Robot móvil autónomo para crear mapas 3D en un ambiente acotado

A. Caverzasi, F. Saravia, O. Micolini, L. Mathé, *Members, IEEE.*

***Abstract*—In this work a mobile robot capable of producing a rough map of an interior of a building is presented. The main aim of the present work is to build a mobile robot, with the ability of self-localization, navigation, and mapping. To accomplish this several technologies were used, like: Arduino development board, the Robot Operating System, and Kinect sensor. The constructed robot offers a wide field of applications; besides mapping, other interesting alternatives are the exploration of dangerous zones, telepresence and education.**

***Keywords*— Mobile autonomous robot, Mapping, Localization, Navigation, Control.**

I. Introducción

Los últimos años han sido de gran importancia para el desarrollo y aplicación de la robótica industrial. De estar acotada al ámbito industrial, la robótica ha pasado a expandirse hacia áreas nunca antes pensadas –médica, doméstica, militar, comercial, entre otras– y a tener aplicaciones de las más diversas. En los últimos 5 años el contexto se vio potenciado por la creación del Robot Operating System que facilita un desarrollo organizado de código reusable y abierto, con herramientas necesarias para cualquier proyecto de robótica. Adicionalmente, la aparición del sensor Kinect, de distribución masiva y bajo costo, desencadenó la construcción de una gran cantidad de robots capaces de "percibir" tres dimensiones espaciales, identificar personas y crear mapas.

El presente trabajo describe el desarrollo de un robot móvil, con la capacidad de auto localizarse, navegar, y generar mapas del lugar por el que se desplaza.

A. *Objetivo General:*

- Explorar y construir un mapa aproximado de un espacio cerrado y acotado, que posee obstáculos pero permiten la circulación.

B. *Objetivos Específicos:*

- Construir un dispositivo móvil que se desplace dentro de la habitación de forma autónoma, con el objetivo de obtener mediciones de la misma y evitar los obstáculos que se interpongan.
- Seleccionar un sensor o un conjunto de sensores que permitan la correcta medición del lugar.
- Determinar sobre que plataforma de hardware y software se va a montar el sistema.

Además cabe mencionar que el desarrollo fue llevado a cabo bajo un requerimiento que establecía que el robot debía ser controlado de forma autónoma y remota, teniendo en cuenta su potencial uso en aplicaciones industriales.

II. Estado del arte

A. *Robots móviles. Clasificación*

Los robots móviles se clasifican según el movimiento que son capaces de realizar; existen dos tipos: *holonómicos* y *no-holonómicos*. En los casos en los que es posible controlar todos los grados de libertad, el robot se denomina holonómico. En caso contrario es no-holonómico. Dentro de la última clasificación existen diferentes modelos cinemáticos como *Diferencial* (dos ruedas cada una con un motor independiente para la tracción) o *Ackerman* (contiene como mínimo un motor para la tracción y uno para controlar la dirección de giro).

B. *Robot Operating System*

Provee librerías y herramientas que brindan a los desarrolladores todo lo necesario para crear aplicaciones relacionadas a la robótica [1]. No es un sistema operativo en el sentido tradicional de administración y planificación de procesos, sino que provee una capa de comunicaciones estructuradas por encima del sistema operativo anfitrión (ej. Linux Ubuntu, Windows, etc). Si bien provee la funcionalidad básica de cualquier sistema operativo tradicional como: abstracción del hardware, acceso a dispositivos de bajo nivel, implementación de funcionalidad de uso común, comunicación entre procesos, mantenimiento de paquetes, etc., además proporciona librerías, y todo tipo de herramientas (visualizadores, GUI's, etc.) para el desarrollo de robots y sus aplicaciones (ej. Rqt, tf, rviz [2]).

ROS organiza el software según una arquitectura de grafos en la que en el más bajo nivel se tienen los 'nodes' (porción de código ejecutable) y éstos se comunican entre sí a través de 'topics' (interfaz de ROS para el paso de mensajes entre nodos). A su vez los nodos se agrupan en 'packages' (colecciones mínimas de código reusable), y éstos en 'stacks' que permiten compartir el código fácilmente [3].

