RIELAC, Vol. 41 3/2020 p. 79-92 Septiembre-Diciembre ISSN: 1815-5928

RIELAC
Revista de Ingeniería Electrónica, Automática y Comunicaciones

Implementación de sistema operativo robótico en una plataforma de robot móvil

Carlos Guillermo Miguélez Machado, Ivón Oristela Benítez González, Alex Manuel Rivera Rivera, Valery Moreno Vega

RESUMEN / *ABSTRACT*

En el presente trabajo se realiza una implementación del Sistema Operativo Robótico (ROS) en un robot móvil diseñado para la interacción y el servicio humano. La plataforma utiliza un sistema digital basado en una placa Arduino Mega y una computadora de placa única (SBC) Raspberry Pi en comunicación con una computadora remota. La placa del microcontrolador realiza todas las operaciones de bajo nivel sobre el hardware. Se estableció una comunicación en serie entre el microcontrolador y la Raspberry Pi que ejecuta varios paquetes ROS para manipular la información. Una cámara Raspberry Pi está conectada al SBC y las imágenes capturadas se envían a la computadora remota para el procesamiento de las mismas utilizando ROS y OpenCv; la respuesta se genera dependiendo de la imagen tomada. El trabajo aporta la descripción de la implementación modular distribuida en capas de ROS en una plataforma móvil desarrollada en Cuba a un costo más bajo que muchas de las existentes en el mercado internacional.

Palabras claves: Sistema operativo robótico (ROS), robot móvil, Raspberry Pi, Arduino

In this paper an implementation of Robotic Operating System (ROS) on a mobile robot designed for human interaction and service is presented. The platform is designed using a digital system based on an Arduino Mega board and a Single Board Computer (SBC) Raspberry Pi in communication with a remote computer. The microcontroller board is used to perform all low-level operations over the hardware. It's established a serial communication between microcontroller and Raspberry Pi which is running various ROS packages to manipulate information. A Raspberry Pi camera it's connected to the SBC and the frames captured by the camera are sent to the remote computer for image processing using ROS and OpenCv; the response is generated on the remote computer in dependence of the image taken. The project gives the description of a layered distributed implementation of ROS in a mobile platform developed in Cuba in lower cost that many present on the market.

Key words: Robotic Operating System (ROS), mobile robot, Raspberry Pi, Arduino board

Robotic Operating System Implementation on a Mobile Robot Platform

1. -INTRODUCCIÓN

Actualmente la robótica móvil es uno de los campos de mayor nivel de expansión en la investigación científica [1]. Las aplicaciones principales incluyen vigilancia, exploración, patrullaje, rescate, entretenimiento, reconocimiento, aplicaciones industriales, servicios personales, entre otras [2,3]. La robótica móvil ha ido evolucionando desde los robots tele-operados que dependen del factor humano hasta su remplazo por un agente robótico resolviendo el problema del error humano pero incrementando la necesidad de algoritmos más sofisticados [4].

La principal complejidad de la robótica en su desarrollo es la implementación de algoritmos para las diferentes funciones que enmarcan el comportamiento de un robot. Además, la eficiente comunicación entre los módulos del mismo es una tarea engorrosa debido a la gran cantidad de procesos ejecutándose en paralelo y a la transferencia de datos entre cada uno de estos

Recibido: 30/4/2020 Aceptado: 3/10/2020

procesos (lectura de sensores, procesamiento de los datos, ejecución sobre los actuadores, entre otros). Aun así, la ciencia a nivel mundial ha mostrado avances importantes en esta área [2], a tal punto que construir un robot desde cero no se considera un aporte significativo, sin embargo, no deja de ser una ardua tarea. De ahí la importancia de la utilización de un sistema operativo para robots con cierta abstracción del hardware que tenga implementado varias de las tareas básicas que han de ser desarrolladas en una plataforma robótica [5, 6].

El sistema operativo robótico (ROS, por sus siglas en inglés) es un *middleware* para el desarrollo de software con aplicaciones en la robótica [7]. ROS brinda servicios como abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comúnmente utilizadas, transmisión de mensajes entre procesos y administración de paquetes de datos [5]. Los conjuntos de procesos basados en ROS se representan en una arquitectura gráfica donde el procesamiento tiene lugar en nodos que pueden recibir, publicar y multiplexar datos de sensores, control, estado, planificación y mensajes de actuadores [6].

ROS también incorpora herramientas de visualización como RViz para visualizar los datos provenientes del robot y ver "lo que ve el robot" [6], y RQt para la visualización y la gestión de procesos [11]. Incorpora una biblioteca de complementos para usar el simulador Gazebo que facilita, en gran medida, el diseño y la simulación [7]. Existen varias distribuciones independientes de simuladores y visualizadores robóticos como como los empleados en los trabajos [1, 3], una ventaja de ROS es que integra visualización, simulación y gestión de procesos en una misma distribución del software.

ROS contiene varias implementaciones de código abierto de funcionalidades y algoritmos robóticos básicos. Estas implementaciones están organizadas en paquetes los cuales se incluyen como parte de las distribuciones de ROS [7]. Viene instalado y configurado en varios robots entre los que existen numerosas plataformas móviles, una de ellas es la *turtlebot* con sus diferentes modelos (figura 1).

Figura 1

Plataforma de bajo costo turtlebot [11].

En la figura 1 de izquierda a derecha se observan varios modelos del robot móvil *Turtlebot*, a la extrema izquierda se encuentra el Turtlebot2 seguido por la serie de Turtlebot3 (*burger, waffle, waffle-pi*), los cuales son utilizados para la implementación de diferentes algoritmos y son consideradas de bajo costo en el mercado internacional [11]. El precio del robot Turtlebot 2 oscila alrededor de los 400 euros, mientras que el precio de la serie Turtlebot3 tiene un costo que oscila alrededor de los 1300 euros. Dichas plataformas están basadas en hardware modular que utilizan la microcomputadora raspberry-pi 3 conectada mediante el puerto usb a una tarjeta de microcontrolador OpenCr. Para la movilidad usan motores de la serie Dynamixel [11]. Aunque dichas plataformas son consideradas de bajo costo, su precio es bastante elevado para proyectos académicos en Cuba, razón fundamental por la cual se decide fabricar un modelo propio de robot móvil con propósitos académicos para la implementación de algoritmos sobre Sistema Operativo Robótico (ROS).

