, lo que permite obtener mediciones de posición y velocidad angular de las ruedas con el fin de calcular la odometría.
- **Puentes-H L298N**: Estos integrados son ampliamente utilizados en robótica educacional y prototipado por su bajo costo y fácil integración. El vehículo utiliza dos puentes-H y cada uno de ellos controla dos motores. Estos integrados reciben la señal PWM (Pulse Width Modulation) de la BluePill y en función de ello ajustan el voltaje de salida hacia los motores.

Estos componentes pueden visualizar en la Fig. 1 y se interconectan según el diagrama de la Fig. 2, donde se puede apreciar que las líneas de color rojo son de alimentación, las líneas de color verde son los datos provenientes de los encoders, las lineas azules representan las señales Pulse Width Modulation y la línea amarilla es la comunicación USB entre la Raspberry Pi y la BluePill.

II-A. Sistema electromecánico

El robot está construido a partir de una plancha de acrílico cortada por láser de 400mm de largo por 250mm de ancho y 5mm de espesor. Un material accesible y de bajo costo que permite un prototipado rápido gracias a su fácil manipulación y mecanización, sin perder confiabilidad y resistencia mecánica.

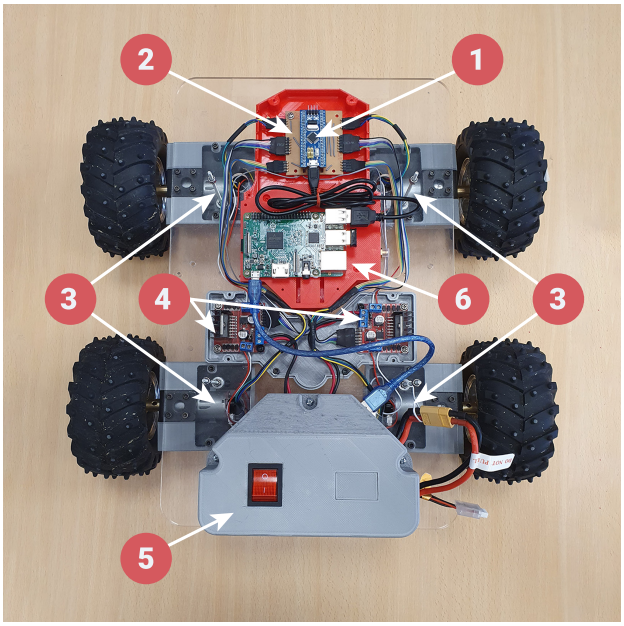

Figura 1. Principales componentes del vehículo desarrollado. 1) BluePill, 2) CarrierBoard, 3) Motores, 4) Puentes-H, 5) Batería y 6) Raspberry Pi.

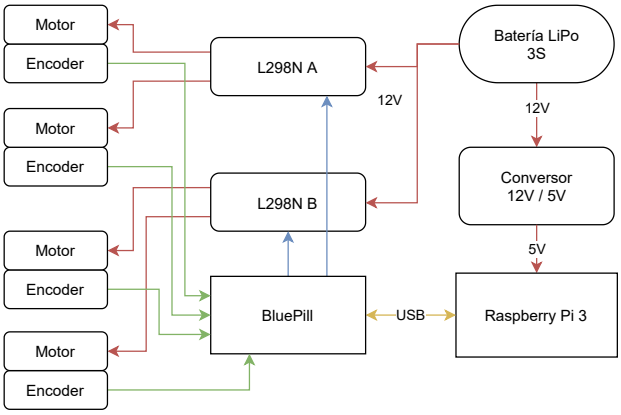

Figura 2. Conexionado de los componentes electrónicos.

El resto de piezas de montaje están impresas en 3D mediante Modelado por Deposición Fundida (FDM, del inglés *Fused Deposition Modeling*) en material PLA (PolyLactid Acid). En caso de no contar con dicha tecnología o la necesidad de disminuir el costo total, dichas piezas pueden ser reemplazadas por anclajes sujetos directamente a la base de acrílico.

