UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM VALLE DE MÉXICO

Implementación de Robótica en la Nube para Interacción de Robots Móviles

T E S I S

Que para obtener el Grado de

Maestro en Ciencias de la Computación

P r e s e n t a

Ing. Marco Antonio Aguilar Tadeo

Tutor Académico:

Dr. José Martín Flores Albino

Tutores Adjuntos:

Dr. Víctor Manuel Landassuri Moreno

Dr. Saul Lazcano Salas

Atizapán de Zaragoza, Edo. de México diciembre de 2018

RESUMEN

El presente trabajo se desarrolla sobre un sistema de interacción entre robots, el cual busca mostrar el concepto de robótica en la nube y los beneficios que este concepto traerá para poner más cerca de todos a los robots.

La robótica en la nube es un campo de investigación de actualidad que pretende otorgar a los robots los servicios que da la nube y así potencializar su funcionamiento y desempeño. Robótica en la nube puede ser descrita como una red social para robots, donde se pueda compartir, almacenar y procesar información para que los robots tengan con mayor facilidad la capacidad de "aprender" a desempeñar nuevas tareas de acuerdo a su diseño y capacidad.

Otro objetivo que tiene la robótica en la nube es la disminución de costos en la elaboración de robots, los cuales poseen una gran cantidad de sensores para trabajar. Sin embargo, al generar y compartir información, los robots no necesitarán estar equipados con muchos sensores, sino únicamente los necesarios para realizar su trabajo de manera segura.

Robótica en la nube es todavía un proyecto en desarrollo, debido a la falta de estándares en el desarrollo de sistemas robóticos. Afortunadamente existen en Internet distintas plataformas para el desarrollo de software para robots, las cuales pretenden convertirse en el estándar para la robótica.

En este trabajo se mencionan algunas de las plataformas para desarrollo de sistemas robóticos y se resaltan sus principales características. Entre ellas destaca ROS (*Robot Operating System*), el cual tiene una gran aceptación por parte de la industria, investigadores y entusiastas alrededor del mundo debido a su filosofía de no reinventar la rueda y reutilizar el código. Por estas razones se hace uso de ROS para el funcionamiento de los robots utilizados en este proyecto.

Por otra parte, se ensamblan dos robots, uno de armado propio y otro de distribución comercial (Turtlebot 3 Burger), para llevar a cabo el sistema de interacción entre ellos. El robot propio será de características limitadas y se construirá con base en componentes comerciales, es decir, solo tendrá capacidades básicas para desplazarse y evitar colisiones,

mientras que el robot de experimentación comercial posee una mayor capacidad de sensores entre los cuales destaca un sensor laser para medir distancias a su alrededor y con el cual se generará y compartirá información para que el otro robot haga uso de ella.

El sistema de interacción consiste en censar un ambiente controlado a través del robot comercial, con esta información se construirá un mapa que a su vez permita generar trayectorias dentro del entorno. Posteriormente se compartirá la ruta hacia el robot de ensamble propio para que este se desplace dentro del área censada de manera segura, logrando desempeñar una tarea que por sí mismo no podría realizar.

Marco Antonio Aguilar Tadeo

ABSTRACT

In this work a robot interaction system is introduced to show the concept of cloud robotics and its benefits for robot communication.

The current research in Cloud robotics aims to provide robots with the benefits of the cloud so their functions and performance are leveraged. Cloud robotics can be described as a social network where robots can share, store and process information that can be "learned" to perform new tasks according to their design and features.

Another objective of cloud robotics is the reduction of robot´s manufacturing cost by reducing the number of the sensors needed to work. Generating and sharing information will allow that robots have only the necessary sensors to perform their work safely.

Cloud robotics is a project in the development stage due to the lack of standards in robotic development systems. Fortunately, different robot software development platforms on Internet aim to become standard for robotics.

In this work some platforms for robotic development systems are summarized and their main characteristics are highlighted. Among them, ROS (Robot Operating System) stands out since it has great acceptance by the industry, researchers and enthusiasts around the world due to its philosophy of not reinventing the wheel and reusing code. For these reasons, ROS was used to control this project robots.

On the other hand, two robots were used to create the interaction system; one robot was designed and manufactured by our research group and the other one was a commercial robot (Turtlebot 3 Burger). Our robot was built using commercial components and its functions were limited to move and avoid collisions; whilst, the commercial robot had a larger number of features and sensors. The most important feature, in the commercial robot, was the laser sensor to measure distances around it and generate the information that was shared and used by our robot.

The interaction system consists on using the commercial robot to scan a controlled environment and use this information to build a map that allows the creation of trajectories.

Subsequently, the route will be share to our robot and it will safely move in the scanned area achieving a task that it cannot do by itself.

Marco Antonio Aguilar Tadeo

Índice de contenido

Índice de figuras……..……………………………………………………………...iv
Índice de tablas…………………………………………………………………………vii
Lista de acrónimos…..……………………………………………………………viii
Capítulo 1: Introducción 1
1.1. Antecedentes..1
1.2. Planteamiento del problema 2
1.3. Delimitación 3
1.4. Objetivos..4
1.4.1. Objetivo general 4
1.4.2. Objetivos específicos 4
1.5. Hipótesis 5
1.6. Justificación..5
1.7. Metodología..6
1.8. Publicaciones derivadas de este trabajo 7
1.9. Organización de la tesis..8
Capítulo 2: Conceptos fundamentales de robótica móvil 9
2.1. Robótica móvil 9
2.2. Cinemática de un robot móvil 10
2.3. Plataformas de *software* para desarrollo de sistemas robóticos 13
2.3.1. Tabla comparativa..18
2.4. Robótica en la nube 21
2.4.1. R2R (Robot-to-Robot) 21
2.4.2. R2C (Robot-to-Cloud) 22
2.5. Robot Operating System (ROS) 23

2.5.1. Arquitectura....23
Capítulo 3: Ensamble, configuración y operación de los robots....29
3.1. Robot móvil de armado propio....29
3.1.1. Etapas de desarrollo del robot de armado propio....29
3.1.2. Primera etapa: Ensamble de chasis, actuadores y sensores....30
3.1.3. Segunda etapa: Programación en Arduino del control básico....31
3.1.4. Tercera etapa: Incorporación de Raspberry PI 3 e instalación y operación básica de ROS....34
3.1.5. Cuarta etapa: Ejecución de ROS para el control del robot....37
3.1.6. Quinta etapa: Pruebas de funcionalidad del robot....39
3.2. Ensamble y operación del Turtlebot 3 Burger....39
3.2.1. Ensamble del robot comercial....40
3.2.2. Instalación y configuración del *software*....41
3.2.3. Pruebas del Turtlebot 3 Burger....42
Capítulo 4: Diseño y desarrollo experimental....45
4.1. Descripción de la plataforma....45
4.2. Creación e interpretación de datos del modelo de espacio de trabajo....46
4.2.1. Creación de la bolsa de datos del Lidar y datos cinemáticos....46
4.2.2. Interpretación de la información de movimiento de translación lineal....47
4.2.3. Interpretación de la información de movimiento rotacional....49
4.3. Generación de trayectoria....51
4.4. Desarrollo del segundo *sketch* de Arduino para el seguimiento de trayectoria..53
4.5. Calibración de movimientos: lineal y rotacional....55
4.5.1. Calibración lineal....56
4.5.2. Calibración rotacional.....59