En el Grupo de Robótica y Mecatrónica de la Universidad Tecnológica de la Habana, Cujae, se requiere la construcción de una plataforma robótica que posea flexibilidad en cuanto a los componentes de hardware para la implementación y prueba de varios de los algoritmos que se desarrollan para la investigación. La plataforma también debe incluir modularidad en cuanto al software que estará dispuesto por capas desde la más baja, que es el firmware, pasando por la capa media que son los programas que se ejecutan en la microcomputadora integrada en la plataforma, hasta los programas que se ejecutan en la computadora remota, los cuales tienen una abstracción del hardware casi completa. Este artículo describe la arquitectura de hardware y software diseñada y creada entre estas tres capas, dígase la comunicación entre las mismas y los programas básicos que se ejecutan en cada una.

El aporte de este trabajo consiste en la descripción de la implementación modular distribuida en capas del Sistema Operativo Robótico (ROS) en una plataforma móvil desarrollada en Cuba a un costo más bajo que las existentes en el mercado internacional. Esta implementación incluye los algoritmos de más bajo nivel que se ejecutan en el microcontrolador del robot así como la comunicación con los niveles superiores del sistema.

El artículo está distribuido de la siguiente manera; en la sección 2 se analizan, desde un marco teórico, los aspectos generales de la navegación en la robótica móvil utilizando ROS para la solución de este problema. La sección 3 describe los subsistemas de hardware de la plataforma, incluida la detección y la actuación. Luego, la sección 4 contiene una explicación de la implementación del software desarrollado en ROS así como los resultados obtenidos en su aplicación en la plataforma diseñada.

2. – FUNDAMENTOS DE LA NAVEGACIÓN EN ROBÓTICA MÓVIL TERRESTRE

La navegación en robótica móvil se puede dividir según las funciones básicas que debe desempeñar una plataforma móvil autónoma. La planificación y el seguimiento de trayectorias, la evasión de obstáculos, la localización y el mapeo son requerimientos en la navegación y la exploración de los robots [2, 8]. Por lo general, el mapeo se hace simultáneamente con la localización, técnica conocida como SLAM (*Simultaneous Localization and Mapping*). Esta técnica constituye objeto de desarrollo futuro del presente proyecto.

2.1. – SLAM, LOCALIZACIÓN Y MAPEO

Los robots móviles han mostrado grandes progresos en términos de movilidad, percepción y la utilización de algoritmos avanzados, que le permiten realizar localizaciones en mapas mundiales bidimensionales así como entornos desconocidos [4]. Por ejemplo, los robots de servicio pueden proporcionar asistencia logística en la oficina mediante el transporte de materiales [9].

En el problema de localización y mapeo simultáneos es importante considerar aspectos como: la generación de mapas y los algoritmos de navegación que buscan alcanzar el destino con una planificación de ruta suave y evasión de obstáculos [10]. El mapa es una característica esencial para la navegación. Por lo tanto, es necesario crearlo e incorporarlo al robot. SLAM es un método para crear un mapa mientras el robot explora el espacio desconocido, detecta su entorno y estima su ubicación actual [11]. La construcción de mapas ha sido desarrollada utilizando acercamientos de fotometría y escáneres láseres como principales métodos. En el caso de la fotometría se obtiene una gran cantidad de datos que pueden ser analizados utilizando el procesamiento de imágenes para la construcción de un mapa. Por otra parte, la utilización de escáneres láseres permite la creación de nubes de partículas a mayores distancias [12].

Existe una variedad de algoritmos para el desarrollo de SLAM, algunos de los más utilizados son los basados en *Extended Kalman Filter* (EKF), SLAM basado en gráficos y SLAM de filtro de partículas [13]. En el caso de ROS se usa el paquete *gmapping* el cual implementa un algoritmo *Fast SLAM* que utiliza los datos de escaneo láser y la odometría para construir un mapa de cuadrícula de ocupación 2D.

Además, de construir mapas del terreno el robot también debe medir y estimar su pose (posición + orientación), esta habilidad es conocida como localización. El método de estimación de poses en interiores más utilizado para los robots de servicio es el *dead reckoning*, que consiste en una estimación relativa de poses [11]. La cantidad de movimiento del robot se mide con la rotación de la rueda. Sin embargo, hay un error entre la distancia calculada con la rotación de la rueda y la distancia real de desplazamiento. En varios casos, la información del sensor de unidad de medida inercial (IMU) se puede utilizar para reducir el error en la posición, compensando el error de posición y la orientación entre el valor calculado y el valor real [14].

2.2. – PLANIFICACIÓN Y SEGUIMIENTO DE TRAYECTORIAS

La planificación de trayectorias consiste en encontrar el mejor camino de un punto de partida hacia un punto de llegada. En [3], esta planificación se realiza sobre un mapa global teniendo en cuenta las distintas posiciones que debe seguir un robot para alcanzar el objetivo sin considerar la velocidad y la aceleración del mismo, lo cual permite una mayor modularidad del sistema. El seguimiento de la trayectoria se encarga de efectuar un control sobre la velocidad y/o la aceleración del robot de tal manera que siga la trayectoria previamente trazada por el planificador, esta tiene en cuenta la dinámica y la cinemática del robot [2].

Para conocer si existen obstáculos se requieren sensores, por ejemplo, de distancia y de visión. Los sensores de visión más populares incluyen cámaras estéreo, cámaras omnidireccionales, y recientemente, RealSense, Kinect, Xtion, que se utilizan ampliamente como cámaras de profundidad, y se usan para identificar obstáculos [6, 10, 11]. El sensor de distancia utilizado en este trabajo es de tipo ultrasónico.