El sistema de propulsión está compuesto por cuatro motores de 12V DC y una batería. El conjunto se completa con ruedas de 120mm todo terreno (tipo DAGU Wild Thumper) con adaptadores compatibles con los ejes de los motores.

II-B. Sistema electrónico

El sistema se encuentra alimentado por una batería recargable de 11.1V LiPo (lithium-ion polymer) la cual está conectada a los puentes-H L298N que se encargan de controlar la dirección de giro y la velocidad de los motores. Para los componentes de baja potencia (CPU, microcontroladores,

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
A. Capovilla, G. Sanchez, M. Murillo & L. Giovanini; "Diseño y construcción de un vehículo skid-steer compatible con ROS"
XI Jornadas Argentinas de Robótica, 2022.

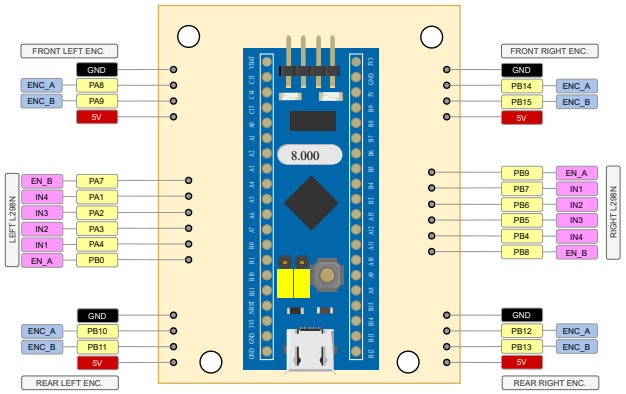

Figura 3. Conexionado entre la BluePill, puentes-H y encoders

encoders, etc.) se monta un regulador de 5V el cual les provee la alimentación.

El sistema de control se divide en dos capas, una de bajo nivel compuesta por el microcontrolador BluePill y un controlador de alto nivel compuesto por computadora de placa reducida Raspberry Pi 3B+. Esta arquitectura permite obtener lo mejor de ambos ecosistemas y lograr modularidad, asimismo brinda un alto grado de autonomía, requiriendo la intervención del usuario solo para enviar planes de misión o para detenerse ante una emergencia. Por un lado, el microcontrolador permite el uso de periféricos específicos como Pulse Width Modulation, temporizadores, interrupciones por hardware, entre otros; pero carecen de capacidad de cómputo y soporte para sistemas operativos compatibles con aplicaciones como ROS. En contraposición, la CPU permite realizar gran cantidad de cálculos y ejecutar aplicaciones de alto nivel, tales como algoritmos de planeamiento y navegación.

La comunicación física entre ambas capas se realiza a través de USB el cual también provee alimentación al microcontrolador. Este último se monta sobre una PCB diseñada para acceder fácilmente a las conexiones necesarias para leer la señales provenientes de los encoders, alimentarlos y además controlar los puentes-H. Para finalizar, la conexión entre la BluePill y los encoders como los puentes-H sigue el diagrama que se puede ver en la Fig. 3 y se encuentran conectados mediante cables finos de tipo AWG28 y conectores de tipo Dupont. En ambos circuitos impresos, el diseño se realizó pensando en métodos tradicionales de fabricación que solo requieran placas de una sola capa.

III. Descripción del software

Uno de los principales beneficios de utilizar el ecosistema de ROS es el aprovechamiento de una gran cantidad de paquetes existentes ampliamente probados en la comunidad, por lo que ciertas tareas se ven simplificadas cuando se intenta desarrollar un robot desde cero. Para que un robot se considere compatible con ROS, se requiere desarrollar el software necesario para convertirlo en un ***nodo*** de ROS, de forma tal que sea capaz de interactuar e intercambiar información con otros nodos y paquetes [5]. El vehículo debe:

- Publicar la información de su geometría utilizando el Formato Unificado para Descripción de Robots (URDF, del inglés *Unified Robot Description Format*),
- Publicar la información de sus sensores y la odometría utilizando ***topics*** de ROS,
- Recibir acciones de control provenientes de otros nodos utilizando ***topics*** de ROS.