4.6. Seguimiento de trayectorias 61
4.6.1. Caso de estudio uno 62
4.6.2. Caso de estudio dos 70
4.7. Discusión de resultados 75
Capítulo 5: Conclusiones 77
5.1. Trabajo futuro 79
Anexo A. Material utilizado en la construcción del robot propio 80
Anexo B. Código desarrollado 82
Anexo C. Diagramas de conexión 93
Referencias 98

Índice de figuras

Figura 2.1: Robot móvil con referencia global y referencia local. Tomada de (Siegwart, Nourbakhsh, & Scaramuzza, 2011) 11
Figura 2.2: Robot móvil alineado con los ejes globales. Tomada de (Siegwart, Nourbakhsh, & Scaramuzza, 2011)..13
Figura 2.3: Diagrama de red robot-to-robot...22
Figura 2.4: Diagrama de red robot-to-cloud. 22
Figura 2.5: Estructura de un repositorio en ROS. 25
Figura 2.6: Estructuras y esquemas de conceptos. Se describen gráficamente algunos conceptos de ROS. a) Representa un servicio. b) Ilustra la estructura de un Nodo. c) Se muestran las partes que componen un paquete. d) Esquema de mensajes y temas. 26
Figura 2.7: Estructura de comunicación entre nodos..26
Figura 3.1: Diagrama de etapas de desarrollo del robot móvil…………………..………...29
Figura 3.2: Versión 1 del robot de elaboración propia. 30
Figura 3.3: Versión 2 del robot de elaboración propia. 31
Figura 3.4: Diagrama de flujo del primer *sketch* de Arduino. 33
Figura 3.5: Robot de elaboración propia ejecutando ROS sobre Ubuntu Mate. 35
Figura 3.6: Emulador 3D Gazebo. 36
Figura 3.7: Diagrama de flujo del primer *sketch* de Arduino para ROS. 38
Figura 3.8: Versión final del robot de elaboración propia. 39
Figura 3.9: Turtlebot 3 Burger. 40
Figura 3.10: Lista de Partes del Turtlebot 3 Burger. 41
Figura 3.11: Mapa generado con Turtlebot 3 en las oficinas del Edificio F del CU UAEM VM. 43
Figura 3.12: Diagrama de nodos y tópicos para generar un mapa con Turtlebot 3. 44
Figura 4.1: Diagrama de comunicación entre los robots, el máster y la nube………………45
Figura 4.2: Grafica de datos obtenidos del Turtlebot 3 para avance del robot. 49
Figura 4.3: Grafica de datos obtenidos del Turtlebot 3 para giro del robot. 50
Figura 4.4: Generación de trayectoria en la computadora “máster” con la herramienta rviz con el algoritmo DWA...52

Figura 4.5: Diagrama de flujo del segundo sketch de Arduino para ROS............................54
Figura 4.6: Calibración lineal de los robots. Secuencia 1 de 4. 56
Figura 4.7: Calibración lineal de los robots. Secuencia 2 de 4. 57
Figura 4.8: Calibración lineal de los robots. Secuencia 3 de 4. 58
Figura 4.9: Calibración lineal de los robots. Secuencia 4 de 4. 58
Figura 4.10: Calibración rotacional de los robots. Secuencia 1 de 4. 59
Figura 4.11: Calibración rotacional de los robots. Secuencia 2 de 4. 60
Figura 4.12: Calibración rotacional de los robots. Secuencia 3 de 4. 60
Figura 4.13: Calibración rotacional de los robots. Secuencia 4 de 4. 61
Figura 4.14: Mapas de los escenarios de pruebas de trayectorias. 62
Figura 4.15: Diagrama de nodos y tópicos para seguimiento de trayectoria en Turtlebot 3.63
Figura 4.16: Diagrama de nodos y tópicos para seguimiento de trayectoria en robot propio. 64
Figura 4.17: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 1 de 5. 65
Figura 4.18: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 2 de 5. 66
Figura 4.19: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 3 de 5. 67
Figura 4.20: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 4 de 5. 68
Figura 4.21: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 5 de 5. 69
Figura 4.22: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 1 de 5. 71
Figura 4.23: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 2 de 5. 72
Figura 4.24: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 3 de 5. 73
Figura 4.25: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 4 de 5. 74

Figura 4.26: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 5 de 5. 75

Índice de tablas

Tabla 2.1: Comparación entre distintas plataformas de software para el desarrollo de sistemas robóticos, destacando sus principales características. 19

Lista de acrónimos

BFL Por sus siglas en inglés *Bayesian Filtering Library*.

CBSD Desarrollo de *Software* Basado en Componente DSBC (Del inglés *Component Based Software Development*).

CU Centro Universitario.

DARPA Por sus siglas en inglés *Defense Advance Research Project Agency*.

DSL Lenguaje de dominio específico (Del inglés *Domain Specific Language*).

DWA Por sus siglas en inglés *Dynamic Window Approach*.

EEROS Por sus siglas en inglés *Easy, Elegant, Reliable, Open and Safe*.

FPGA Por sus siglas en inglés Field-Programmable Gate Array.

GB Gigabyte.

GPS Sistema de posicionamiento global (Del inglés *Global Positioning System*).

IaaS Por sus siglas en inglés *Infrastructure as a Service*.

IDE Entorno de Programación (Del inglés *Integrated Development Environment*).

IMU Sensor Inercial o Unidad de Medición Inercial (Del inglés *Inertial Measurement Unit*).

IoT Internet de las cosas (Del inglés *Internet of Things*).

IP Protocolo de Internet (Del inglés *Internet protocol*).

KB Kilobyte.

KDL Por sus siglas en inglés *Kinematics and Dynamics Library*.

LASER Por sus siglas en inglés *Light Amplification by Stimulated Emission of Radiation*.

LDS Por sus siglas en inglés *Laser Distance Sensor*.

LiDAR Por sus siglas en inglés *Laser Imaging Detection and Ranging*.

LiPo Litio y polímero.

MARS Por sus siglas en inglés *Mobile Autonomous Robot Software*.

MEMS Por sus siglas en inglés *Micro Electro Mechanical Systems*.

MOOS Por sus siglas en inglés *Mission Oriented Operating Suite*.

MRDS Por sus siglas en inglés *Microsoft Robotics Developer Studio*.

OCL Por sus siglas en inglés *Orocos Components Library*.

OpenCV Por sus siglas en inglés *Open Source Computer Vision Library*.

Orocos Por sus siglas en inglés *Open Robot Control Software u Open Real-time Control Services*.

PaaS Por sus siglas en inglés *Platform as a Service*.

PWM Modulación por Ancho de Pulsos (Del inglés *Pulse Width Modulation*).

R2C Por sus siglas en inglés *Robot-to-Cloud*.

R2R Por sus siglas en inglés *Robot-to-Robot*.

RaaS Por sus siglas en inglés *Robot as a Service*.

ROCK Por sus siglas en inglés *The Robot Construction Kit*.

ROS Sistema Operativo robot (Del inglés *Robot Operating System*).