C. *Odometría*

Técnica que permite estimar el cambio de posición de un sistema, a partir del uso de datos provenientes de sensores como encoders, magnetómetros, giróscopos, etc. Se utiliza a menudo para calcular la posición relativa de robots móviles. Bajo estos conceptos se hace posible la implementación de los requerimientos vinculados a la localización y el control del robot. Por otro lado, la odometría visual consiste en el proceso de estimación de la posición y la orientación del robot a través

del análisis de imágenes tomadas con cámaras asociadas. En otras palabras, se estudia el efecto del movimiento sobre las imágenes producidas por las cámaras a bordo del robot. Algunos factores que podrían dificultar el proceso son la falta de iluminación, escenas con poca textura y escenas con muchos objetos en movimiento. Por ello es frecuente el uso de cámaras de profundidad RGBD para realizar la odometría visual en robots móviles. Dicho concepto permite la implementación de los requerimientos vinculados al mapeo y visualización.

III. Características del robot

En el presente trabajo se construyó un robot no-holonómico, con un modelo cinemático diferencial. El mismo se compone de las siguientes partes: en el más bajo nivel, una placa electrónica para el control de los motores y sensores de odometría. Luego un microcontrolador para la programación de la funcionalidad de bajo nivel como el control y la localización del robot, y finalmente una computadora a bordo para la navegación, visualización y mapeo.

Numerosas alternativas fueron analizadas en cuanto a sensores, microcontroladores, y sistemas operativos como así también los frameworks de desarrollo que poseen. A continuación se describen las más apropiadas para el proyecto.

A. Localización del robot. Posición

Para el cálculo y control de trayectoria, se requiere medir distancias recorridas para estimar la posición. Se consideraron: encoders rotacionales, sensores de ultrasonido, sensores infrarrojos, y sensor Kinect. Se optó por utilizar encoders, dado que vienen integrados junto a los motores utilizados para efectuar el movimiento del robot. Pero además, como complemento a los encoders, se utilizó la cámara de profundidad de Kinect RGBD, ya que ésta fue usada para la creación de mapas del entorno.

B. Localización del robot. Orientación

Se consideraron: magnetómetro HMC5883L, giróscopo MPU 6050 (GY-521), y sensor Kinect. Dado que el magnetómetro es susceptible a interferencias magnéticas, se optó por el giróscopo integrado en el módulo MPU 6050.

C. Placa de desarrollo. Microcontrolador

Se analizaron: NXP LPC1343 Board y Arduino Mega2560. Luego de experimentar sobre ambas, se eligió Arduino debido a su avanzado desarrollo tanto en hardware como software (posee una importante cantidad de módulos de hardware y sensores, además de librerías de software para utilizarlos), y su activa comunidad de desarrolladores.

D. Sistema operativo

El desarrollo fue llevado a cabo sobre el Robot Operating System - ROS, que corre bajo Linux. Se utilizó Ubuntu 12.10.

En la Figura 1 se puede observar que el robot está compuesto de dos plataformas. En la parte inferior se encuentra el circuito electrónico, el microcontrolador, y las baterías que alimentan todo el sistema. En la parte superior se puede ver la computadora a bordo y el sensor Kinect para mapeo.

Figura 1. Robot movil. Su composición.

IV. Arquitectura del sistema

El robot posee una arquitectura modular, diseñada en función de los objetivos propuestos en el apartado I. La Figura 2 muestra la interacción de cada componente a través de sus interfaces, y luego se describe la responsabilidad de cada una.

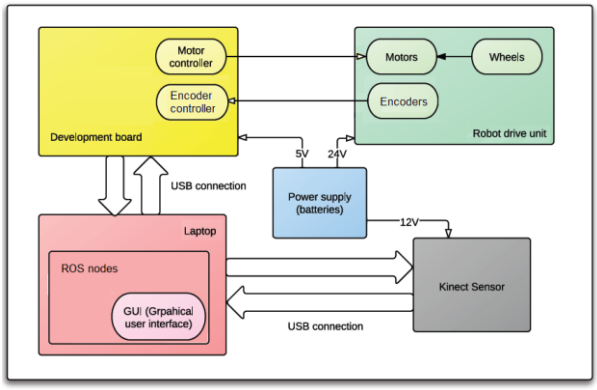

Figura 2. Diseño modular de la arquitectura del sistema.