Otro conjunto de requerimientos es el cumplimiento de los ROS Enhancement Proposals (REP), tales como el REP-103: "*Standard Units of Measure and Coordinate Conventions*" [21], que proporciona una referencia de las convenciones de unidades y coordenadas utilizadas en ROS; y REP-105: "*Coordinate Frames for Mobile Platforms*" [22], que especifica las convenciones y el significado semántico para el nombramiento de los marcos de referencia utilizadas en plataformas móviles con ROS.

Se desarrollaron los siguientes paquetes para convertir el vehículo en un nodo de ROS:

- `hibachi_description`: contiene la descripción geométrica del vehículo y sus partes en formato URDF.
- `hibachi_firmware`: contiene el firmware que se ejecuta en la BluePill. Gestiona la comunicación entre el microcontrolador y la CPU, recibe *setpoints* de velocidad para cada uno de los motores, se encarga que estos *setpoints* se cumplan mediante control PID y envía la información de los encoders a la CPU.
- `hibachi_base`: es el nodo que realiza la interfaz con el hardware. Posee la definición de cada una de las juntas del vehículo y sus correspondientes interfaces de control. Se encarga de realizar el nexo entre ROS y la BluePill.

III-A. El paquete `hibachi_firmware`

El firmware de la BluePill se encuentra programado en el paradigma orientado a objetos con distintas clases que en conjunto conforman una librería que permite la correcta operación de los componentes conectados al microcontrolador. Las dos tareas principales a ejecutar son:

1. La recepción de *setpoints* de velocidad angular para los lazos de control de cada uno de los motores.
2. El envío de la información de los encoders hacia la CPU para el cálculo de la odometría.

La implementación específica para el presente trabajo se encuentra distribuida en tres archivos. Primero, el header `Hibachi.h` donde se encuentran definidos los pines GPIO utilizados, ciertas variables mecánicas como la cantidad de pulsos por revolución de los encoders y los parámetros por defecto de los controladores PID. Segundo, el archivo fuente correspondiente `Hibachi.cpp`, donde se encuentran las instancias de las clases por cada conjunto motor-encoder, los controladores de velocidad que implementan un controlador PID y la rutina de control del puerto de comunicación, que al recibir un comando, lo procesa y ejecuta las correspondientes acciones. Los comandos definidos se utilizan para establecer un nuevo *setpoint* de velocidad para cada motor y para obtener la medición de posición y velocidad. Para finalizar,

sinc(i) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
A. Capovilla, G. Sanchez, M. Murillo & L. Giovanini; "Diseño y construcción de un vehículo skid-steer compatible con ROS"
XI Jornadas Argentinas de Robótica, 2022.

el archivo `main.cpp` es el que ejecuta todas las rutinas de inicialización, incluyendo la configuración de un temporizador para ejecutar de forma continua los cálculos de los lazos de control. Luego de esto, el programa queda a la espera de nuevos comandos para procesar.

III-B. El paquete `hibachi_base`

Una de las maneras de implementar un robot utilizando ROS, es mediante el modelado del robot en software, representado por la interfaz de hardware (extendiendo la clase `hardware_interface::RobotHW`). Un aspecto importante a destacar es que el acceso al hardware del robot está desacoplado, por lo que no se expone el protocolo de comunicación con el hardware de bajo nivel. Esto implica que los controladores y la estructura de hardware se pueden desarrollar de forma independiente y aisladas, lo que permite mezclar y reutilizar controladores de distintos tipos [23]. Para el vehículo en cuestión, se deben controlar las juntas que de describen los cuatro motores del robot, por lo que se hace uso de `hardware_interface::JointStateInterface` para publicar la posición y velocidad de las juntas y `hardware_interface::VelocityJointInterface` para enviar *setpoints* de velocidad a las juntas. De esta forma, cada una de las juntas que modelan los motores están expuestas utilizando interfaces estándar de ROS, permitiendo que cualquier software de terceros que haya sido diseñado para operar con dichas interfaces sea compatible con nuestro vehículo [24], evitando así tener que escribir código para resolver muchas tareas comunes que ya han sido resueltas.

