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

Agustín Caverzasi[#1], Fernando Saravia[#2], Orlando Micolini [#3], Ladislao Mathé [#4] Leandro Federico Lichtensztein[#5]

[#] *Facultad de Ciencias Exactas, Físicas y Naturales, Universidad Nacional de Córdoba*
Av. Vélez Sarsfield 1611, Córdoba, Argentina

[1] agucaverzasi@gmail.com
[2] fersarr@gmail.com
[3] omicolini@compuar.com
[4] ing.ladislao.mathe@gmail.com
[5] leandrosnm@gmail.com

***Abstract*— This work presents a mobile robot capable of producing a rough map of an interior of a building. The goal is to build a mobile robot, with the ability of self-localization, navigation, and mapping. To accomplish this, several technologies were used: Arduino development board, the Robot Operating System (ROS), and a 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.**

***Resumen*— En este trabajo se presenta a un robot móvil capaz de realizar un mapa aproximado del interior de un entorno cerrado. El objetivo principal del trabajo es la descripción de la construcción del robot móvil, con capacidad de auto-localización, navegación y mapeo. Para ello se utilizaron varias tecnologías, tales como: la placa de desarrollo Arduino, el Sistema Operativo del robot (ROS), y el sensor Kinect. El robot desarrollado, además de permitir realizar mapeos, ofrece un amplio campo de aplicaciones como exploración de zonas peligrosas, la telepresencia y la educación.**

I. Introducción

Los últimos años han sido de gran importancia para el desarrollo y aplicación de la robótica. 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– ROS- 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:

Exploración y construcción de un mapa aproximado en un espacio cerrado y acotado, el cual que posee obstáculos pero permiten la circulación.

B. Objetivos Específicos:

Diseño e implementación de una plataforma móvil controlada tanto de forma autónoma, como remota, capaz de desplazarse dentro de una habitación, con el objetivo de obtener mediciones de la misma y evitar los obstáculos que se interpongan.

II. Estado del arte

A. Robots móviles. Clasificación

Los robots móviles son clasificados según el movimiento que son capaces de realizar; existen dos tipos: Si 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: Diferencial (dos ruedas cada una con un motor independiente para la tracción) o Ackerman (un motor para la tracción y uno para controlar la dirección de giro como mínimo).

B. Robot Operating System

Provee librerías y herramientas que brindan a los desarrolladores todo lo necesario para crear aplicaciones relacionadas a la robótica [1], [2]. 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, 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, GUIs, etc.) para el desarrollo de robots y sus aplicaciones (ej. Rqt , tf, rviz [3]).

ROS organiza el software según una arquitectura de grafos en la que los nodos son porciones ejecutables de código y éstos se comunican entre sí a través de 'topics'. Estos últimos definen la interfaz con la que se comunican dichos 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 [4].

C. Odometría

Técnica que permite estimar el cambio de posición de un sistema, a partir del uso de datos provenientes de sensores – encoders, magnetómetros, giróscopos, etc. –. Se utiliza a

978−1−4799−4269−5/14/$31.00

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 que produce el movimiento sobre las imágenes obtenidas por las cámaras a bordo del robot. Algunos factores que podrían dificultar el proceso son la falta de iluminación, escenas con escasa textura o 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

Se construyó un robot no-holonómico cinemático diferencial. El diseño se compone de dos niveles (ver Figura 1): en el nivel inferior tendrá lugar una placa electrónica para el control de los motores y sensores de odometría, y un microcontrolador para la programación de la funcionalidad de bajo nivel como el control y la localización del robot. El siguiente nivel está compuesto por una computadora a bordo y una cámara RGBD que llevarán a cabo las tareas de navegación, visualización y mapeo.

A continuación se describen las alternativas analizadas.

A. Sensores para localización.

La posición del robot es estimada a partir de la medición de distancias recorridas por el mismo. Se tuvieron en cuenta los siguientes sensores: encoders rotacionales, sensores de ultrasonido, infrarrojos, y una cámara RGBD. Se optó por utilizar encoders, dado que presentan la ventaja de que ya se encuentran integrados a los motores que le darán el movimiento al robot.