- *Unidad de manejo del robot:* subsistema totalmente mecánico y de hardware. Contiene los motores para el movimiento de tracción del robot, una rueda frontal libre para su estabilización, y sensores usados para la localización.
- *Baterías*: subsistema de alimentación del robot.
- *Placa de desarrollo:* subsistema compuesto por la placa de desarrollo Arduino, y sensores asociados. Controla por software la unidad de manejo del robot, y realiza la localización y el control. Se comunica con la computadora a bordo del robot.
- *Computadora a bordo:* subsistema que ejecuta ROS sobre Ubuntu. Contiene los nodos robot (el que se comunica con Arduino para enviar comandos de movimiento), planner (planificación de trayectorias), y todos los relacionados a la visualización y mapeo con el sensor Kinect.
- *Sensor Kinect:* subsistema que envía streams de color y profundidad a la computadora a bordo para su procesamiento y visualización.

La Figura 3 muestra el diseño funcional del sistema. En ella se puede apreciar la interacción entre cada nodo. Los recuadros en línea punteada son nodos del sistema operativo ROS, mientras que los dos recuadros sombreados representan la computadora a bordo del robot (o maestro), y la computadora

remota (o esclavo) utilizada para el modo de operación de telepresencia que se explica a continuación.

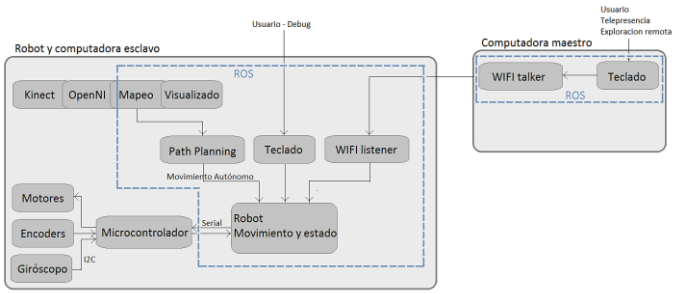

Figura 3. Diseño funcional de la arquitectura del sistema.

El robot cuenta con tres modos de operación:

- *Modo debugging:* se puede observar en **Figura 4** un nodo de ROS llamado keyboard para ingresar comandos de movimientos directamente desde el teclado de la computadora a bordo del robot.

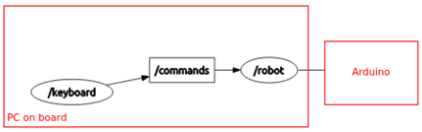

Figura 4. Modo debugging. Conexión entre nodos de ROS.

- *Modo de telepresencia:* la **Figura 5** muestra la comunicación TCP entre la PC remota y la PC a bordo. El nodo wifi_talker (en la PC remota) envía comandos al nodo wifi_listener (en la PC a bordo), y este último retransmite los comandos al nodo robot, haciendo uso de una conexión USB-Serial con Arduino.

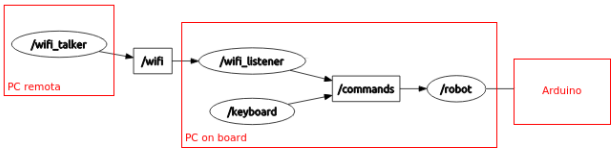

Figura 5. Modo telepresencia.

Modo de mapeo y exploración: la **Figura 6** agrega un nodo en la PC a bordo que permite al robot planear sus movimientos autónomamente en base a un mapa, generado con el sensor Kinect. Dicho nodo es el encargado de computar los caminos óptimos y de enviar los comandos al robot.

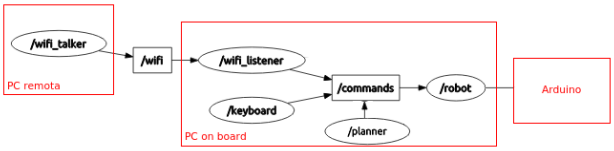

Figura 6. Modo de mapeo y exploración autónoma.

V. Localización

Es necesario que el robot sea capaz de auto-localizarse, para ello debe conocer su posición en cada instante. Se lo referencia en un sistema de coordenadas donde su estado queda definido por (x, y, Φ), que representan los desplazamientos en cada eje y la orientación del robot (ver **Figura 7**). Para el avance del robot, se calcula la distancia Euclidiana desde el punto inicial (x_1, y_1) hasta el punto final (x_2, y_2).