RTT *Real-Time Toolkit*.

SD Por sus siglas en inglés *Secure Digital*.

SLAM Por sus siglas en inglés *Simultaneous Localization and Mapping*.

SO Sistema operativo.

TCP	Por sus siglas en inglés *Transmission Control Protocol*.
UAEM	Universidad Autónoma del Estado de México.
VM	Valle de México.

Capítulo 1: Introducción

1.1. Antecedentes

La robótica es un campo de investigación tecnológica que ha tenido grandes avances en la última década, sin embargo, para que los robots realicen tareas complejas requieren de costosos recursos, esto se debe a que demandan capacidades de procesamiento de alta complejidad para que trabajen autónomamente. Al igual que las personas usan las redes de comunicación para compartir recursos, la llamada robótica en la nube (*Cloud Robotics*) permitiría que los robots hicieran uso de dichos recursos alojados en la nube, interactuando a través de la misma y creando una base de conocimiento que podrán compartir.

Como primer antecedente de robótica en la nube se tiene el proyecto Mercurio (Goldberg, 1994) (en inglés *The Mercury Project*), en los inicios de los años 90 y al popularizarse la *World Wide Web* y los protocolos de Internet (IP), en 1994 logró conectar el primer robot industrial a la web con una interfaz intuitiva que permitía operar el mismo mediante cualquier navegador de Internet (tele-operación), dando inició el concepto de *Networked Robotics* que junto al de *Cloud Computing*, dan paso a lo que hoy es *Cloud Robotics*.

Actualmente se están poniendo en práctica muchos servicios basados en el concepto de “la nube”, por ejemplo, en servicios de alojamiento de datos (documentos y fotografías) proporcionando la facilidad de que se acceda de forma remota y universal a estos archivos. Otro ejemplo de servicios en la nube son las redes sociales que permiten la comunicación directa y grupal de personas en cualquier lugar del mundo, también es posible el uso de recursos de procesamiento de alta demanda, desde procesadores de texto hasta sistemas de cálculo para actividades específicas. Por todo lo anterior, los sistemas basados en la nube han venido a cambiar la forma en la que se accede y procesa la información.

El Instituto Nacional de Estándares y Tecnología (NIST) (Mell & Grance, 2011) define *Cloud Computing* como: "*un modelo que permite el acceso a la red bajo demanda y con ubicuidad práctica para compartir recursos configurables (por ejemplo: servidores, almacenamiento, redes, aplicaciones y servicios) que pueden ser rápidamente accesibles y*

suministrados y liberados con mínimo esfuerzo de gestión o interacción del proveedor de servicios".

Si de una forma semejante a las redes sociales se construye una red de servicios disponibles para los robots, estos podrían comunicarse, coordinarse, informarse, transmitir y recibir información que trabajando como una unidad autónoma no se podría tener.

Por lo tanto, si han sido muy importantes los recursos en la nube para las personas en la actualidad ¿Cómo se verían beneficiados los robots con un concepto similar?

En 2010, James Kuffner propuso el término *Cloud Robotics* y mencionó varias ventajas y potenciales beneficios de este nuevo campo (Kuffner, 2010). Los principales beneficios de robótica en la nube son disminuir el costo de fabricación de los robots y permitir que operen de manera autónoma pero coordinada. Actualmente un robot requiere de muchos sensores para realizar una tarea específica y autónoma, esto lo hace poco accesible a las personas o empresas, ya que deberían desembolsar cantidades importantes de dinero por un robot muy especializado, que frecuentemente solo puede desempeñar una tarea correctamente.

Antes de que el concepto de *Cloud Robotics* sea lo que hoy se entiende, se había propuesto una arquitectura de interacción entre robots, (Kumar, Rus, & Sukhatme, 2008) *Networked Robotics*. Se conceptualizaba como una interacción entre varios robots funcionando juntos y en coordinación, o de manera cooperativa con sensores, computadoras embebidas y usuarios humanos. Permitiendo además que múltiples robots y entidades auxiliares lleven a cabo tareas que rebasan los recursos de un solo robot.

La robótica en la nube tiene como principal objetivo crear un Internet para robots, es decir, una gran base de datos de información y aplicaciones para consulta mediante una conexión a Internet, logrando así adquirir habilidades que permitan al robot realizar tareas nuevas de manera correcta y coordinada.

1.2. Planteamiento del problema

El presente trabajo ilustra el concepto de robótica en la nube, tiene el objetivo de plantear cómo una infraestructura de *hardware* y *software* que ofrezca servicios en tiempo real,

permitiría transmitir capacidades entre robots de diferente complejidad con la meta de hacer más accesible la robótica. En este proyecto se hará uso de una plataforma de intercambio de información entre dos robots: un sistema robótico de bajo costo que mediante un sistema de comunicación basado en el protocolo de Internet sea capaz de aprovechar el modelo del ambiente generado por un robot de mayor capacidad y utilizar un algoritmo para seguir trayectorias en entornos controlados.

Dicha plataforma permitiría que, a través de servicios disponibles en Internet para robots, estos puedan potencializar sus capacidades a través del intercambio de información y recursos entre ellos. La robótica en la nube hecha realidad podría hacer que los robots fueran tan comunes como los teléfonos celulares de ahora.

Particularmente la plataforma que se plantea para mostrar el concepto de robótica en la nube va a consistir en dos robots móviles, uno comercial equipado con sensores y actuadores que le permiten modelar su entorno y otro de ensamble propio con capacidades limitadas, que compartirán información de modelado de ambientes controlados y recursos para la generación de trayectorias para la navegación en ambientes controlados.

1.3. Delimitación

Para el desarrollo de este trabajo, se hará uso de un robot móvil de armado propio, apoyado de una computadora para procesamiento de información y un micro controlador para dar la capacidad de tener sensores de seguridad y evitar colisiones. Además de un robot comercial (Turtlebot 3 Burger), equipado con sensores y actuadores, entre los que destaca un sensor tipo Lidar (del inglés *Laser Imaging Detection and Ranging*), para generar mapas de su entorno en 2D. En este proyecto no se busca ni manipular objetos ni alterar el entorno, el robot será un observador con el objetivo de aprender de su ambiente para así lograr trazar rutas evadiendo obstáculos.

En este proyecto se contemplan los siguientes puntos:

- **Capacidad del robot de ensamble propio:** el robot armado para este trabajo tendrá capacidades básicas de locomoción y contará con un sensor ultrasónico para evitar colisiones.

- ***Hardware* y *software* del robot de ensamble propio:** en cuanto a *hardware* se hará uso de una Raspberry Pi 3, Arduino, dos motorreductores, un sensor de distancia, una rueda libre y un chasis de acrílico como base de ensamble. En relación al *software*, se instalará el SO Ubuntu Mate y ROS (*Robot Operating System*) en su versión Kinetic Kame.
- **La precisión en el seguimiento de la trayectoria:** este concepto requiere de aumentar las capacidades del robot de armado propio, en esta etapa se destaca la economía de recursos para la construcción del robot, limitando sus capacidades naturales. Si se quiere aumentar la precisión bastará con aumentar la tecnología de motores, pues la información será la misma en todo caso. Por lo anterior, no se considera evaluar la precisión del seguimiento de la trayectoria.