IV. Instrucciones de armado

Dada la naturaleza modular del diseño, es posible realizar la construcción del vehículo sin seguir un orden particular, excepto por algunos pasos secuenciales. A su vez, esta modularidad permite el cambio o reemplazo de ciertas partes o componentes según requerimientos y disponibilidad, sin que afecten al funcionamiento en conjunto.

Como primer paso se montan los motores a la plancha de acrílico. Cada uno de los motores posee sus respectivas monturas en "L". En caso de adquirir por separado la caja reductora y/o el encoder, se recomienda ensamblarlos antes de fijarlos. La alimentación de los motores se realiza a través de las salidas de potencia de los puentes-H. Dado que el integrado se encuentra montado en la PCB, la conexión entre ellos se realiza mediante borneras a tornillo. En el caso de los encoders, se conectan a la PCB de la BluePill a través de un conector tipo Dupont. Cabe recordar que la conexión de los puentes-H y los encoders al PCB sigue el esquema de la Fig. 3. Una vez se tienen los conjuntos ensamblados, se instalan junto con sus correspondientes protectores impresos en 3D.

Por otro lado se montan las dos PCB que contienen los puentes-H junto con su caja protectora. El integrado L298N con el que cuentan los puentes-H posee la capacidad de controlar dos motores DC a la vez, dado que cuenta con dos drivers independientes con salidas separadas. Para la correcta operación y evitar el daño del integrado se debe respetar el esquema de conexiones de la Fig. 3 al conectar cada uno de los motores. Cabe destacar que la polaridad en la que se conecten los motores influirá luego en la definición de los GPIO en el firmware del microcontrolador.

El conjunto de alimentación y distribución de energía del vehículo está compuesto por la batería, el regulador para bajo voltaje y la PCB de distribución. Dichos componentes deben ser conectados según el diagrama que se muestra en la Fig. 2 y son montados dentro de la correspondiente caja contenedora. El cableado de potencia se realiza mediante cables de 1mm² para soportar la corriente de los motores y la alimentación a las placas de la CPU y el microcontrolador. La instalación del adaptador USB, permite la alimentación de la Raspberry Pi mediante un cable utilizando el puerto micro USB.

La PCB donde se monta la BluePill, denominada *CarrierBoard*, cuenta en ambos laterales con los conectores necesarios para la alimentación y la comunicación de las señales provenientes de los encoders, así como la interfaz para las dos PCB que montan los puentes-H. Luego del montaje de la CarrierBoard y cableado de los elementos antes descritos se recomienda realizar un primer ensayo para verificar el correcto funcionamiento del conjunto electromecánico. Para ello se propone cargar el firmware `test-01.cpp`, el cual consiste en el accionamiento individual de cada uno de los motores en la dirección "forward" y luego en la dirección "backward" escribiendo en el puerto serie la posición angular y la velocidad de cada uno de los encoders. En el caso de que la dirección de rotación del motor sea inversa, se deben intercambiar, en el firmware, los GPIO definidos para el control del sentido de rotación. Para el caso que el sentido de rotación del motor sea el correcto pero el encoder reporte una velocidad de signo opuesto, se deben intercambiar los GPIO definidos como canales de entrada del encoder.

Una vez comprobado el correcto funcionamiento de los motores en función de las pruebas de dirección y control de velocidad, se procede a la instalación en la BluePill del firmware operativo para ROS. Por último, la BluePill debe ser conectada a uno de los puertos USB de la Raspberry Pi mediante el cable correspondiente, para así alimentar y comunicar físicamente ambos dispositivos.

La Raspberry Pi requiere una instalación de GNU/Linux compatible con ROS Noetic. En este trabajo se utilizó Debian 10 (Buster) para arquitectura `arm64`. Luego de instalar ROS siguiendo la documentación oficial, es necesario crear un directorio de trabajo de `catkin` e instalar los paquetes `hibachi_description`, `hibachi_firmware`, `hibachi_base` y `diff_drive_controller`.