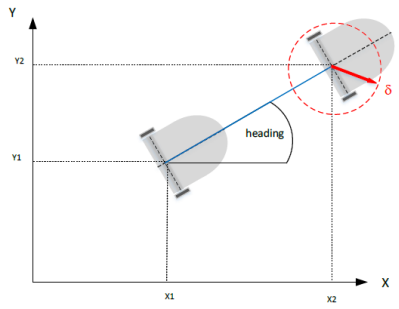

Figura 7. Posicionamiento del robot en un sistema de coordenadas. Localización del robot.

Debido a potenciales desviaciones, es posible que el robot no pase exactamente por el punto (x_2, y_2), entonces se admite un error denominado delta δ. Para controlar que el robot haya llegado a destino, se debe cumplir la siguiente desigualdad:

$$dx^2 + dy^2 \leq \delta^2$$

La ecuación se puede visualizar como un círculo de radio delta, que delimita una zona con centro en el objetivo (ver circulo punteado de la **Figura 7**). Dentro de dicha zona, el error es admisible. Si el robot se encuentra dentro de la zona, se considera que ha alcanzado el objetivo.

La implementación de la función de localización se realizó sobre el microcontrolador Arduino. El análisis de resultados se puede ver en la sección IX.A.

VI. Control

Bajo condiciones ideales, los dos motores del robot deberían ser idénticos, y al proporcionarles la misma entrada, deberían avanzar a la misma velocidad. En la realidad, esto no sucede y las diferencias entre los motores resultan en desvíos del robot.

Como se busca lograr que el robot se mueva en línea recta sobre el eje X, el objetivo del controlador es que el crosstrack error (componente perpendicular a la trayectoria, "CTE" de aquí en adelante) disminuya a cero. Para ello, se implementó un controlador Proporcional Integrador Derivador el cual permite orientar el ángulo de giro del robot según la linea de trayectoria deseada, disminuyendo el CTE a cero.

El robot controla el ángulo de giro a través de la diferencia de velocidad de sus dos motores. Dado que el prototipo fue implementado con 4 comandos, up, down, right y left, se busca que ambos motores giren a la misma velocidad siempre, y con sentidos contrarios de giro en el caso de left y right.

A. *Controlador Proporcional*

Lleva su nombre dado que el ángulo de corrección α es proporcional al CTE, según un valor denominado Kp.

$\alpha = Kp * CTE$ siendo α el ángulo de corrección

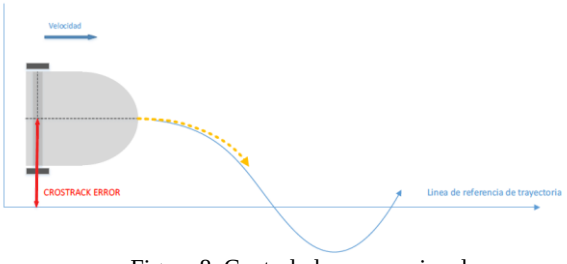

Figura 8. Controlador proporcional

Luego de sucesivas correcciones, el robot llegará a la línea de referencia y la sobrepasará (Ver **Figura 8**). Este efecto de oscilación y las amplitudes dependen del valor de Kp. Un mayor valor implica correcciones más significativas en cada paso, pero también un sobrepasamiento mayor y un error estacionario mayor. Pero el robot oscila continuamente. Para solucionarlo se introdujo un nuevo término. El controlador PD.

B. Controlador proporcional derivador

Para hacer las correcciones más adaptivas, y evitar un sobrepasamiento pronunciado, se disminuye la amplitud de la correccion a medida que el robot se acerca a la línea de referencia, y se aumenta la corrección a medida que se aleja (Ver **Figura 9**). Dado que la derivada representa pendientes instantáneas, derivando el CTE respecto al tiempo, se miden los cambios en la orientación. Al llegar a la orientación deseada, la derivada será cero (CTE=cte) y la trayectoria del robot será paralela a la línea de referencia. Computacionalmente, se la puede calcular como una diferencia de errores:

$$\alpha = Kp * CTE + Kd * dCTE / dt$$

siendo $dCTE / dt = \mathrm{CTE_t} - \mathrm{CTE_{t-1}} / \Delta t$

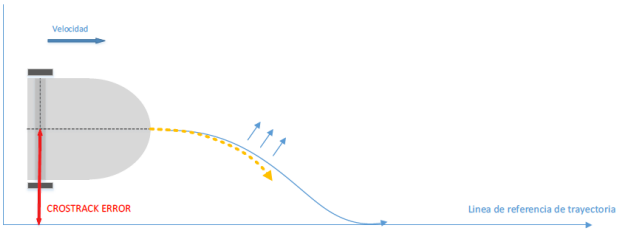

Figura 9. Controlador proporcional derivador

No obstante, el hecho de que el robot y la linea de trayectoria deseada sean paralelos, no implica que sean coincidentes. Para lograrlo, es necesario introducir un último termino. El controlador PID.

C. Controlador proporcional derivador integrador

A medida que transcurre el tiempo el robot se va alejando de la trayectoria deseada producto de una desviación sistemática (Ver **Figura 10**). Para que esto no ocurra es necesario que el robot se acerque paulatinamente a la línea de referencia deseada compensando la desviación. Éste se mide a partir de la integral del error, que representa la suma de los CTE a lo largo del tiempo. La ecuación completa del controlador PID se muestra a continuación:

$$pid = Kp * error + Kd * \frac{derror}{dt} + Ki * sum(error)$$

Proporcional Derivador Integrador

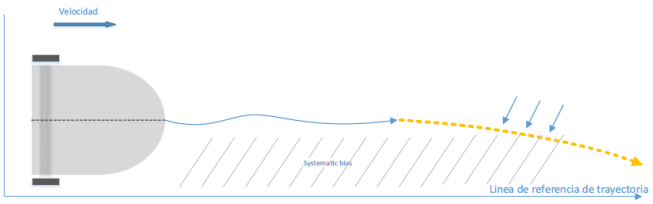

Figura 10. Controlador proporcional dervidaor integrador

El diseño completo del sistema de control se muestra en la **Figura 11**. Hay un controlador PID para cada motor, y el lazo se cierra a partir de los encoders que permiten calcular la velocidad de cada uno de éstos. Luego de realizar las correciones con el controlador, se envía una señal PWM (Modulación por ancho de pulso) al motor, con la velocidad de giro corregida.

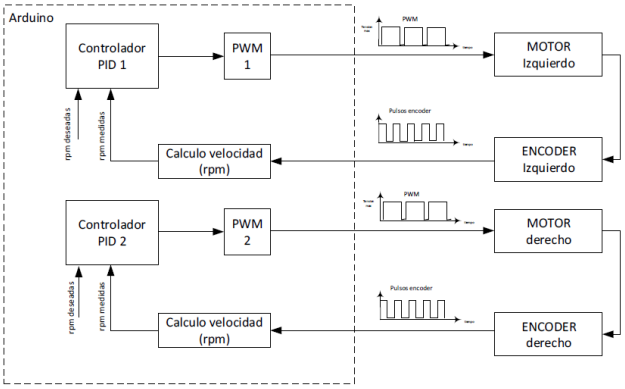

Figura 11. Sistema de control del robot.

Dicha implementación fue llevada a cabo en el microcontrolador Arduino. El análisis de resultados puede verse en la sección IX.B.

VII. MAPEO

El mapeo se realizó a partir de la información que proveen las cámaras RGB y RGBD del sensor Kinect para la obtención de imágenes del entorno (de color y profundidad), que permiten realizar la odometría visual. Ésta consiste en el proceso de estimación de la posición y la orientación del robot a partir del análisis de una secuencia de imágenes tomadas desde una cámara a bordo del robot. Existen dos formas conocidas para realizar el cómputo del movimiento a partir de una secuencia de imágenes tomadas por cámaras desplazándose:

A. Basado en apariencias

Identifica regiones correlacionando imágenes a partir de los valores de intensidad de los píxeles de cada imagen. Se utilizó la implementación del proyecto RTAB-Map (Labbé).

B. Basado en extracción de características

Identifica regiones con características particulares y distintivas, y luego correlaciona la secuencia de imágenes. Se utilizó la implementación ccny_rgbd (Ivan Dryanovski).