Por lo que en este proyecto se busca ilustrar como la plataforma de comunicación entre robots permitiría aprovechar las capacidades entre los mismos.

1.4. Objetivos

1.4.1. Objetivo general

Hacer uso de una plataforma robótica que muestre el concepto de robótica en la nube. La plataforma consistirá en dos robots que se comunicarán para seguir una trayectoria con base en un mapeo de un entorno controlado. Uno de ellos tendrá capacidades de censado a través de un sensor Lidar, mientras que el otro robot tendrá capacidades básicas de movimiento.

1.4.2. Objetivos específicos

- Analizar las funcionalidades y características de las plataformas para el paradigma de robótica en la nube y hacer un estudio comparativo.
- Esquematizar los componentes de una plataforma de prueba para la comunicación entre robots, con el fin de ilustrar la interacción entre robots.
- Ensamble y configuración de un robot propio basado en el sistema operativo ROS.
- Puesta en marcha de un robot de investigación comercial.
- Configuración y mapeado de un ambiente controlado.

- Uso de herramientas basadas en ROS para la generación de trayectorias y evasión de obstáculos.
- Comunicación y transferencia de la trayectoria al robot de armado propio.

1.5. Hipótesis

La robótica en la nube otorgaría la capacidad de compartir información entre robots al contar con servicios accesibles automáticamente. Se propone que a través de una plataforma se muestre el beneficio del intercambio de datos para la realización de tareas.

El uso de la robótica en la nube permitirá:

- La construcción de robots con bajo costo, es decir, sin la necesidad de contar con un gran número de sensores y actuadores complejos, así como tarjetas de control o computadoras a bordo, sino únicamente los necesarios para su funcionamiento de manera correcta y segura.
- A robots de bajo costo, desempeñar nuevas tareas e incluso aprender otras aprovechando los beneficios que ofrece la infraestructura de robótica en la nube, como pueden ser consulta, almacenamiento, descarga y/o procesamiento de información.

1.6. Justificación

Los robots son cada vez más comunes en la vida de las personas, hasta hace no mucho tiempo se consideraban solo disponibles para actividades complejas y costosas. Por ejemplo, soldar, transportar, pintar, cortar, por mencionar algunas, pero recientemente con la mejora de dispositivos de procesamiento (microprocesadores, microcontroladores, FPGA, etc.) y dispositivos electromecánicos "MEMS" (*Micro Electro Mechanical Systems*), los robots comienzan a estar presentes en diversas actividades humanas. Desafortunadamente los robots todavía no son accesibles para todo el mundo, debido a su alto costo. Los robots requieren de *hardware* y *software* especial para realizar una tarea determinada, lo que los limita en funcionamiento y esta es otra razón para que los robots no se hayan convertido en la principal

opción para realizar diversas actividades. Otra dificultad en robótica es la falta de estándares que permitan unificar su control y manejo de información.

Recientemente se han venido desarrollando proyectos para estandarizar los procesos de diseño en robots. ROS es un ejemplo para destacar de estos desarrollos debido a su arquitectura de comunicación, la cual es modular y a su filosofía de código abierto. ROS tiene el concepto de distribuir los procesos a través de unidades que se comunican por medio de protocolo de internet haciendo posible que se convierta en la plataforma de un sistema basado en la nube para robots. Otro aspecto a resaltar es el crecimiento y la aceptación que tiene alrededor del mundo por parte de la industria, entusiastas e investigadores.

La robótica en la nube representa ahorros significativos en *hardware* y *software* para el desarrollo de robots, ya que permite a los mismos acceder a una gran base de datos con información relevante acerca de su entorno. Esto permitiría a un robot no solo ahorrar en el aspecto económico, sino que les daría la capacidad de llevar a cabo más de una tarea.

Además, brinda la capacidad de permitir a los robots comunicarse entre sí, permitiendo intercambiar, subir y descargar conocimientos. Esto significa que si un robot necesita realizar una tarea que desconoce, puede descargar las instrucciones para llevar a cabo dicha labor, además de tener la posibilidad de pedir ayuda en caso de no contar con los recursos de *hardware* o *software* necesarios para ejecutar una acción.

Una de las principales ventajas que ofrece la robótica en la nube, es la de disminuir costos en la fabricación de robots por lo que no será necesario que dispongan de grandes capacidades de procesamiento, memoria o sensores para su desempeño, la comunicación con la nube permitirá a los robots poder realizar más de una labor.

1.7. Metodología

Para el desarrollo del proyecto se plantea lo siguiente:

- **Recursos disponibles de robótica en la nube**. Estudio y comprensión del tema de robótica en la nube (*Cloud robotics*).

- **Establecer especificaciones del sistema de robótica en la nube**. Se analiza y comprende el funcionamiento de robótica en la nube, características de *hardware* y *software* que le permiten a estas plataformas brindar su servicio.
- **Elección de *hardware***. Con base en la información obtenida y las pruebas que se realicen con el *software* de robótica en la nube se podrá tener una idea clara de las especificaciones técnicas de *hardware* necesario.
- **Desarrollo del *hardware* y *software* del proyecto**. Con todos los datos recopilados hasta este punto será posible realizar el ensamble de los robots para su funcionamiento y comunicación con la nube, haciendo uso de ROS como base de comunicación. Para el caso del robot comercial, se realizará el ensamble y conexión de chasis, sensores y actuadores. El robot de armado propio se ensamblará por etapas para lograr una correcta locomoción del mismo.
- **Calibración y pruebas**. Mediante un robot de experimentación comercial (Turtlebot3 Burger), se recopilará información de un entorno controlado mediante sus sensores y actuadores. Después se construirá un mapa del entorno a partir de los datos obtenidos y posteriormente se trazarán trayectorias para compartirse hacia el robot de armado propio. El robot propio será capaz de transitar en el entorno censado por el Turtlebot 3 de manera segura.

1.8. Publicaciones derivadas de este trabajo

Durante el desarrollo de esta tesis se realizó el artículo de revista indizada en Latindex:

"Plataformas de *Software* para el Desarrollo de Sistemas Robóticos" publicado en la revista de Programación Matemática y Software (PMS) Vol. 10, Núm. 3, octubre 2018, ISSN: 2007-3283.

Además, se realizaron las siguientes ponencias en el CU UAEM VM:

"Robótica en la Nube: Actualidad y Perspectivas", presentado en el Congreso Internacional Multidisciplinario del CU Valle de México 2016 IMCXX-CUVM 2016.

“*Robot Operating System* en un Ambiente Virtual” y “ROS: *Robot Operating System* para la Implementación de un Robot Móvil”, presentados en Coloquios del Centro Universitario UAEM Valle de México durante 2017.

1.9. Organización de la tesis

La organización de la tesis es la siguiente:

En el capítulo 2 se darán los antecedentes de la robótica móvil, así como una clasificación de la misma de acuerdo con la generación a la que pertenecen. Se mencionarán algunas plataformas disponibles en Internet para el desarrollo de *software* para robótica, resaltando sus principales características mediante una tabla. En este capítulo también se describe la robótica en la nube y sus principales características. Finalmente, se describirá el Sistema Operativo Robot (ROS), el cual es utilizado en ambos robots del proyecto de esta tesis.