V. Validación y resultados experimentales

Para validar el correcto funcionamiento del AGV se realizaron los siguientes experimentos:

1. Desplazarse en línea recta a 0.5 m/s hasta los 1.80 m.
2. Rotar en el lugar a 86°/s hasta alcanzar 180°.
3. Seguir una trayectoria representada por un rectángulo. Esto requiere seguir la secuencia de puntos dada por: $(0,0)$, $(2,0)$, $(2,1{,}5)$, $(0,1{,}5)$ y $(0,0)$.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
A. Capovilla, G. Sanchez, M. Murillo & L. Giovanini; "Diseño y construcción de un vehículo skid-steer compatible con ROS"
XI Jornadas Argentinas de Robótica, 2022.

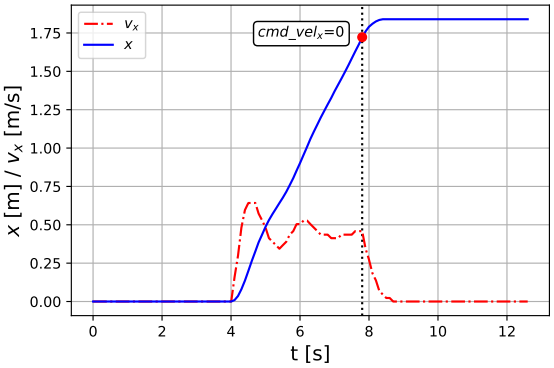

Figura 4. Cálculo de odometría para un recorrido en línea recta.

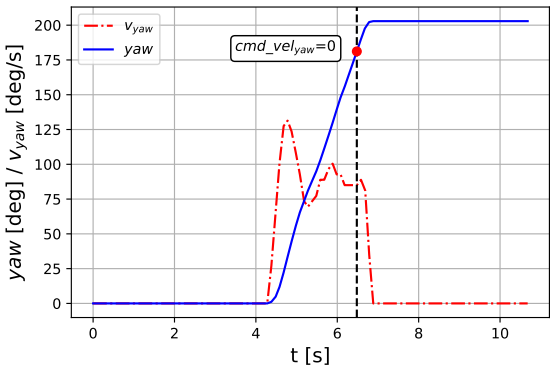

Figura 5. Cálculo de odometría para un giro en el lugar.

La Fig 4 muestra la evolución de la estimación de la posición y la velocidad en el eje-x con respecto al tiempo. Se puede ver como la velocidad en el eje-x oscila alrededor de 0.5 m/s y cerca de los 8 segundos el algoritmo de control detecta que el vehículo a llegado a recorrer 1.8 m, por lo que envía el comando de frenado (`cmd_vel_x=0`). La inercia del vehículo hace que se siga desplazando hasta un valor cercano a 2 m. Este fenómeno se puede apreciar al observar que la gráfica de la velocidad lineal empieza a decrecer a partir de los 8 segundos.

La Fig 5 muestra la evolución de la estimación del ángulo de *yaw* y la velocidad angular en el eje-z con respecto al tiempo. Se puede ver como la velocidad en el eje-z oscila alrededor de 86°/s hasta que el algoritmo detecta que el vehículo llegó a girar los 180° pasados los 6 segundos. De forma similar al experimento anterior, luego de que el comando de control es cero, el vehículo se sigue moviendo producto de la inercia.

Finalmente se configuró un controlador utilizando la técnica MPC (Model Predictive Control) para que el vehículo siga un conjunto de *waypoints* utilizando la información provista por el cálculo de odometría. Este experimento se realizó siete veces. Como se puede observar en la Fig. 6, los puntos rojos son los *waypoints* y las distintas líneas representan el recorrido

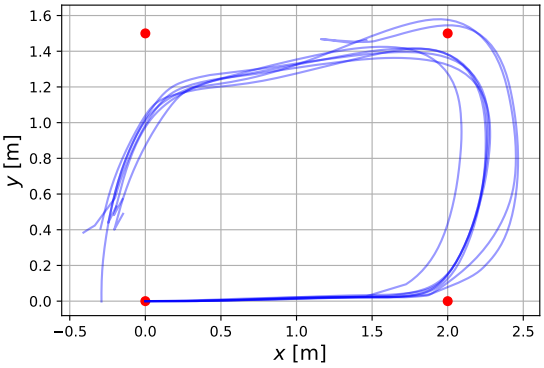

Figura 6. Seguimiento de cuatro *waypoints* utilizando MPC y odometría.

del vehículo en las coordenadas xy. Como es de esperarse, las estimaciones de posición utilizando solo la odometría se degradan con el tiempo, lo que evidencia la necesidad de utilizar otros sensores como fuentes de datos complementarias a fin de mejorar la capacidad de navegación del vehículo.