3. - SUBSISTEMAS DEL ROBOT

En el sistema digital propuesto en el trabajo se utiliza una computadora de placa única (*Single Board Computer SBC*) Raspberry Pi para la gestión de procesos de alto nivel, como por ejemplo la planificación de trayectorias. Una cámara *raspicam* está conectada a la Raspberry Pi. El SBC está conectado por USB con un microcontrolador Arduino MEGA, el cual

implementa las tareas de bajo nivel como lectura de sensores y el control sobre los actuadores. Además, se ha diseñado un sistema sonar, para la detección y evasión de objetos, conformado por un sensor ultrasónico y un servomotor. La fuente de alimentación es un banco de energía solar que puede proporcionar 2A y 5V. Para la tracción, el robot tiene dos motores de CC y un módulo L298 para el control de los motores, este subsistema presenta una fuente de alimentación independiente que consta de tres baterías AA. Una descripción gráfica del sistema puede observarse en la figura 2.

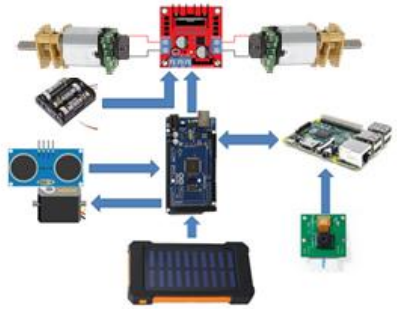


Figura 2

Componentes de hardware del diseño propuesto para la plataforma robótica.

3.1.- SISTEMA SONAR

El módulo de rango ultrasónico HC-SR04 permite medir la distancia a los obstáculos en un rango de 2 a 400 cm, la precisión de rango puede alcanzar hasta 3 cm. El módulo incluye un transmisor ultrasónico, un receptor y un circuito de control. El circuito se montó en una plataforma giratoria utilizando un servomotor para cubrir 180 grados de medición de distancia, como se muestra en la figura 3.

El servo utilizado para el sistema de sonda fue el SG 90, que es liviano con un par de salida relativamente alto. Puede ser controlado por cualquier biblioteca de software, incluido "servo.h", que es compatible con la placa Arduino utilizada. El voltaje de funcionamiento está entre 4.8 y 5.2 V y tiene un par de parada de 1.8 kgf.cm.

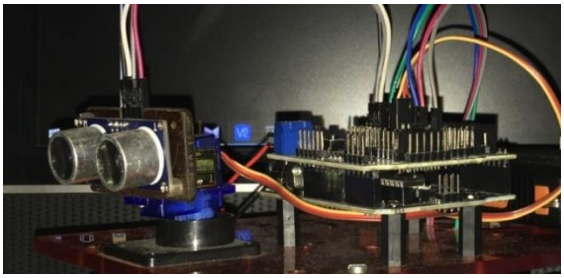


Figura 3

Sistema sonar.

El sistema sonar será utilizado para la detección de obstáculos con respecto al robot en un ángulo de 180°, de esta manera se pueden evadir dichos obstáculos evitando que el robot colisione.

3.2.- CÁMARA

La cámara Raspberry Pi, mostrada en la figura 4, es una placa de cámara lanzada por la Fundación Raspberry Pi. Esta cámara es un complemento adicional de alta calidad de 8 megapíxeles con sensor de imagen Soy IMX219 diseñado para Raspberry Pi. Es capaz de captar imágenes estáticas de 3280 x 2464 píxeles y admite video 1080p30, 720p60 y 640 x 480 p 90. Se conecta a la Raspberry Pi a través de uno de los pequeños conectores en la superficie superior de la placa y utiliza la interfaz CSi dedicada, diseñada especialmente para interactuar con las cámaras.

Figura 4

Cámara Raspberry.

La cámara se utiliza para observar desde la computadora remota el campo visual del robot. Además, se usa en la implementación de algoritmos de reconocimiento facial y reconocimiento de ejes, cuyos resultados son mostrados en la sección 4.3. Estos algoritmos de procesamiento de imágenes son incorporados con el objetivo de promover investigaciones futuras sobre la plataforma, por ejemplo, realizar seguimiento de usuarios mediante reconocimiento facial, o SLAM visual.

3.3.- Sistema de tracción

Para el sistema de tracción se utilizan dos motores acoplados a ruedas controlados por el *driver* L298n, dicho *driver* es un puente H doble utilizado para el control del sentido de rotación. El motor utilizado es de corriente directa con un voltaje operacional de 5V a 12V. El actuador posee engranaje y un codificador Hall para medir el desplazamiento y la velocidad angular del eje (Figura 5). El modelo de motor consta de un disco magnético de seis polos acoplado en su eje y dos sensores de efectos Hall.

Figura 5

Motor de corriente directa con codificador Hall.

3.4.- PLACA DE MICROCONTROLADOR ATMEL

El diseño utiliza una placa de desarrollo que se basa en el microcontrolador Arduino MEGA. La placa tiene las especificaciones mostradas en la tabla 1.

Tabla 1
Características de Arduino MEGA.

Elementos	Características
Microcontrolador	ATmega328
Voltaje operacional	5V
Fuente de alimentación	6-20V
Entradas digitales	14 (seis tienen PWM)
Entradas analógicas	6
SRAM	2 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz

3.5.- SBC RASPBERRY PI

El hardware de la plataforma robótica necesita una unidad central de procesamiento para ejecutar el sistema operativo robótico y los algoritmos de alto nivel relacionados con el control del robot. Se utiliza una Raspberry Pi para estas tareas y se muestra en la Figura 6.

Figura 6

SBC Raspberry Pi.

Raspberry Pi es una computadora de placa única de bajo costo desarrollada en Reino Unido por la fundación Raspberry Pi. En todo el mundo, Raspberry Pi se utiliza para aprender programación, construir prototipos e incluso aplicaciones industriales. Raspberry Pi es compatible con sistemas operativos como Linux y Windows 10, también proporciona pines GPIO que permiten la interfaz con hardware externo. El modelo utilizado en el proyecto fue el modelo B + de Raspberry Pi 3 y algunas características se enumeran en la Tabla 2.

Tabla 2
Características de Raspberry Pi 3 modelo B+.