El capítulo 3 presentará la forma en que se elaboró el robot móvil propio, así como las características, ensamble y puesta en marcha del robot comercial.

En el capítulo 4 se mostrarán los resultados de los experimentos de operación de la plataforma de interacción entre los robots. También se presentarán los resultados de compartir información entre los robots.

Las conclusiones se presentarán en el capítulo 5, así como también las líneas o trabajos de investigación a futuro.

Capítulo 2: Conceptos fundamentales de robótica móvil

2.1. Robótica móvil

Un robot es cualquier máquina de accionamiento automático que sustituye el trabajo humano y puede o no realizar sus funciones de forma semejante a la humana, en general, la robótica es la disciplina de la ingeniería que se ocupa del diseño, construcción y operación de robots (Hu, Tay, & Wen, 2012). Los robots son una realidad y existen principalmente en la industria de manufactura para realizar tareas difíciles, repetitivas y peligrosas, como son: pintar, cortar y soldar, por mencionar sólo algunas. Hay tareas complejas, como aquellas que requieren la intervención directa del ser humano, se busca que los robots igualen o superen las capacidades de las personas y por lo tanto que sean completamente autónomos.

Existen diversas maneras de clasificar a los robots, puede ser de acuerdo a su generación o a su inteligencia y posteriormente a su mecánica de funcionamiento o también por la tarea que se encomienda al robot.

De acuerdo con (Tapia García & López Hernández, 2017), la clasificación por generación es:

1.- Robots *Play-back*: Los cuales ejecutan una rutina almacenada.

2.- Robots controlados por sensores: Ejecutan movimientos con base en información obtenida a través de sus sensores.

3.- Robots controlados por visión: Interactúan con el entorno basándose en información obtenida por visión artificial.

4.- Robots controlados adaptablemente: Son aquellos que tienen la capacidad de reprogramarse con base en información obtenida a través de sus sensores.

5.- Robots con inteligencia artificial: Utilizan métodos de inteligencia artificial para resolución de problemas y toma de decisiones.

6.-Los robots médicos: Principalmente prótesis y robots tele-operados.

7.-Los androides: Robots que se asemejan a los seres humanos.

8.- Los robots móviles: Cuentan con patas, ruedas u orugas que les brindan capacidad de locomoción. Tienen la capacidad de utilizar la información captada por sus sensores en tiempo real y se ocupan principalmente en la industria. También se utilizan robots de este tipo para actividades de vigilancia, agricultura, militares, exploración y/o investigación (terrestre, subterránea, marina, submarina, aérea y espacial), entre otras.

2.2. Cinemática de un robot móvil

La cinemática es la descripción del movimiento, sin considerar en cuenta las causas que lo producen. En robótica móvil es vital comprender el funcionamiento mecánico para elaborar un robot y para facilitar la programación del control de *hardware*.

Los robots móviles no son los más complejos sistemas mecánicos dentro de la robótica, existen otros con mayor dificultad de acción, por ejemplo, un brazo robótico que manipula objetos tiene más de un eje de movimiento, en comparación con un robot móvil que solo posee dos.

Es importante conocer el entorno del robot ya que de este depende el movimiento que pueda realizar, los caminos que pueda seguir y la manera de posicionarse dentro de su ambiente.

La estimación de posición de un robot móvil que se desplaza en su ambiente de trabajo es difícil de calcular debido a diversos factores como pueden ser el derrape, fallas de movimiento o la sincronización con el tiempo del avance, retroceso o giro.

En un robot de ruedas, cada una de ellas representa la posibilidad de movimiento, pero también de restricción del mismo. Afortunadamente esta parte ya ha sido estudiada (Siegwart, Nourbakhsh, & Scaramuzza, 2011) y existen modelos matemáticos para describir el comportamiento de este tipo de robots.

En la Figura 2.1 se representa la relación entre una referencia global y una referencia local del robot. Donde la referencia global se da por $\{X_I, Y_I\}$ mientras la referencia local se especifica a partir de un punto *P* el cual se establece en el chasis del robot y desde este punto se definen los ejes $\{X_R, Y_R\}$.

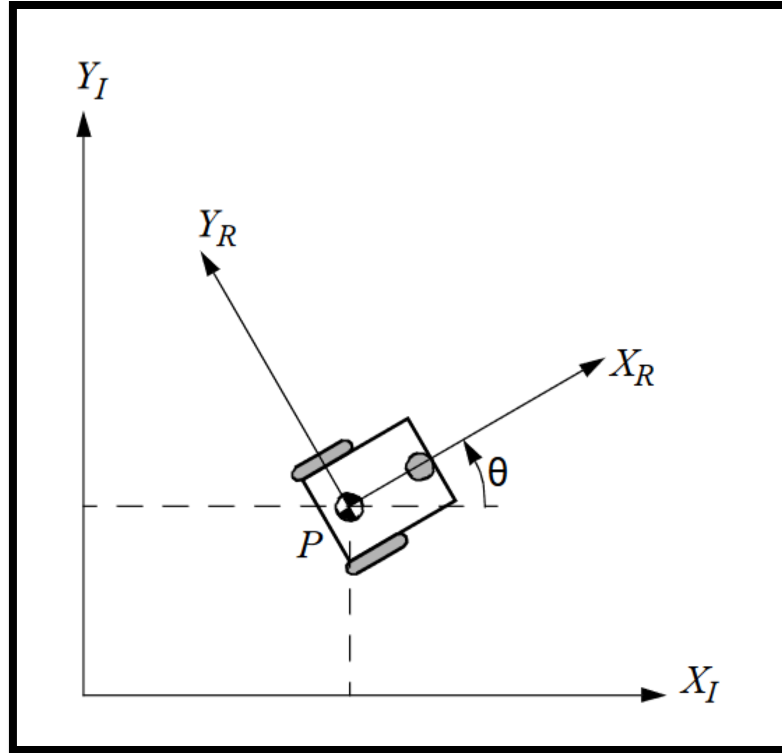


Figura 2.1: Robot móvil con referencia global y referencia local. Tomada de (Siegwart, Nourbakhsh, & Scaramuzza, 2011)

La posición de *P* se especifica por las coordenadas *X*, *Y* y la diferencia angular representada por θ entre la referencia global y la local. Por lo tanto, se puede representar la pose del robot como un vector de estos elementos, aclarando con el subíndice *I* que la base es la referencia global.

$$\xi_I = \begin{bmatrix} X \\ Y \\ \theta \end{bmatrix} \tag{2.1}$$

Para describir el movimiento del robot, será necesario esquematizar el movimiento a lo largo de los ejes de la referencia global para moverse a lo largo de los ejes de referencia locales. Esto se logra utilizando la matriz de rotación.