VI. Conclusiones y trabajos futuros

El diseño y construcción de un AGV compatible con ROS brinda la posibilidad de acceder a una plataforma estandarizada que permite a los desarrolladores e investigadores dejar de implementar soluciones a problemas que ya han sido resueltos y enfocarse en resolver problemas interesantes en las distintas áreas de aplicación de AGV. La utilización de ROS facilita la integración e intercambio de sensores. Por ejemplo, se puede cambiar de forma sencilla una Unidad de Medición Inercial (IMU, del inglés *Inertial Measurement Unit*) por otra, siempre y cuando publique sus datos en un mensaje de tipo `sensor_msgs/Imu` y cumpla con los estándares definidos en la REP-103.

La utilización de componentes disponibles en el mercado local facilita la capacidad de construcción y esperamos que este trabajo haga que la adopción de ROS sea más simple para quienes no se han aventurado en este camino. Una vez que el hardware del vehículo está configurado correctamente, es capaz de recibir comandos de tipo `geometry_msgs/Twist` para calcular sus acciones de control y enviar datos para el cálculo de la odometría local, solo resta agregar los sensores y paquetes de ROS necesarios de acuerdo a los requerimientos de cada aplicación. Para dar otro ejemplo, si se desea realizar navegación autónoma *indoor* y esquivar obstáculos, es necesario contar con una IMU para complementar el cálculo de odometría de los encoders y sensores tales como LIDAR o cámaras estéreo para detección de obstáculos; posteriormente se configuran los paquetes de ROS, tal como el stack de `navigation` y estamos en condiciones de realizar la tarea.

El grupo de trabajo planea agregar una IMU de tipo LSM9DS1, un módulo GPS y un LIDAR con el fin de poder realizar experimentos en ambientes *indoor* y *outdoor*. Esto requiere escribir *drivers* para y definir esquemas de conexiones

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
A. Capovilla, G. Sanchez, M. Murillo & L. Giovanini; "Diseño y construcción de un vehículo skid-steer compatible con ROS"
XI Jornadas Argentinas de Robótica, 2022.

para algunos componentes de hardware. Asimismo, se pretende comenzar a migrar el software desarrollado a ROS2, ya que ROS Noetic está llegando al fin del ciclo de vida.

Finalmente, queremos destacar que el código fuente utilizado en el desarrollo de este AGV se encuentra disponible en `https://github.com/gmsanchez/hibachi`.

AGRADECIMIENTOS

Los autores agradecen a la *Universidad Nacional de Litoral*, la *Agencia Nacional de Promoción Científica y Tecnológica* y el *Consejo Nacional de Investigaciones Científicas y Técnicas* (CONICET) de Argentina.

REFERENCIAS

[1] H. Mousazadeh, «A technical review on navigation systems of agricultural autonomous off-road vehicles,» *Journal of Terramechanics*, vol. 50, n.º 3, págs. 211-232, 2013.

[2] J. A. Thomasson, C. P. Baillie, D. L. Antille, C. R. Lobsey, C. L. McCarthy y col., *Autonomous technologies in agricultural equipment: A review of the state of the art*. American Society of Agricultural y Biological Engineers, 2019.

[3] R. Khan, F. Mumtaz, A. Raza y N. Mazhar, «Comprehensive study of skid-steer wheeled mobile robots: development and challenges,» *Industrial Robot*, vol. 48, págs. 142-156, ago. de 2020. DOI: 10.1108/IR-04-2020-0082.