Elementos	Características
CPU	Broadcom BCM2837B0, Cortex-A53 64-bit 64-bit SoC @ 1.4 GHz
Memoria	1 GB LPDDR2 SDRAM
Conectividad	2.4 GHz and 5 GHz IEEE 802.11 .b/g/n/ac wireless LAN, Bluetooth 4.2, BLE 4 x USB 2.0 ports
Pines	40 GPIO
Fuente de alimentación	5 V/ 2.5 A DC, conector micro-USB
Tarjeta SD	Puerto micro-SD para cargar el sistema operativo
CPU	Broadcom BCM2837B0, Cortex-A53 64-bit 64-bit SoC @ 1.4 GHz

Tabla 3
Costos de los materiales.

Elemento	Costo USD
Raspberry pi model3 B+	40.95
Arduino Mega	15.99
Cámara de Raspberry pi	29.51
Servomotor SG 90	2.20
Driver L298	9.08
Sensor de distancia ultrasónico HC-SR04	4.95
2 motores DC con *encoders*	45.16
banco de energía	28.99
3 baterías AA	2.25
Cables de conexión para prototipos	6.99
Placa acrílico	8.99
Total	159.06

Los costos de los componentes analizados en la tabla anterior son extraídos de un análisis del mercado Amazon en México, que es en donde fueron comprados los componentes. Un análisis de otras ofertas fue realizado en otros mercados como *tmall, alibaba, taobao*, encontrándose una variación en los costos de ±15% con respecto a los presentados en la tabla 3. A continuación se presentan los costos que representan las horas de trabajo invertidas por los autores del trabajo

Tabla 4
Salario de los participantes del proyecto

Participantes	Días dedicados a la investigación	Salario diario	Total (CUP)
Carlos G. Miguélez Machado	90	37.29	3356.1
Ivón O. Benítez González	30	98.12	2943.6
Valery Moreno Vega	30	98.12	2943.6
Alex Manuel Rivera Rivera	90	4.16	374.4
Total	-	-	9617.7

Utilizando la tasa de cambio del banco cubano del 17 de septiembre del año 2020 el valor total calculado de los salarios es de 388.71 USD. Sumando un total de 547.77 USD. A continuación se listan los costos de varias plataformas similares.

Tabla 5
Costos de varias plataformas similares

Robot	Costo USD	Investigación o educación	Código abierto
Turtlebot 3 burger	1325	Investigación	si
E-puck	889	ambas	si
Khapera IV	4445	ambas	no
Rice r-one	254	ambas	no

4. – ARQUITECTURA DE SOFTWARE

El desarrollo del software se ha realizado en ROS. La Figura 7 muestra el sistema implementado usando una computadora remota conectada con la Raspberry Pi del robot a través de Wi-Fi. La Raspberry Pi y el microcontrolador están conectados mediante el puerto USB y usando el protocolo *rosserial* el cual que permite conectar diferentes módulos de hardware utilizando conexión serial. El tipo de datos es definido en cada dispositivo y se transmiten usando las funciones de las librerías de *rosserial* la cual integra un nodo de Python en el servidor que gestiona la transmisión de información y le da visibilidad a ROS de los nodos creados en los diferentes dispositivos [13]. De esta manera se puede desarrollar el *firmware* deseado y hacerlo visible a ROS en forma de nodos. La implementación de dicha estructura provee modularidad y un alto nivel de independencia entre las capas de *firmware* y software.

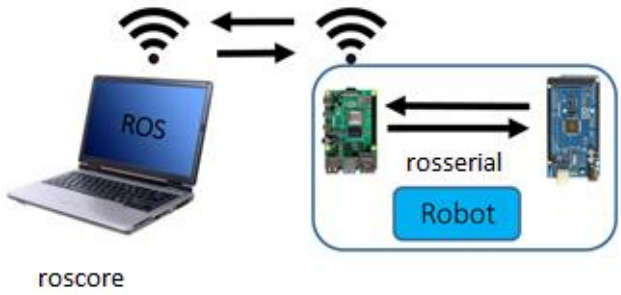


Figura 7

Sistema con ROS.

Para el desarrollo de este trabajo se utiliza la versión *Kinetic* de ROS instalada tanto en la Raspberry Pi como en la computadora remota. El sistema operativo instalado en la computadora remota es Ubuntu 16.04 (Xenial) y el sistema operativo instalado en Raspberry Pi es Ubuntu Mate (16.04). Las bibliotecas *ROS-lib* para Arduino se utilizan para ejecutar los nodos de ROS en Arduino y la comunicación con la plataforma se realiza utilizando el protocolo *rosserial*. El paquete *rosserial* está instalado en la Raspberry Pi.

Generalmente, la arquitectura ROS se implementa utilizando nodos que son las unidades ejecutables básicas en ROS. Cada nodo se comunica con otros nodos mediante mensajes [10]. La comunicación del mensaje se divide en tres tipos diferentes (*topics*, servicios y acciones). Los *topics* son formas de comunicación unidireccionales y continuas entre nodos. Una

comunicación usando *topics* está compuesta por un nodo publicador (emisor) y uno o varios nodos suscriptores. Los servicios son formas de comunicación no continua y bidireccional, los nodos que establecen este tipo de comunicación son el servidor y el/los clientes del servicio. Las acciones se basan en la emisión de una solicitud por parte del cliente que el servidor se tarda un tiempo relativamente grande en completar, mientras la tarea encomendada está en proceso de ejecución el servidor envía una retroalimentación al cliente del estado de la tarea.

Los algoritmos en este trabajo son implementados en forma de nodos que radican en las diferentes capas del sistema diseñado. En la capa más baja que es el firmware se programan los nodos que controlan los actuadores y leen la información de los sensores. En la capa media, en la Raspberry Pi, se posicionan los nodos que implementan algoritmos de más complejidad como son generadores de mapas y controles de manejo del robot. En la computadora remota se implementan nodos de mayor complejidad computacional, estos serán los que tengan mayor abstracción de la capa de hardware que en este caso es un nodo de procesamiento de imágenes. El grafo de la figura 8 ilustra los nodos (óvalos) implementados en el Arduino (azules), en la Raspberry Pi (verdes) y en la computadora (blanco). Los rectángulos constituyen los *topics* que publica y suscribe cada nodo.