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Esta matriz se puede utilizar para dibujar el movimiento en la referencia global $\{X_I, Y_I\}$ al movimiento en términos de la referencia local $\{X_R, Y_R\}$. Esta operación se denota por $(\theta)\dot{\xi}_I$ porque el cálculo de esta operación depende del valor de θ:

$$\dot{\xi}_R = R\left(\frac{\pi}{2}\right)\dot{\xi}_I \tag{2.3}$$

En la Figura 2.2 donde el robot se encuentra alineado en los ejes globales y locales, es fácil calcular la matriz de rotación ya que $\theta = \frac{\pi}{2}$

$$R\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Dada cierta velocidad $(\dot{x}, \ \dot{y}, \ \dot{\theta})$ en la referencia global, se pueden calcular los componentes de movimiento a lo largo de los ejes locales del robot X_R y Y_R. En este caso debido al ángulo del robot, el movimiento a lo largo de X_R es igual a $\dot{y}$ mientras el movimiento a lo largo de Y_R es $-\dot{x}$, por lo tanto:

$$\dot{\xi}_R = R\left(\frac{\pi}{2}\right)\dot{\xi}_I = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ -\dot{x} \\ \dot{\theta} \end{bmatrix} \tag{2.5}$$

Con base en estas ecuaciones es posible comprender la cinemática de un robot móvil y así poder planificar el control del mismo a nivel *software*.

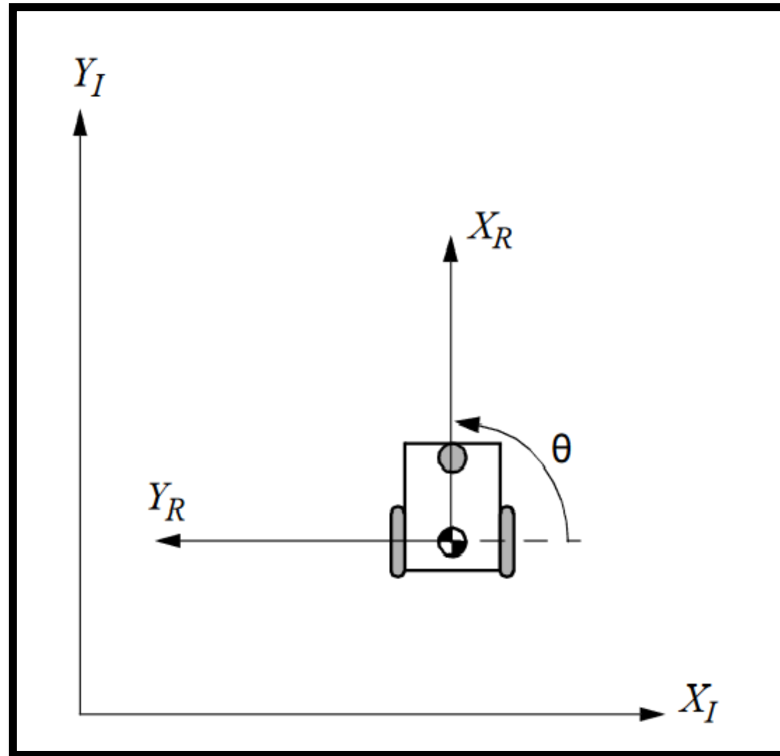


Figura 2.2: Robot móvil alineado con los ejes globales. Tomada de (Siegwart, Nourbakhsh, & Scaramuzza, 2011)

A través de este modelo se observan las variables de posición, orientación y su velocidad que son descriptivas de la cinemática de un robot móvil. En los recursos que proporciona ROS para la descripción cinemática de un robot ya contempla esta estructura de datos. Se destaca lo anterior para ilustrar como ROS contempla esta información facilitando el modelado y manejo de robots.

2.3. Plataformas de *software* para desarrollo de sistemas robóticos

Actualmente se desarrollan diferentes sistemas para control de robots, los cuales utilizan la filosofía del “Desarrollo de Software Basado en Componente” (CBSD por sus siglas en inglés). En CBSD se favorece la reutilización de código para facilitar la creación de nuevos programas (Alonso, et al., 2012), dichos sistemas van desde desarrollos sencillos y básicos, hasta los complejos. En muchas ocasiones al iniciar un proyecto de robótica se puede comenzar sin apoyarse en trabajos previos, no obstante, hay que considerar que actualmente existe trabajo generado por otras investigaciones. Este es el caso de ROS (*Robot Operating*

System), el cual es una plataforma de desarrollo de *software* que provee una serie de ejemplos y librerías que simplifican la creación de aplicaciones para robots con características de *hardware* diferentes (Bravo Sánchez & Forero Guzmán, 2012), de esta forma se pueden reutilizar códigos que están probados y que se encuentran disponibles para su consulta y descarga, alcanzando más rápidamente los objetivos del trabajo, ganando tiempo para enfocarse en mejorar las aplicaciones de robótica.

Una de las ventajas que ofrecen las plataformas de *software* para el desarrollo de sistemas robóticos, es la de poder almacenar y compartir las investigaciones (proyectos de robótica) llevados a cabo por instituciones públicas y privadas, así como por investigadores, estudiantes y aficionados. Esto permite que muchos trabajos puedan ser retomados por alguien más y no se pierdan los avances generados, sin importar si el propósito del proyecto es de investigación, comercial o lúdico. En las plataformas de uso libre, incluso es posible contribuir a mejorar el proyecto mismo. Por ejemplo, si se encuentra un error en el código se hace una copia o clonación del sistema (*fork* en inglés), y una vez mejorado se envía a los administradores para su evaluación (*pull request* en inglés), en caso de hacer una mejora importante y significativa al sistema, se agregan los cambios generados a la versión más reciente de la plataforma de *software*.

La construcción de robots requiere de una gran inversión económica y de tiempo, razón por la cual no han alcanzado una mayor popularidad. No obstante, existe una gran demanda comercial de ellos donde el costo de materiales y componentes son un elemento de inversión importante. Además, los algoritmos para darles capacidades de operación y de respuesta desde bajo nivel (manejo de actuadores y sensores) hasta alto nivel (inteligencia: toma de decisiones) son obra de cada diseñador, lo que dificulta aún más su implementación. Por tal motivo, no contar con un estándar de desarrollo de *software* para robots es parte de este problema. Afortunadamente, han surgido distintos proyectos de plataformas de *software* para el desarrollo de sistemas robóticos, los cuales buscan convertirse en un estándar para el diseño de robots, reduciendo la inversión de tiempo, dinero e investigación.

Distintos proyectos alrededor del mundo se están desarrollando, todos ellos con diferentes características como son lenguajes de programación, propósitos (reconocimiento de voz,

robots móviles, agarre de objetos, mapeo, etc.) o compatibilidad (*hardware*, sistemas operativos, aplicaciones), a continuación, se describen varios de ellos.

ARTOO. Es un *framework* para robótica, drones e Internet de las cosas (IoT). Proporciona un Lenguaje de Dominio Específico (DSL por sus siglas en inglés) simple pero potente para robots. ARTOO está basado en SINATRA (Mizerany, s.f.), el cual es un DSL para la rápida creación de aplicaciones web con un mínimo esfuerzo, mediante el lenguaje de programación Ruby e incluso toma código prestado de SINATRA. Artoo se encuentra disponible en inglés, fue lanzado en "Los Ángeles Ruby Conference 2013", es una infraestructura de *software* para robots de código abierto que trabaja con el lenguaje de programación RUBY. Trabaja con los sistemas operativos Linux, Windows y Mac OS X. Ofrece la posibilidad de conectar varios dispositivos de *hardware* de manera sencilla (por ejemplo: ARDrone, Raspberry Pi, joystick, teclado, Arduino, motores, servomotores, ledes, sensores análogos, entre otros) (The Hybrid Group, 2014).