Los métodos basados en extracción de características generalmente son más precisos y rápidos que los globales (basados en apariencia), siendo estos últimos muy caros computacionalmente. Debido a que la cámara va situada sobre un robot móvil, el factor velocidad es muy importante, ya que durante los movimientos se producen vibraciones y cualquier movimiento repentino no daría tiempo a la correlación global de las imágenes. Por ello, se utilizó ccny_rgbd (Dryanovski). Con la herramienta de ROS, Rviz, es posible visualizar los mapas resultantes y la trayectoria del robot durante el mapeo. Los resultados se presentan en la sección IX.C.

VIII. NAVEGACIÓN

Un problema al que se enfrenta todo robot móvil es cómo

alcanzar su objetivo dado un mapa y su ubicación en el mismo. Para ello se plantea un algoritmo en el que dados: un mapa del lugar donde se encuentra el robot, posición inicial, posición final u objetivo, y costo de realizar cada movimiento, debe encontrar el camino de menor costo.

A partir del módulo de mapeo, se obtiene otro mapa 2D haciendo un corte al mapa 3D en el eje Z (altura) según el plano de la cámara. Éste es representado mediante una matriz de dimensión 100x100, en la cual cada elemento se corresponde con 10cm de la realidad. Esta es la entrada al módulo planificador, junto con las coordenadas actuales del robot, y el punto al cual desea llegar.

Se analizaron e implementaron tres algoritmos. A continuación se realiza una breve descripción de cada uno:

A. *Breadth-first search BFS*

Se basa en una conocida estrategia de búsqueda sobre grafos la cual permite calcular distancias desde un nodo, hacia todos los demás. Para ello, se elige un nodo raiz (estado inicial) y luego, se realiza una expansión hacia todos sus vecinos. En el siguiente paso se visita a los vecinos de los vecinos, y así sucesivamente hasta recorrer todos los nodos. Además contiene una lista donde guarda los nodos visibles cada vez que se hace una expansión a un nodo vecino, y se tiene un costo asociado a cada nodo (valor-g), que se incrementa en cada expansión. Este valor lleva la cuenta de cuántas veces es necesario expandirse desde el estado inicial para alcanzar el estado actual. Al final del algoritmo, se llegará al objetivo, y el valor g será el largo del camino óptimo.

B. *A* (a estrella)*

Es una variante más eficiente del algoritmo BFS. Brinda una mejor performance a partir del uso de heurísticas que hacen que no sea necesaria la expansión hacia todos los nodos del grafo. Para expandirse no sólo elige a que nodo hacerlo según el valor-g, sino que también usa una función heurística h. Esta se representa mediante una matriz donde cada elemento provee una estimación o suposición optimista de cuantos pasos necesita el robot hasta alcanzar su objetivo:

$$h(x,y) \leq distancia\ al\ objetivo\ desde\ x,y$$

La modificación sobre el algoritmo BFS es muy simple. La nueva estrategia de decisión en la expansión, se basa en elegir el nodo con el menor valor-f (que ya contiene el valor-g):

$$f = g + h(x, y)$$

g=distancia desde el nodo inicial hasta el actual
h(x,y)= estimación desde el nodo actual hasta el final

En caso de que la matriz h contenga todos 0, el algoritmo se comporta exactamente igual al BFS.

C. *Basado en programación dinámica*

Éste algoritmo calcula el mejor camino hacia él desde todos los puntos del mapa. En el mundo real, estocástico, las acciones son no deterministas (ej. el robot podría sufrir un desliz, tomar un camino en el que hay otro robot, etc.). Entonces, permite hacer un plan no sólo para la posición más probable, sino para cualquier posición del mapa en la que el robot se encuentre. Se basa en programación dinámica, lo que disminuye la cantidad de cálculos necesarios.

Se consideró que el BFS (Breadth-first search) era el más apropiado en relación a los objetivos y requerimientos del proyecto. En la sección **¡Error! No se encuentra el origen de la referencia.** se pueden ver los resultados obtenidos.

IX. Análisis de resultados

A. *Localización*

Según las pruebas realizadas, para movimientos de giro el robot registra un error menor al 4%. Por otro lado, para los movimientos lineales se logra tener un error muy pequeño (siempre menor al 5%).

En cada comando que el planificador le envía al robot, le debe indicar cuánto avanzar, y si se toma una cantidad de avance baja, por ejemplo 10cm, se mantiene un error de localización muy pequeño. Luego, cuando el robot necesite avanzar cantidades mayores lo hará como una secuencia de comandos.