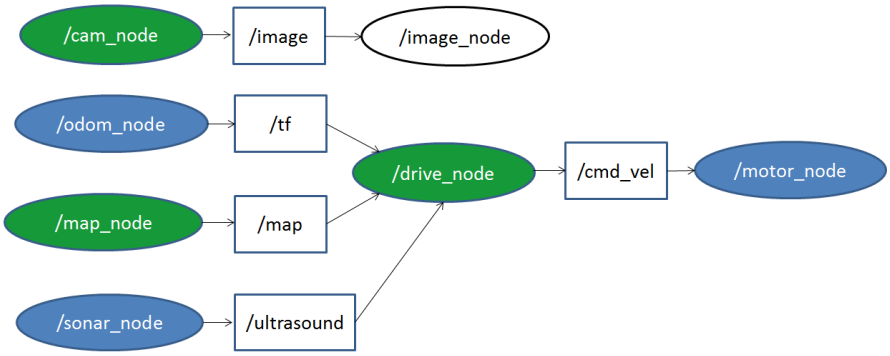


Figura 8

Grafo de nodos y topics ROS.

4.1.- NODOS ROS IMPLEMENTADOS EN EL MICROCONTROLADOR

En el Arduino se encuentran desarrollados tres nodos. El nodo ("sonar_node") que publica continuamente los datos leídos por el sensor ultrasónico en el *topic* ("/ultrasound"). El ("/motor_node") está implementado para el control de motores, este nodo suscribe al *topic* ("cmd_vel") el cual contiene la velocidad angular (w) y lineal (v) que debe seguir el robot. Este nodo calcula la velocidad que debe aplicarse a cada rueda dada una velocidad angular y una velocidad lineal del robot, utilizando el modelo matemático dado por las siguientes ecuaciones:

$$V_r = 2v + \frac{wL}{2R}$$
$$V_l = 2v - \frac{wL}{2R} \quad (1)$$

Donde V_r y V_l son las velocidades tangenciales de la rueda derecha e izquierda respectivamente, R es el radio de las ruedas y L es la distancia entre las mismas.

El tercer nodo implementado en Arduino es ("odom_node") que calcula la pose actual del robot (x, y, Θ) utilizando el desplazamiento relativo de las ruedas respecto a la posición anterior. El modelo se deduce a partir de la figura 9.

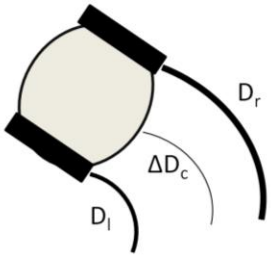


Figura 9

Modelo de odometría de un robot diferencial.

En la figura 9 se define D_l como el desplazamiento relativo de la rueda izquierda del robot, D_r es el desplazamiento relativo de la rueda derecha y ΔD_c constituye el desplazamiento relativo respecto a la posición anterior del punto central entre ambas ruedas. Según esta relación se define la expresión (2):

$$\Delta D_c = \frac{D_l + D_r}{2} \tag{2}$$

Si se considera $\Delta\theta$ como pequeñas variaciones en el ángulo de orientación del robot y L es la distancia entre las ruedas del robot, es posible establecer la siguiente ecuación:

$$\Delta\theta = \frac{D_r - D_l}{L} \tag{3}$$

Luego el modelo de odometría viene dado por la expresión (4):

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) + \Delta D_c(k)\cos(\theta(k)) \\ y(k) + \Delta D_c(k)\sin(\theta(k)) \\ \theta(k) + \Delta\theta(k) \end{bmatrix} \tag{4}$$

Donde X, Y son las variables que definen la posición en el eje coordenado y Θ el ángulo de orientación. El desarrollo matemático para obtener las expresiones (2)-(4) se encuentra en [7].

4.2.- NODOS ROS IMPLEMENTADOS EN RASPBERRY PI

En la Raspberry Pi se desarrollan tres nodos. El ("/drive_node") que se publica en el *topic* ("/cmd_vel"). Este nodo suscribe al *topic* ("/ultrasound") del ("/sonar_node") y al *topic* ("/map") del ("/map_node"). El ("/drive_node") utiliza la información de los sensores ultrasónicos y la información del mapa para generar la velocidad angular y lineal del robot.

El ("/map_node") genera el mapa mostrado en la figura 10. Para la visualización de este mapa se utiliza la herramienta RViz de ROS la cual brinda la posibilidad de observar en tiempo real la información de los *topics* que se encuentran en ejecución. El mapa generado es de una habitación de aproximadamente 9 m^2, la figura 10 tiene una escala de 1m por división. Este mapa tiene una resolución de 0.25 m. El mapa fue generado mediante código creando un arreglo bidimensional de valores, donde las casillas oscuras están ocupadas por obstáculos mientras que las claras son espacios libres por las cuales se planea la trayectoria del robot. En la figura 10 también se puede apreciar el punto (0; 0) del plano coordenado donde se establece el mapa, definiendo el eje x (rojo) y el eje y (verde).

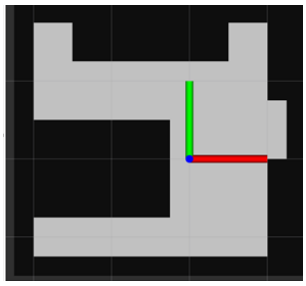


Figura 10

Mapa de una habitación generado por el algoritmo gmapping de ROS.

El ("/drive_node") implementa el algoritmo de Dijkstra para la planificación de trayectorias. Dado un punto de partida y un punto de llegada el algoritmo calcula la ruta más corta entre dichos puntos evitando los obstáculos que se encuentran en el mapa. La implementación de este algoritmo en el nodo considera el diámetro del robot manteniendo una distancia prudencial de los obstáculos para evitar la colisión de las extremidades del robot con los mismos. La trayectoria generada por este algoritmo consiste en un arreglo de puntos (x; y) por los cuales deberá pasar el robot. Un ejemplo de trayectoria generada se visualiza en la figura 11, donde el punto de partida es en (-1.5; -1.0) y el punto de llegada es (-2.0; 1.5).