[4] A. Araújo, D. Portugal, M. S. Couceiro, J. Sales y R. P. Rocha, «Desarrollo de un robot móvil compacto integrado en el middleware ROS,» *Revista Iberoamericana de Automática e Informática industrial*, vol. 11, n.º 3, págs. 315-326, 2014, ISSN: 1697-7920. DOI: 10.1016/j.riai.2014.02.009.

[5] S. S. H. Hajjaj y K. S. M. Sahari, «Bringing ROS to Agriculture Automation: Hardware Abstraction of Agriculture Machinery,» *International Journal of Applied Engineering Research*, vol. 12, n.º 3, págs. 311-316, 2017.

[6] D. F. Carlson, A. Fürsterling, L. Vesterled, M. Skovby, S. S. Pedersen, C. Melvad y S. Rysgaard, «An affordable and portable autonomous surface vehicle with obstacle avoidance for coastal ocean monitoring,» *HardwareX*, vol. 5, e00059, 2019, ISSN: 2468-0672. DOI: https://doi.org/10.1016/j.ohx.2019.e00059.

[7] A. Majumdar, N. Gamez, P. Benavidez y M. Jamshidi, «Development of robot operating system (ROS) compatible open source quadcopter flight controller and interface,» en *2017 12th System of Systems Engineering Conference (SoSE)*, 2017, págs. 1-6. DOI: 10.1109/SYSOSE.2017.7994980.

[8] E. Gallimore, R. Stokey y E. Terrill, «Robot Operating System (ROS) on the REMUS AUV using RECON,» en *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*, 2018, págs. 1-6. DOI: 10.1109/AUV.2018.8729755.

[9] D. Betancur-Vásquez, M. Mejia-Herrera y J. Botero-Valencia, «Open source and open hardware mobile robot for developing applications in education and research,» *HardwareX*, vol. 10, e00217, 2021, ISSN: 2468-0672. DOI: https://doi.org/10.1016/j.ohx.2021.e00217.

[10] *Clearpath Robotics*, https://clearpathrobotics.com/, Accessed: 2022-01-24.

[11] *TurtleBot*, https://www.turtlebot.com/, Accessed: 2022-01-24.

[12] *AgileX*, https://global.agilex.ai/, Accessed: 2022-01-24.

[13] *Linorobot*, https://linorobot.org/, Accessed: 2022-01-24.

[14] *Noah Robot (hardware)*, https://github.com/GonzaCerv/noah-hardware, Accessed: 2022-01-24.

[15] *Noah Robot (software)*, https://github.com/GonzaCerv/noah-software, Accessed: 2022-01-24.

[16] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler y A. Ng, «ROS: an open-source Robot Operating System,» en *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, mayo de 2009.

[17] *Robot Operating System (ROS)*, https://www.ros.org/, Accessed: 2021-08-01.

[18] G. M. Sánchez, M. H. Murillo, J. E. Benavidez, N. N. Deniz y L. L. Giovanini, «Hibachi: A ROS compatible skid-steer vehicle,» en *2021 XIX Workshop on Information Processing and Control (RPIC)*, 2021, págs. 1-6. DOI: 10.1109/RPIC53795.2021.9648498.

[19] *Emlid Navio2*, https://navio2.emlid.com/, Accessed: 2022-01-24.

[20] *Blue Pill (STM32F103C8T6)*, https://stm32-base.org/boards/STM32F103C8T6-Blue-Pill.html, Accessed: 2022-01-24.

[21] T. Foote y M. Purvis, *Standard Units of Measure and Coordinate Conventions*, https://www.ros.org/reps/rep-0103.html, Accessed: 2021-08-01, oct. de 2010.

[22] W. Meeussen, *Coordinate Frames for Mobile Platforms*, https://www.ros.org/reps/rep-0105.html, Accessed: 2021-08-01, oct. de 2010.

[23] F. Pucher, *ROS Control, An overview*, https://fjp.at/posts/ros/ros-control/, Accessed: 2021-08-01, mar. de 2020.

[24] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. R. Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke y E. F. Perdomo, «ros_control: A generic and simple control framework for ROS,» *Journal of Open Source Software*, vol. 2, n.º 20, pág. 456, 2017. DOI: 10.21105/joss.00456.