Para estimar la orientación, se han considerado: un magnetómetro HMC5883L, y un giróscopo MPU 6050 (GY-521). Luego de algunos estudios y pruebas con ambos sensores, se decidió incluir el giróscopo, dado que el magnetómetro en muchas ocasiones es susceptible a interferencias magnéticas.

B. Placa de desarrollo. Microcontrolador

Para llevar a cabo la implementación del control y la localización del robot, se analizaron las siguientes placas de desarrollo a incluir en el robot: NXP LPC1343 Board y Arduino Mega2560. Luego de experimentar sobre ambas, se eligió Arduino debido a su estado de desarrollo tanto en hardware como software (ésta posee una importante cantidad de módulos adicionales de hardware para manejo de sensores, y librerías de software).

C. Sistema Operativo

El desarrollo del sistema fue llevado a cabo sobre el Robot Operating System ejecutado sobre Linux (Ubuntu 12.10).

Figura 1. Robot móvil. Su composición.

IV. Arquitectura del Sistema

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

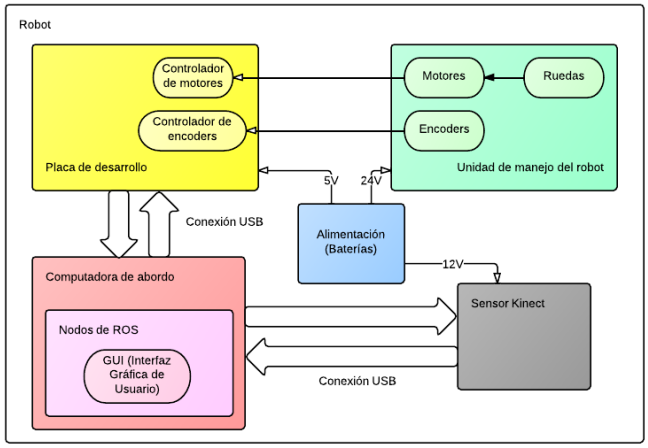

Figura 2. Diseño modular de la arquitectura del sistema.

La Fig. 2 muestra los componentes del robot. Las funciones de los principales componentes son:

- Unidad de manejo del robot: Contiene los motores que dan movimiento robot, una rueda frontal libre para su estabilización, y encoders para la localización.
- Placa de desarrollo: Controla por software la unidad de manejo del robot, y realiza la localización y el control. Además, envía y recibe datos desde/hacia la computadora a bordo del robot.
- Computadora a bordo: Es el módulo principal donde corre ROS. Contiene los nodos: 'robot' que envía comandos de movimiento a Arduino, 'planner' para la planificación de trayectorias, y todos los nodos relacionados a la visualización y mapeo realizado a partir del sensor Kinect.
- Sensor Kinect: Envía streams de color y de profundidad a la computadora que realiza las tareas de mapeo y visualización.

La Figura 3 muestra el diseño funcional del sistema. En ella se puede apreciar la interacción entre cada nodo. Por un lado, los recuadros en línea punteada encierran nodos pertenecientes al sistema operativo ROS.

Por el otro, los dos recuadros sombreados constituyen la computadora de abordo del robot (o maestro), y la computadora remota (o esclavo) utilizada para el modo de operación de telepresencia según se explica a continuación en modos de operación.

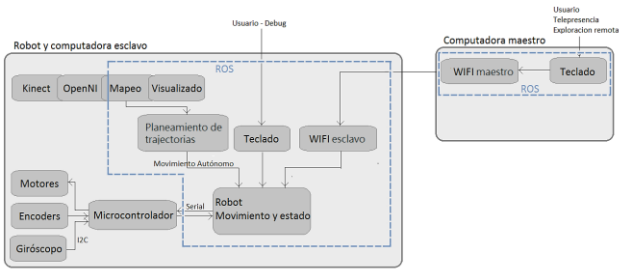

Figura 3. Diseño funcional de la arquitectura del sistema.

El robot cuenta con tres modos de operación:

- Modo debugging: en la Fig. 4 se puede observar un nodo de ROS llamado 'keyboard' para el ingreso de comandos directamente desde el teclado de la computadora de abordo del robot, que se traducirá luego en un movimiento en el nodo 'robot'.