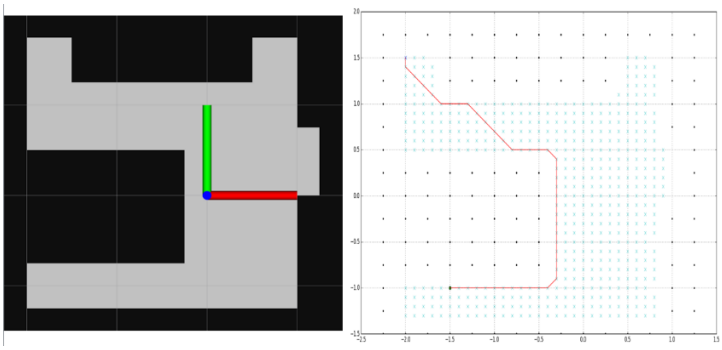


Figura 11

Mapa de una habitación a la izquierda. Trayectoria genearda entre dos puntos a la derecha

El nodo ("/drive_node") también implementa un control PID sobre la velocidad angular (w) y mantiene constante la velocidad lineal (v) con un valor de 0.3 m/s. El control PID sobre (w) es descrito en la expresión (5). Este control toma como referencia las metas locales de la trayectoria generada y como variable medida del proceso la odometría del robot ("/tf"). Generándose de esta manera un seguimiento de trayectoria.

$$W = Kpa(\alpha) + Kia\int(\alpha)dt + Kda\frac{d(\alpha)}{dt} \quad (5)$$

Donde α es el error en la orientación del robot con respecto al objetivo. Los valores de Kpa, Kia, Kda se corresponden con las constantes proporcional, integral y derivativa del control de velocidad angular, los cuales se muestran en la tabla 3.

Tabla 6

Valores de las constantes proporcional, integral y derivativa.

Kpa	Kia	Kpa
6.0	0.00001	2.5

En el seguimiento de la trayectoria se fijan las metas locales (ver figura 12), marcadas sobre el recorrido planeado con el algoritmo de Dijkstra. Las nueve metas locales se corresponden con los siguientes pares coordenados (-0.4; -1.0), (-0.3; -0.9), (-0.3; 0.4), (-0.4; 0.5), (-0.8; 0.5), (-1.3; 1.0), (-1.1; 1.0), (-2.0; -1.4) y finalmente (-2.0; 1.5). En este caso el punto de partida del robot es (-1.5; -1.0) y el punto de llegada es (-2.0; 1.5), que coincide con la meta global.

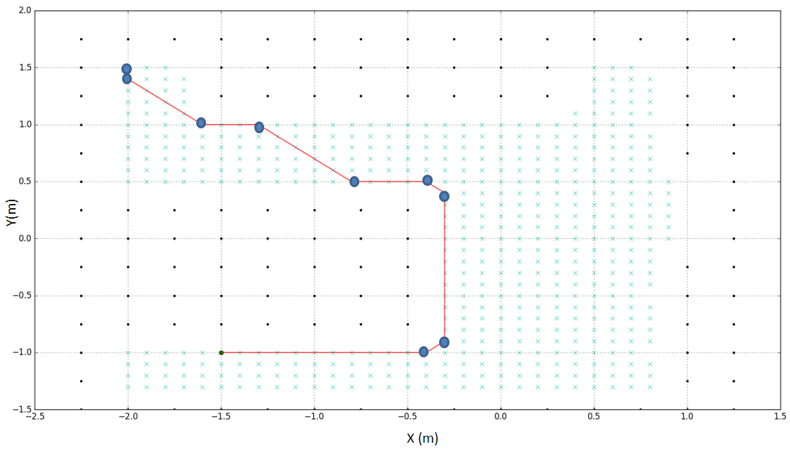


Figura 12

Ampliación de la trayectoria generada con las metas locales a seguir por el robot

En la figura 13 se muestra en forma de barra azul claro las nueve metas locales por las que el robot debe navegar según la planificación definida previamente. Además, se observa como los errores en la posición (rho) y en la orientación (alpha) tienden a cero por la acción de los controles sobre las velocidades lineal y angular. A partir de las dimensiones del espacio de trabajo se permite un margen de error en la posición de 0.1 m. Por ejemplo, al arribar a la meta local 1, el error en la posición es de 0.09 m como puede apreciarse en las figuras 13 y 14, y durante el trayecto no se observa cambio en la orientación, manteniéndose en 0 el valor de alpha. Un cambio significativo del error en la posición se observa en la transacción de la meta local 2 a la meta local 3, donde existe un cambio en la orientación y al estabilizarse en 0, el error en la posición tiende a cero hasta alcanzar la meta local 3

.

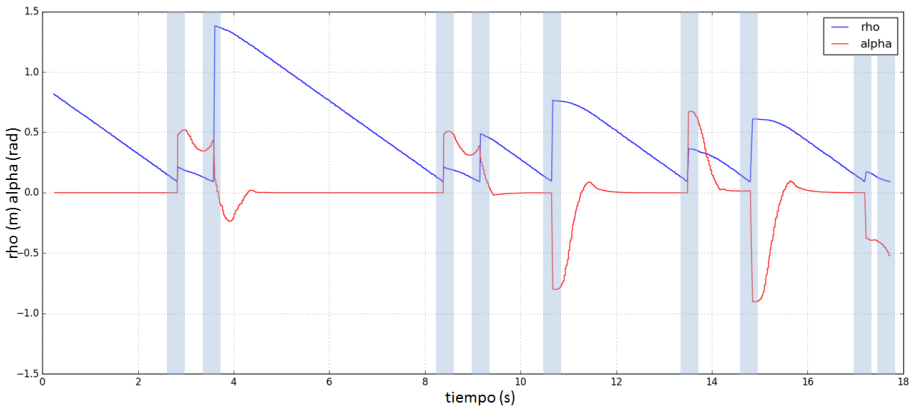


Figura 13

Gráfica del error en la posición (rho) y el error en la orientación (alpha)

En la figura 14 se muestran los valores reales (x; y) de la posición del robot en la trayectoria planificada así como los valores definidos en las metas locales que se corresponde con la referencia a alcanzar. En esta figura se observa como en todo el recorrido se sigue la trayectoria en ambos ejes. Es importante destacar que si el robot arriba a una meta local, la referencia cambia automáticamente a la siguiente.