B. *Control*

En todos los test realizados para diferentes velocidades dadas, ambos motores se ajustaron correctamente y tuvieron la misma velocidad promedio. Esto se comprobó con un tacómetro digital. A continuación se muestra el resultado para una velocidad dada de 10rpm:

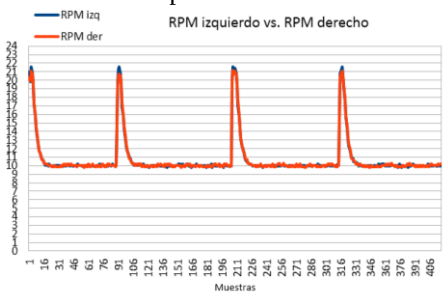

Figura 12. Test de control. Velocidades de cada motor en [rpm] para cada comando. Curvas superpuestas.

A medida que se le exigen velocidades de avance más grandes, el tiempo de respuesta empeora. Por lo tanto el mejor funcionamiento se obtiene para velocidades más pequeñas.

C. *Mapeo y visualización*

Los resultados obtenidos del módulo de mapeo y visualización se muestran a continuación. El mapa de la Figura 13 fue generado a medida que se realizaba una rotación de 360° de la cámara RGBD en el centro de la habitación, permitiéndole una observación completa del entorno.

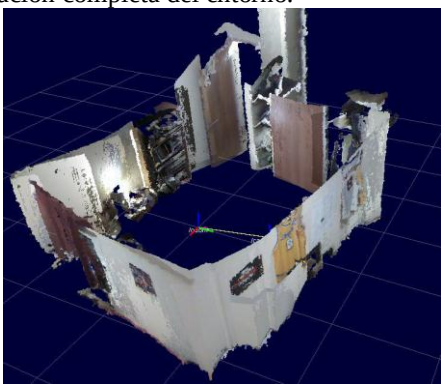

Figura 13. Mapa de una habitación generado con el sensor Kinect.

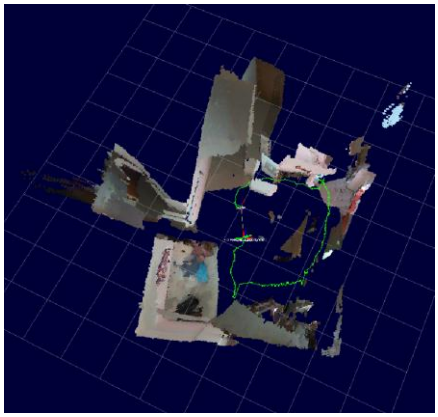

Figura 14. Se muestra la trayectoria del robot durante el mapeo

En cuanto a la precisión, las mediciones realizadas mostraron que el error, tanto durante las traslaciones, como en las rotaciones, es insignificante para los fines de este proyecto.

D. Navegación

A continuación, se muestra un fragmento del mapa en 2D obtenido a partir de Figura 13. El punto inicial es el elemento marcado con un círculo azul, mientras que el objetivo, puerta de salida hacia otro entorno a explorar, se marca en rojo:

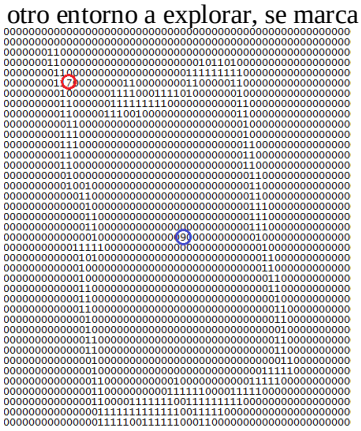

Figura 15. Mapa en 2D creado a partir de la visualización con Kinect.

El nodo planner de ROS donde se implementa el algoritmo BFS, devolverá como resultado un mapa con el camino óptimo, y además enviará los siguientes comandos al robot para que se desplace hasta el punto dado (Ver Figura 16).

```
real_path(robot)= ['init', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up',
'up', 'up', 'up', 'left', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up',
'up', 'up', 'up', 'up', 'up', 'right', 'up', 'up', 'up', 'up', 'left', 'up',
'up', 'up', 'stop']
```

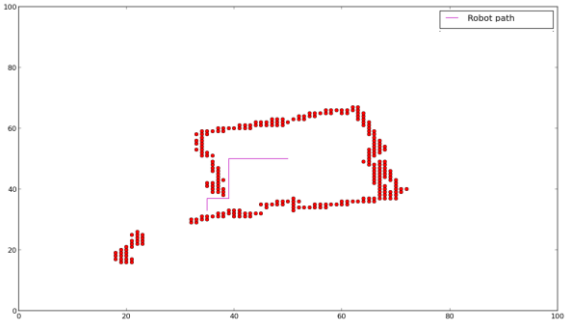

Figura 16. Camino optimo generado por el modulo planner de ROS.