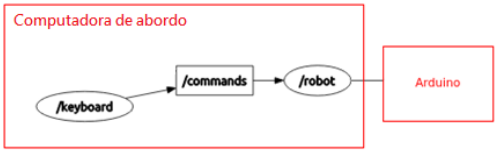

Figura 4. Modo debugging. Conexión entre nodos de ROS.

- Modo de telepresencia: la Fig. 5 muestra la comunicación TCP entre la computadora remota y la computadora de abordo. El nodo 'wifi_talker' (en la computadora remota) envía comandos al nodo 'wifi_listener' (en la computadora de abordo), y este último retransmite los comandos al nodo robot, haciendo uso de una conexión USB-Serial con la placa Arduino.

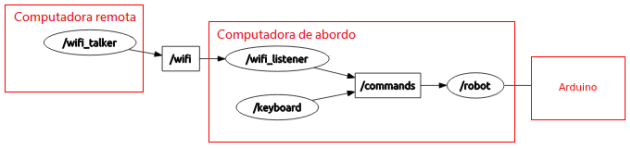

Figura 5. Modo telepresencia.

- Modo de mapeo y exploración: la Fig. 6 agrega un nodo 'planner' en la computadora de abordo que permite al robot planear sus trayectorias autónomamente (ver sección IV) según el mapa generado por el sensor Kinect (ver sección IV). Este nodo es el encargado de computar los caminos óptimos y enviar los comandos al robot.

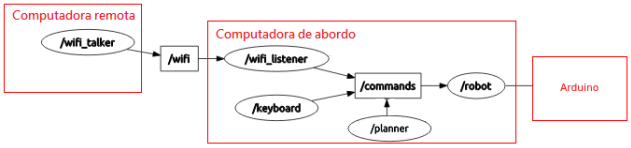

Figura 6. Modo de mapeo y exploración autónoma.

V. Localización

Para que el robot sea capaz de auto-localizarse, debe conocer su posición en cada instante. Por ello, 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).

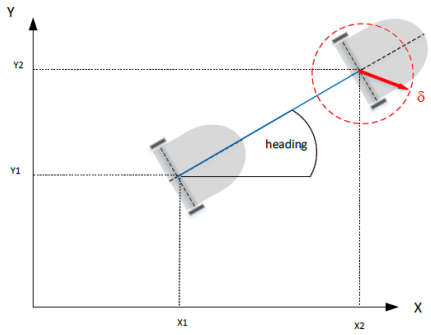

Figura 7. Localización. Posicionamiento en un sistema de coordenadas.

Debido a potenciales desviaciones, es posible que el robot no pase exactamente por el punto (x2, y2), entonces se admite un error denominado delta δ . Para controlar que el robot haya llegado a destino, se debe cumplir la siguiente desigualdad:

$$dx^2 + dy^2 \leq \delta^2 \quad (1)$$

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 se considera 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, si se proporciona la misma entrada a cada servomotor del robot deberían avanzar a la misma velocidad. En la práctica, esto no sucede y las diferencias entre los motores resultan en desvíos del robot. Ya que se busca lograr un movimiento rectilineo sobre el eje x, el objetivo del controlador es que el cross-track error (componente perpendicular a la trayectoria, "CTE" de aquí en adelante) disminuya hacia cero (Ver Figura 8).

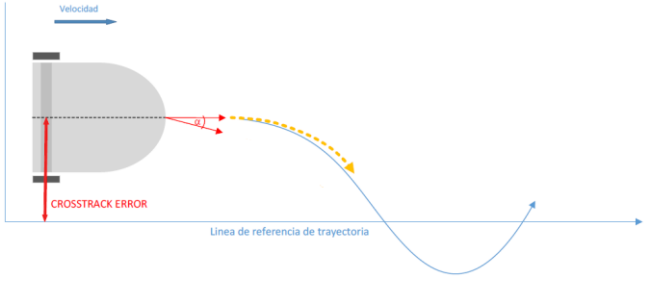

Figura 8. Movimiento del robot y error.