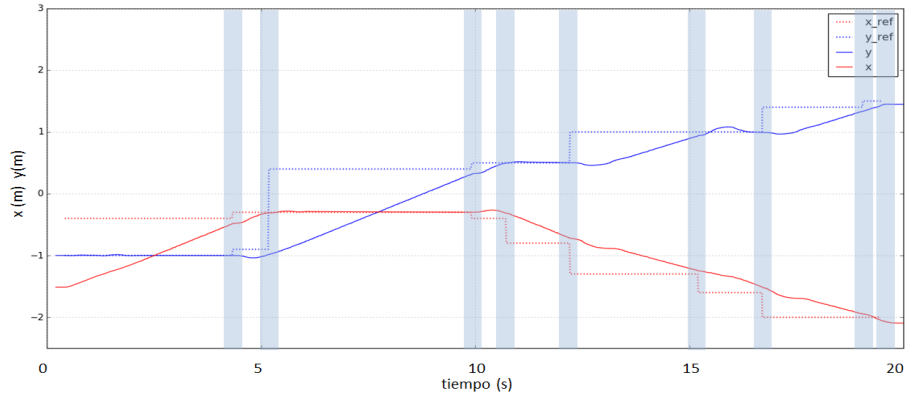


Figura 14

Gráfica de posición (x; y)

El nodo de cámara ("/cam_node") también se implementa en la Raspberry Pi y publica el *topic* ("/image"), este *topic* es recibido por la computadora remota donde se realiza el procesamiento de la imagen como se muestra en la figura 15.

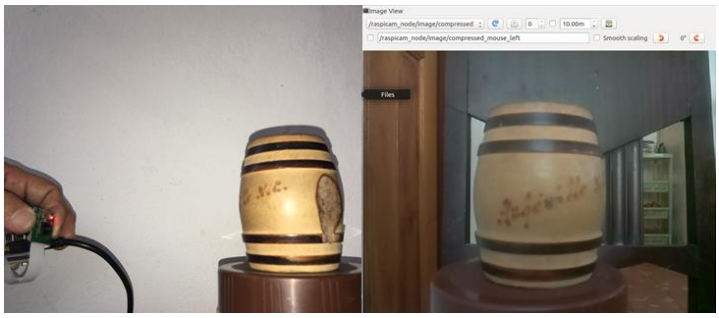


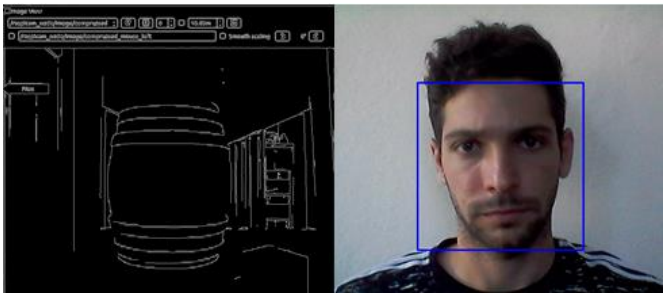
Figura 15

Cámara Raspberry a la izquierda, imagen tomada por la cámara Raspberry a la derecha.

4.3.- NODOS ROS IMPLEMENTADOS EN LA COMPUTADORA REMOTA

Además, de los tres nodos que se ejecutan en la Raspberry Pi en la computadora remota se está ejecutando el nodo maestro (*roscore*). Este nodo no intercambia directamente información con ningún nodo en particular, es por ello que no se representa en el grafo mostrado en la figura 8. Internamente *roscore* registra todos los nombres de los nodos así como los nombres mensajes que se publican, basado en el sistema mostrado en la figura 7.

Por ejemplo, cuando un nodo suscribe un mensaje en específico este primero verifica que se encuentre publicado en el *roscore* y de esta manera se efectúa la comunicación entre nodos. Específicamente, el ("/image_node") que suscribe el *topic* ("/image") como se observa en el grafo de la figura 8, implementa los algoritmos necesarios para la detección de rostros y, además, la detección de ejes como se observa en la figura 16. Este nodo se implementa en la computadora remota pues se requiere por el usuario ver lo que el robot está visualizando con la cámara en tiempo real y realizar el correspondiente procesamiento de imágenes, el cual demanda un costo computacional superior al que puede garantizarse con la Raspberry Pi.



A **B**

Figura 16

Procesamiento de imágenes en la computadora remota. A-Detección de ejes. B Detección de rostros.

5. - CONCLUSIONES

En este trabajo se diseña una plataforma con las características requeridas para desarrollar e implementar algoritmos usados en las aplicaciones de la robótica móvil. Dichos algoritmos pueden estar encaminados al seguimiento y planificación de trayectorias, localización y mapeo simultáneo o visión artificial aplicada a la robótica. En la plataforma diseñada se propone la utilización de una arquitectura de software implementada en ROS para la navegación, así como la visualización y procesamiento de las imágenes obtenidas por la cámara instalada. La arquitectura planteada es modular y versátil permitiendo la futura implementación de nuevos programas para la realización de diferentes tareas en el robot.

Conocidos los puntos de partida y de llegada y el mapa del espacio de trabajo se implementó el algoritmo de Dijkstra para la planificación de trayectorias del robot, lográndose esta tarea en 7.75 s según las características de la Raspberry Pi.

Con el desarrollo del control de la velocidad lineal y angular de la plataforma se logró el seguimiento de la trayectoria previamente planeada con un error en la posición de 0.1 m.