Carmen Robot Navigation Toolkit. Es un proyecto perteneciente a Christopher Fedor y Reid Simmons de la Universidad Carneige Mellon, comenzó en 1991 y trabaja bajo el sistema operativo Linux, está escrito en lenguaje de programación C, pero ofrece compatibilidad con lenguaje JAVA. Es una colección de *software* para control de robots móviles de código abierto. Disponible en inglés, Carmen está diseñado para proveer servicios de navegación como: control de base y sensores, registro, esquivar obstáculos, localización, planear rutas y mapeo. El proyecto ha sido patrocinado por el programa MARS (*Mobile Autonomous Robot Software*) de DARPA (*Defense Advance Research Project Agency*) (Fedor & Simmons, s.f.).

EEROS. (*Easy, Elegant, Reliable, Open and Safe*) Es una infraestructura de código abierto para desarrollo de *software* para robots, opera bajo el sistema operativo Linux y con el lenguaje de programación C++. Se desarrolla por NTB Universidad de Tecnología en Suiza desde el año 2012. EEROS consta de tres sistemas conectados, el Sistema de Control, el Secuenciador y el Sistema de Seguridad, los cuales trabajan juntos para desarrollar rápidamente el nuevo *software* de robótica y minimizar errores potencialmente peligrosos. EEROS se encuentra aún en desarrollo, sin embargo, ha tenido buenos resultados en sus versiones prototipo, está disponible en inglés (Rüf Stiftung, 2014).

Microsoft Robotics Developer Studio. (MRDS). Es un Proyecto de Microsoft que inició en 2006, ofrece un entorno de simulación 3D basado en el motor de simulación física AGEIA PhysX, un lenguaje de programación visual y soporte en tiempo de ejecución. Opera bajo el sistema operativo Windows 7 y su uso es libre y de código abierto. Está desarrollado sobre el entorno .NET y soporta lenguajes de programación como VB.NET, Python, Visual Basic, VPL o C#. Disponible en inglés (Microsoft, 2017) (Alcalá Tomás, Celorio Aponte, & Montoya Álvarez, 2013).

miniBloq. Es un proyecto perteneciente a Julian U. Da Silva Gillic, desarrollado en el Instituto Wyss de la Universidad de Harvard desde 1993, está compilado con C++. Es un entorno de programación gráfica de código abierto que tiene como principal objetivo la robótica educativa a través de facilitar la programación y su aprendizaje en personas con pocos conocimientos en informática, además de contar con su robot llamado "Root". Opera bajo el sistema operativo Windows y Linux, este último con algunas limitaciones, cuenta con una interfaz de usuario avanzada donde se puede programar con los lenguajes de programación Python, JavaScript y Swift (da Silva Gillig, 2016). Disponible en inglés y con tutoriales en español.

MOOS. Por sus siglas en inglés *Mission Oriented Operating* Suite, es una plataforma construida en C++ para investigación en robótica, se comenzó en 2001 y su creador es Paul Newman del Departamento de Ingeniería Oceanográfica del Instituto de Tecnología de Massachusetts. MOOS se define como un conjunto de capas que se comunican entre sí y que en conjunto crean aplicaciones robustas. La última versión es llamada MOOS v10, la cual opera en los sistemas operativos Linux y Mac OS X (Newman P. , s.f.) (Newman P. M., 2008). Es compatible con los lenguajes de programación Matlab y Java, está disponible en idioma inglés.

MyRobotLab. Es un proyecto de código abierto para robótica y control de máquinas, funciona con Java y con los sistemas operativos Windows, Linux y Mac. Ofrece servicios para visión artificial, reconocimiento de voz, control de motores y servomotores y comunicación con microcontroladores. Disponible en inglés (myrobotlab, s.f.).

OpenCV. (*Open Source Computer Vision Library*). Es una librería de código abierto creada por Intel en el año 1999, está orientada a la visión artificial y aprendizaje de máquinas. Opera bajo los sistemas operativos Linux, Mac, Windows y Android. Soporta los lenguajes de programación Python, Java, C/C++ y MATLAB. Cuenta con tutoriales en inglés, español y japonés (OpenCV team, 2017).

Orca. Es una plataforma para desarrollo de sistemas robóticos basados en componentes, su uso es libre y opera de manera completa en los sistemas operativos Linux, Windows y el sistema operativo Mac aún se encuentra en fase experimental. En un principio Orca era parte de otra plataforma llamada OROCOS, sin embargo, nunca se integró al proyecto y en 2005 inició Orca con el objetivo de estimular la reutilización código para el continuo progreso en la robótica. El proyecto está construido con lenguaje de programación C++ y para compilar se pueden utilizar los lenguajes C ++, Java, Python, PHP, C #, Visual Basic, Ruby y Objective C, pero solo para escribir partes del código (Brooks, Kaupp, Makarenko, & Moser, s.f.) (Makarenko Alexei, 2006). Disponible en inglés.

Orocos. (*Open Robot Control Software u Open Real-time Control Services*) Es un proyecto de código abierto enfocado principalmente al control de robots y máquinas en tiempo real, fue una idea de Herman Bruyninckx en el año 2000 y patrocinado por la Unión Europea. Trabaja con el sistema operativo Linux y utiliza el lenguaje de programación C++, se encuentra disponible en inglés. Orocos provee a C++ cuatro librerías con las que trabaja RTT (*Real-Time Toolkit*), OCL (*Orocos Components Library*), KDL (*Kinematics and Dynamics Library*) y BFL (*Bayesian Filtering Library*) (Bruyninckx, s.f.).

Player. Es desarrollado por un equipo internacional de investigadores en robótica, su propósito es crear *software* libre para hacer investigación en sistemas de robots y sensores. Fue creado por Brian Gerkey, Richard Vaughan y Andrew Howard en la Universidad del sur de California en el año 1999. Opera bajo los sistemas operativos Linux, Solaris, BSD y Mac OSX (Darwin). El proyecto Player es ampliamente utilizado e incluye los paquetes de software “Stage” y “Gazebo” los cuales son simuladores en 2D y 3D respectivamente. Tiene compatibilidad con una gran variedad de robots comerciales y *hardware* (sensores y actuadores). Es compatible con los lenguajes de programación C++, Tcl, Java, Python y

cualquier lenguaje que admita sockets TCP. Disponible en inglés (Gerkey, Vaughan, & Howard, 2014).

ROCK. (*The Robot Construction Kit*) Es una plataforma para desarrollo de sistemas robóticos de código abierto, basado en RTT de Orocos. Proporciona todas las herramientas necesarias para configuración y ejecución de sistemas robóticos. Inicialmente se desarrolló en el DFKI Centro de Innovación Robótica y su principal colaborador es la Universidad Católica Leuven. Trabaja con sistema operativo Linux (Ubuntu y Debian) y está disponible en inglés. Es compatible con los lenguajes de programación C++ y Ruby (Center, s.f.).