Para ello, se implementó un controlador Proporcional Integrador Derivador (PID) el cual permite orientar el ángulo de giro del robot según la linea de trayectoria deseada, disminuyendo el CTE a cero. El diseño completo del sistema de control se muestra en la Figura 9. Hay un controlador PID para cada motor, y el lazo se cierra con los datos de localización que proveen los encoders, a partir de los cuales se calcula la velocidad de cada uno de los motores. Luego de realizar las correciones con el controlador, se envía una señal PWM (Modulación por ancho de pulso) a cada motor, con la velocidad de giro

corregida.

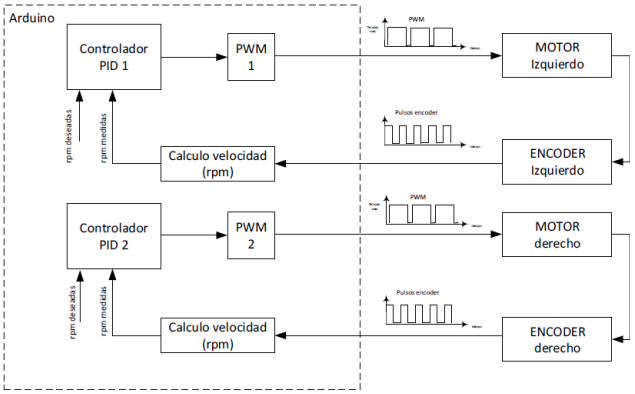

Figura 9. Sistema de control del robot.

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 cuatro 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 Derivador Integrador

En la ecuación (2) se muestra cada termino del controlador PID.

$$\propto = K_p * CTE + K_d * \frac{d\ CTE}{dt} + K_i * \sum CTE \qquad (2)$$

El primer término denominado 'proporcional' lleva su nombre dado que el producto determina un ángulo de corrección α proporcional al error CTE, según un valor Kp, $\alpha = Kp{*}CTE$. Luego de sucesivas correcciones α, el robot alcanzará la línea de referencia de trajectoria y la sobrepasará (Ver Fig. 8). Rápidamente, hará el mismo movimiento pero en sentido opuesto, y asi sucesivamente, produciendose un efecto de oscilación cuya amplitud depende del valor Kp.

Un mayor valor implica correcciones más significativas en cada paso, pero también un sobrepasamiento mayor y un error estacionario mayor.

Para hacer las correcciones más adaptivas y evitar un sobrepasamiento pronunciado, el segundo término de la ecuación (2) denominado 'derivador' disminuye la amplitud de la corrección a medida que el robot se acerca a la línea de referencia y se aumenta la corrección a medida que se aleja. Dado que la derivada representa pendientes instantáneas, derivando el valor de 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 $dCTE / dt = \mathrm{CTE_t} - \mathrm{CTE_{t-1}} / \Delta t$.

No obstante, el hecho de que el robot y la línea de trayectoria deseada sean paralelos, no implica que sean coincidentes. Para lograrlo, se introdujo el último término de (2) denominado 'integrador' cuyo efecto permite que el robot se acerque paulatinamente a la línea de referencia deseada compensando la desviación sistemática que habia sufrido. Éste se calcula a partir de la integral del error, que representa la suma de todos los errores CTE a lo largo del tiempo.

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, es decir, obtención de imágenes (de color y profundidad) del entorno que permiten realizar la odometría visual se realizó a partir de la información que provee la cámara RGB-D del sensor Kinect. 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 mismo. Existen dos formas conocidas para realizar el cómputo del movimiento a partir de una secuencia de imágenes tomadas por cámaras desplazándose. Estas son brevemente descriptas a continuación.

A. Método basado en apariencias

Dicho enfoque identifica regiones correlacionando imágenes a partir de los valores de intensidad de los píxeles de cada imagen. Para su análisis se utilizó una implementación en ROS presentada por Labbe and Michaud [5] y llevada a cabo por el proyecto RTAB-Map.

B. Método basado en extracción de características

Identifica regiones con características particulares y distintivas, y luego correlaciona la secuencia de imágenes. Se utilizó el método introducido por Dryanovski et al. [6] cuya implementación está disponible para ROS en un módulo denominado ccny_rgbd.