Se demuestra el éxito de ROS en una plataforma diseñada en Cuba para las tareas de visualización e interconexión de procesos, así como para la generación de mapas. Con la arquitectura de software distribuida se incrementa la velocidad de desarrollo por el uso de módulos independientes.

Las futuras investigaciones están encaminadas a la implementación de seguimiento de trayectorias previamente planificadas en mapas globales de entornos no estructurados. Dichos algoritmos podrían ser implementados sobre la plataforma que se propuso en el presente artículo.

AGRADECIMIENTOS

Agradecimientos a BUZZER de Guadalajara, México por facilitar varios de los componentes usados en la construcción de la plataforma desarrollada.

REFERENCIAS

1- Wu Q, Chen Z, Wang L, Lin H, Jiang Z, Li S et al. Real-time dynamic path planning of mobile robots: a novel hybrid heuristic optimization algorithm. Sensors. 2020; 20 (1): 19,188.
2- Rubio F, Valero F, Llopis-Albert C. A Review of mobile robots: Concepts, methods, theorical framework, and applications. International Journal of Advanced Robotic Systems. 2019; 16(2).
3- Torres, M., Moreno V. Diseño y simulación de un robot móvil recolector de objetos. Revista Ingeniería Electrónica, Automática y Comunicaciones, Rielac. 2009; 30(3): 16-20.
4- Chen J, Cho Y. Detection of Damaged Infrastructure on Disaster Sites using Mobile Robots. 16th International Conference on Ubiquitous Robots (UR). Jeju, Korea; (2019). p. 24-27.
5- Aitken JM, Veres SM, Judge M. Adaptation of system configuration under the robot operating system. IFAC Proceedings Volumes. 2014; 47(3): 4484-4492.
6- Quigley M, Gerkey B, William D. Programming Robots with ROS A Practical Introduction to the Robot Operating System. 1st ed. Boston: USA; (2015).
7- Pektas O. Design of an autonomous mobile robot based on ROS. International Artificial Intelligence and Data Processing Symposium (IDAP). Malatya; (2017). p. 1-5.
8- Guang-Zhong Y, Bellingham J, Dupont P, Fischer P, Floridi L et al. The grand challenges of Science Robotic. Science Robotics. 2018; 3(14).
9- Bekey G, Kumar V, Sanderson A, Wilcox B, Zheng Y. International assessment of research and development in robotics. World Technology Evaluation Center. Baltimore, Maryland; (2006).
10- Siegwart R, Nourbakhsh IR, Scaramuzza D. Introduction to Autonomous Mobile Robots (Intelligent robotics and autonomous agents). 2nd ed. Massachusetts: USA; 2011.
11- Pyo Y, Cho H, Jung R, Lim T. ROS Robot Programming From the basic concept to practical programming and robot. 1sd ed. Seoul: Republic of Korea; (2017).
12- Kim P, Chen J, Cho Y. SLAM-driven Robotic Mapping and Registration of 3D Point Clouds. Automation in Construction.(2018); 89(4): 38-48
13- Zubrycki I. Introducing modern robotics with Ros and Arduino. Journal of Automation Mobile Robotics and Intelligent Systems. 2014; 8(1): 69-75.
14- Alatise M, Hancke G. A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods. IEEE Access. (2020); 8(2): 39830-39846.

CONFLICTO DE INTERESES

Ninguno de los autores manifestó la existencia de posibles conflictos de intereses que debieran ser declarados en relación con este artículo

CONTRIBUCIÓN DE LOS AUTORES

Carlos Guillermo Miguélez Machado: contribución en el aspecto de la implementación y uso de Robotic Operating System (ROS) en una plataforma robótica de hardware libre así como de los aspectos de programación en las capas de Software. Redacción del borrador del artículo y de su versión final.

Ivón Oristela Benítez González: contribución importante en los aspectos de diseño del sistema de control implementado. Análisis de la revisión bibliográfica e interpretación de la misma. Revisión crítica del borrador del artículo y de la versión final.

Alex Manuel Rivera Rivera: contribución en los aspectos de diseño de hardware y utilización de los sensores en el robot. Contribución en la redacción del borrador del artículo y en el análisis bibliográfico.

Valery Moreno Vega: contribución en el diseño de la tarea planteada y en la utilización de herramientas de visualización como son los grafos mostrados. Revisión crítica de la versión final del artículo a publicar.

AUTORES

Carlos Guillermo Miguélez Machado, Ingeniero en Automática trabaja en la Universidad Tecnológica de La Habana José Antonio Echeverría, Cujae, La Habana, Cuba. Correo electrónico cmiguelezmachado@gmail.com. https://orcid.org/0000-0002-1455-3218. Sus intereses de investigación radican en los sistemas digitales y la inteligencia artificial aplicada a la robótica móvil.

Ivón Oristela Benítez González, Ingeniero en Automática (2005), Master en Informática Industrial y Automatización (2010), Doctor en Ciencias Técnicas (2017), profesora titular en la Universidad Tecnológica de La Habana José Antonio Echeverría (Cujae). Correo electrónico, novi@automatica.cujae.edu.cu. https://orcid.org/0000-0002-1096-305X. Sus temas principales de investigación son el control avanzado de procesos, robótica educativa e industrial, y la automatización de procesos.

Alex Manuel Rivera Rivera, Estudiante de 2do año de Ingeniería en Automática en la Universidad Tecnológica de La Habana José Antonio Echeverría (Cujae). Correo electrónico, alexmanuelrivera@gmail.com. https://orcid.org/0000-0002-2054-9645. Sus temas principales de investigación son la robótica móvil y la robótica educativa.

Valery Moreno Vega, Ingeniero en Máquinas Computadoras (1993), Master en Informática Aplicada (1995), Doctor en Ciencias Técnicas (2004), profesor asociado del Dpto. de Mecatrónica en el Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM). Correo electrónico, valery.moreno@gmail.com. https://orcid.org/0000-0001-5182-4059. Sus temas principales de investigación son el desarrollo de aplicaciones para IoT, robótica aplicada, y el desarrollo de sistemas para dispositivos móviles y entornos Web destinados a recolección y procesamiento de metadatos georeferenciados.