X. Conclusión

La utilización de un diseño modular y una arquitectura basada en componentes, permitió desarrollar cada uno por separado y en paralelo. Esto agilizó mucho los tiempos de desarrollo. Luego se fueron integrando y probando las partes.

Por otra parte, ROS y Arduino disponen de librerías y drivers que permiten utilizar los sensores muy rápidamente, con comunidades de desarrollo activas que proveen buenos tutoriales y documentación.

El control del robot fue una de las etapas con mayores desafíos. La estructura mecánica y el tipo de rueda delantera causaron muchos problemas que exceden a la implementación y el funcionamiento del controlador. Respecto al mapeo y la odometría visual, fue de vital importancia para el proyecto contar con las implementaciones de ROS mencionadas en el apartado VII. En la etapa de generación de trayectorias y navegación, debido a la modularidad del sistema, resultó muy simple utilizar el mapa como entrada del planeador de trayectorias y a su vez enviar la secuencia de comandos al microcontrolador.

XI. Referencias

[1] Robot Operating System. (s.f.). Obtenido de http://www.ros.org/wiki/

[2] Robot Operating System. (s.f.). *Tools*. Obtenido de http://www.ros.org/wiki/Tools

[3] Robot Operating System. (s.f.). *Concepts*. Obtenido de http://www.ros.org/wiki/ROS/Concepts

[4] Dryanovski, I. c. (s.f.). *ROS*. Obtenido de http://www.ros.org/wiki/ccny_rgbd

[5] Ivan Dryanovski, R. G. (s.f.). Fast Visual Odometry and Mapping from RGB-D Data. *2013 International Conference on Robotics and Automation (due May 2013)*.

[6] Labbé, M. M. (s.f.). Appearance-based loop closure detection in real-time for large-scale and long-term operation. IEEE Transactions on Robotics.

Orlando Micolini (M'98**)** Grado en Ingeniero Electricista Electrónico, año 1981, de la UNC (FCEFyN) Argentina. Postgrado Especialista en Telecomunicaciones, año 2002, de la UNC (FCEFyN). Magíster en Ciencias de la Ingeniería, año 2002, de la UNC (FCEFyN). Director de la Carrera de Ingeniería en Computacion en la UNC (FCEFyN) (2004-actualidad). Titular de la asignatura Arquitectura de Computadoras en la UNC (FCEFyN) (1996-actualidad). Actualmente está realizando el doctorado de Ciencias Informática en la UNLP, en sistemas multi-core heterogéneos con Redes de Petri. (email: omicolini@compuar.com)

Ladislao Mathé (M'92) Grado en Ingeniero Electricista Electrónico en la UNC (FCEFyN) Argentina en 1976. En 1977, se unió al Departamento de Ingeniería Electrónica de la UNC (FCEFyN), como Profesor de Control. Director del Grupo de Robótica y Sistemas Integrados GRSI de la UNC (FCEFyN) desde 2003. Fue presidente de la subsección Córdoba IEEE-Sección 9 durante los años 2003-2004. Actual presidente del AESS de la IEEE. (email: mathe@ieee.org)

Caverzasi Agustín (M'11) 1989, nacido en la ciudad de Las Varillas, Córdoba, Argentina. En 2008 ingresó a la carrera Ingeniería en Computación en la UNC (FCEFyN), Argentina. En 2013 obtuvo su título de grado de Ingeniero en Computación. (email: agucaverzasi@gmail.com)

Saravia Rajal Fernando (M'11) 1990, nacido en Salta, Argentina. En 2008 ingresó a la carrera Ingeniería en Computación en la UNC (FCEFyN), Argentina. En 2013 obtuvo su título de grado de Ingeniero en Computación. (email: fersarr@gmail.com)