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 a la hora de escoger un método, 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 el módulo ccny_rgbd [6].

Por otra parte, la herramienta de visualización de ROS Rviz, hizo posible ver los mapas resultantes y la trayectoria del robot durante el proceso de mapeo. Los resultados se presentan en la sección IX.C.

VIII. NAVEGACIÓN

El problema de navegación consiste en determinar qué estrategia utilizará el robot para 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, la posición inicial, la posición final u objetivo, y costo de realizar cada movimiento, el robot debe encontrar el camino de menor costo.

Desde el módulo de mapeo (Ver sección IV) se obtiene un mapa 3D, a partir del cual se alcanza un mapa 2D al hacer un corte en el eje z (altura) según el plano de la cámara. Éste es representado mediante una matriz de 100x100 elementos, en la cual cada uno representa 10cm de la realidad. Dicha matriz será la entrada al módulo

planificador de trayectorias, 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 raíz (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:

$$h(x,y) \leq distancia\ al\ objetivo\ desde\ x,y \qquad (3)$$

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

$$f = g + h(x,y) \qquad (4)$$

Siendo g la distancia desde el nodo inicial hasta el actual, y $h(x,y)$ la 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*

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

Luego de probar dichos algoritmos descriptos, se optó por utilizar el algoritmo BFS (Breadth-first search) por simplicidad, y dado que brindó una buena performance en relación a los objetivos propuestos en el presente trabajo. Dicho algoritmo ha sido implementado en un nodo de planificación de trayectorias denominado 'planner' de ROS. El modulo devolverá como resultado un mapa con el camino óptimo que deberá seguir el robot para llegar a su objetivo.

IX. Análisis de los 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 inferior al 5%).

En cada comando que el planificador le envía al robot, le debe indicar cuánto avanzar. Si se toma una cantidad de avance baja, por ejemplo 10 cm, se mantiene un error de localización pequeño, y cuando el robot necesita avanzar cantidades mayores lo hace como una secuencia de pequeños avances.

B. *Control*

En todos los ensayos realizados para diferentes velocidades, ambos motores se ajustaron correctamente y tuvieron la misma velocidad promedio. En cada caso ha sido comprobado con un tacómetro digital. En la Figura 10 se muestra el resultado para una velocidad escogida de 10rpm:

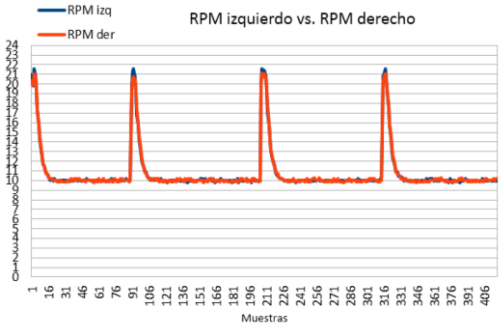

Figura 10. Test de control. Velocidades de cada motor en [rpm] para cada comando. Curvas superpuestas.

Luego de sucesivos tests a diferentes velocidades, se ha comprobado que 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*

A partir del módulo de mapeo y visualización se obtuvo el mapa de la Figura 11, que fue generado a medida que se realizaba una rotación de 360° de la cámara RGB-D en el centro de la habitación permitiéndole una observación completa del entorno. En Figura 12 se muestra el mapa de la misma habitación desde otro ángulo desde el cual se puede apreciar la trayectoria del robot en verde.

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.

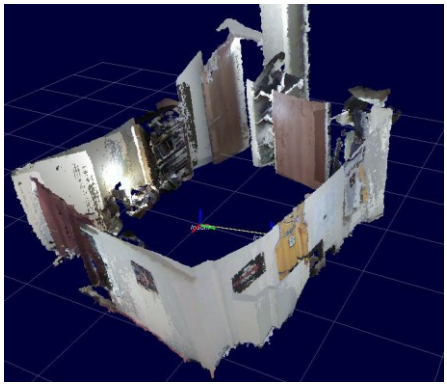

Figura 11. Mapa de una habitación generado con el sensor Kinect.

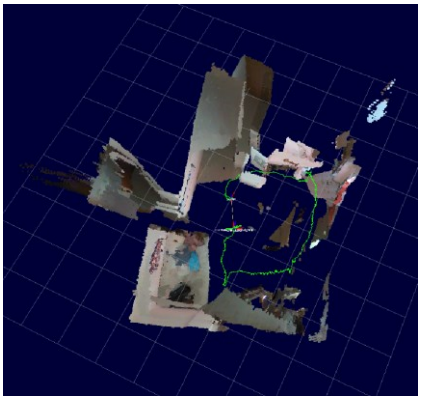

Figura 12. Se muestra la trayectoria del robot durante el mapeo.

D. Navegación

En la Figura 13 se muestra un fragmento del mapa 2D obtenido a partir de la

Figura 11. El punto inicial es el elemento marcado con un círculo azul, mientras que el objetivo, la puerta de salida hacia otro entorno a explorar, se ve en rojo.

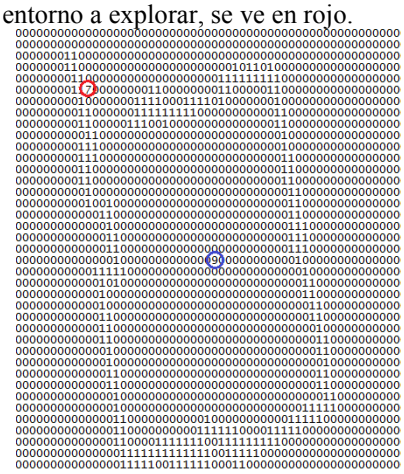

Figura 13. Mapa en 2D creado a partir de la visualización con Kinect.

El nodo de planificación de trayectorias devuelve como resultado un mapa con el camino óptimo (Ver Figura 14), y además luego de una trasformación del camino en movimientos, le envía los siguientes comandos al robot para que se desplace hasta el punto dado:

```
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']
```

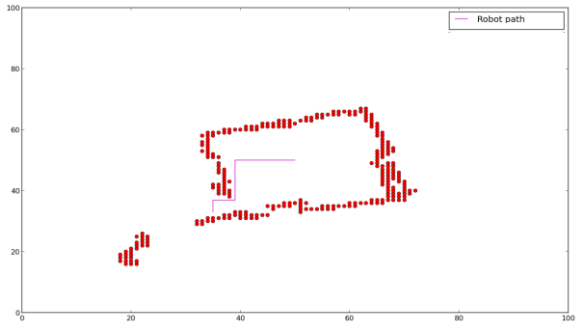

Figura 14. Camino óptimo generado por el módulo planner de ROS.

X. Conclusión

La utilización de un diseño modular y una arquitectura basada en componentes, permitió hacer un desarrollo de cada componente por separado y en paralelo lo cual aceleró notablemente los tiempos de desarrollo. Luego, las partes se fueron probando individualmente e integrándose en el sistema.

Por otra parte, las tecnologías 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 a lo largo de todo el desarrollo, abordado con la implementación de un controlador PID. Respecto al mapeo y la odometría visual, fue de vital importancia para el proceso de implementación contar con los módulos de ROS que han sido utilizados, según se describió 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 final al microcontrolador para que sean traducidos en movimientos del robot.

Referencias

[1] WillowGarage, “Robot Operating System(ROS),” http://wiki.ros.org, 2012. .

[2] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS: an open-source Robot Operating System,” in ICRA Workshop on Open Source Software, 2009, vol. 32, pp. 151–170.

[3] WillowGarage, “ROS Tools.” [Online]. Available: http://www.ros.org/wiki/Tools.

[4] WillowGarage, “ROS Concepts.” [Online]. Available: http://www.ros.org/wiki/ROS/Concepts.

[5] M. Labbe and F. Michaud, “Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation,” IEEE Trans. Robot., vol. 29, no. 3, 2013.

[6] Dryanovski Ivan, R. G. Valenti, and J. Xiao, “Fast visual odometry and mapping from rgb-d data,” IEEE Int. Conf. Robot. Autom., 2013.