ROS (*Robot Operating System*) es una infraestructura para el desarrollo de *software* para robots, provee una serie de ejemplos, librerías, herramientas y convenciones que simplifican la creación de aplicaciones para robots con características de *hardware* diferentes. La filosofía de ROS es hacer *software* que pueda ser reutilizado en otros robots, es decir, lograr que el código que se crea para un robot, pueda ser compartido y utilizado en otros robots para así ahorrar tiempo y esfuerzo en el desarrollo de aplicaciones. ROS está bajo la licencia *open source*. Se encuentra disponible en inglés, alemán, francés, italiano, japonés, coreano, portugués, chino simplificado y español. ROS surgió originalmente en el año 2007 bajo el nombre de "switchyard", fue desarrollado por el Laboratorio de Inteligencia Artificial de Stanford. A partir del año 2008 el desarrollo del proyecto se lleva a cabo en el Instituto de Investigación de Robótica Willow Garage, en California, Estados Unidos (Willow Garage, s.f.). ROS opera con el sistema operativo Linux y soporta los lenguajes de programación C, C++, Python y Java, este último aún en fase experimental.

2.3.1. Tabla comparativa

Con base en la información previamente descrita para algunas plataformas de *software* para el desarrollo de sistemas robóticos, se presenta la Tabla 2.1 destacando las principales características de cada una.

Tabla 2.1: Comparación entre distintas plataformas de software para el desarrollo de sistemas robóticos, destacando sus principales características

Plataforma de desarrollo	**Tipo de licencia.**	**SO con que trabaja.**	**Lenguajes de programación soportados.**	**Idiomas soportados.**	**Servicios que ofrece.**
ARTOO.	*Open Source.*	Linux, Windows y Mac OS X.	Ruby.	Inglés.	Proporciona un DSL sencillo pero potente para robots, drones e IoT. Compatible con diferentes dispositivos de *hardware*.
Carmen Robot Navigation Toolkit	*Open Source*	Linux.	C y Java.	Inglés.	Brinda servicios de navegación para robots móviles.
EEROS	*Open Source*	Linux.	C++.	Inglés.	Proporciona un *framework* y una arquitectura viables para robots en educación e industria.
Microsoft Robotics Developer Studio	*Open Source*	Windows.	VB.NET, Python, Visual Basic, VPL o C#.	Inglés.	Entorno de simulación 3D, un lenguaje de programación visual y soporte en tiempo de ejecución.
miniBloq	*Open Source*	Windows y Linux (este último con algunas limitaciones)	Python, JavaScript y Swift.	Inglés y español.	Robótica educativa y un lenguaje de programación sencillo para niños y adultos, además de un robot llamado “Root”.
MOOS	*Open Source*	Linux y Mac OS X.	C++, Matlab y Java.	Inglés.	Brinda servicios de navegación para robots móviles marinos.
MyRobotLab	*Open Source*	Linux, Windows y Mac OS.	Java.	Inglés.	Visión artificial, reconocimiento de voz, control de motores y comunicación con microcontroladores.
OpenCV	*Open Source*	Linux, Windows, Mac OS y Android.	Python, Java, C/C++ y MATLAB.	Inglés, español y japonés.	Visión artificial y aprendizaje de máquinas.
Orca	*Open Source*	Linux, Windows y el SO Mac OS aún en fase de prueba.	C++, Java, Python, PHP, C #, Visual Basic, Ruby y Objective C.	Inglés.	Repositorio para estimular la reutilización de software y compartir librerías de alto nivel.
Orocos	*Open Source*	Linux.	C++.	Inglés.	Control de robots y máquinas en tiempo real.

Plataforma de desarrollo	Tipo de licencia.	SO con que trabaja.	Lenguajes de programación soportados.	Idiomas soportados.	Servicios que ofrece.
					Además cuatro librerías para C++.
Player	*Open Source*	Linux, Solaris, BSD y Mac OSX (Darwin).	C++, Tcl, Java, Python y cualquier lenguaje que admita sockets TCP.	Inglés.	Paquetes de software “Stage” y “Gazebo” los cuales son simuladores 2D y 3D respectivamente. Compatibilidad con robots y hardware comercial.
ROCK	*Open Source*	Linux.	C++ y Ruby.	Inglés.	Plataforma para desarrollo de sistemas robóticos con todas la herramientas necesarias para configuración y ejecución de los mismos.
ROS	*Open Source*	Linux y Mac OS X. Windows con algunos errores.	C, C++, Python. Java aún en modo experimental.	Inglés, alemán, francés, italiano, japonés, coreano, portugués, chino simplificado y español.	Infraestructura para desarrollo de software para robots, repositorios, localización y mapeo simultaneo, simulación 2D y 3D, identificación de objetos, robots móviles, control, planificación de rutas, agarre de objetos, visión artificial y reconocimiento facial y de gestos. Gran compatibilidad con hardware.

Dada la información anterior se puede resumir que todas las plataformas de *software* para desarrollo de sistemas robóticos mencionadas ofrecen *open source*. El sistema operativo que predomina es Linux, con excepción de MRDS el cual trabaja con Windows, los sistemas operativos Mac y Android también figuran como una opción para algunas plataformas. La diversidad en lenguajes de programación es amplia, pero los más utilizados son C++, Python y Java. El idioma que aparece disponible para todas las plataformas es el inglés y en muchos casos es el único idioma soportado. Es importante analizar los objetivos que se persiguen al desarrollar un proyecto para poder hacer una correcta elección de la plataforma de desarrollo de *software*, además de evaluar los conocimientos del desarrollador en cuanto a lenguajes de programación, sistema operativo e idioma.

2.4. Robótica en la nube

Debido a su alto costo de desarrollo e implementación, los robots no han logrado colocarse como un producto comercial y al alcance de todos. La creación de un robot requiere de diseño de *hardware* complejo, así como de capacidad de almacenamiento, sensores, motores y demás componentes electrónicos que le permiten al robot desarrollar una tarea, además del *software* para controlar dichas actividades y que el robot las realice con eficiencia.

Decir "la nube" es referirse de manera metafórica al Internet o a los servicios que brinda la red de redes, como puede ser el correo electrónico, almacenamiento de archivos o información como los servicios de *Dropbox*, *One Drive*, *Google Drive* o *Box*, entre otros, aplicaciones como las redes sociales o incluso servicios para jugar en tiempo real, donde mediante una conexión de banda ancha se puede disfrutar al máximo de un video juego, sin necesidad de contar con un *hardware* poderoso por parte del usuario, todo esto de manera gratuita y permitiendo ahorros importantes.

Robótica en la nube tiene como objetivo utilizar esta infraestructura disponible en Internet, pero orientada a los robots, es decir, crear sitios web donde los robots puedan hacer uso de *hardware* remoto, almacenar, consultar y descargar información para realizar sus tareas y poder "aprender" nuevas funciones.

Robótica en la nube tiene dos niveles de arquitectura como se menciona en (Hu, Tay, & Wen, 2012):

2.4.1. R2R (Robot-to-Robot)

Es una red local para un grupo de robots (Figura 2.3). Esta arquitectura, también conocida como *Networked Robotics*, tiene como su principal función el proporcionar un medio para compartir información entre robots dentro de una red de comunicación privada.

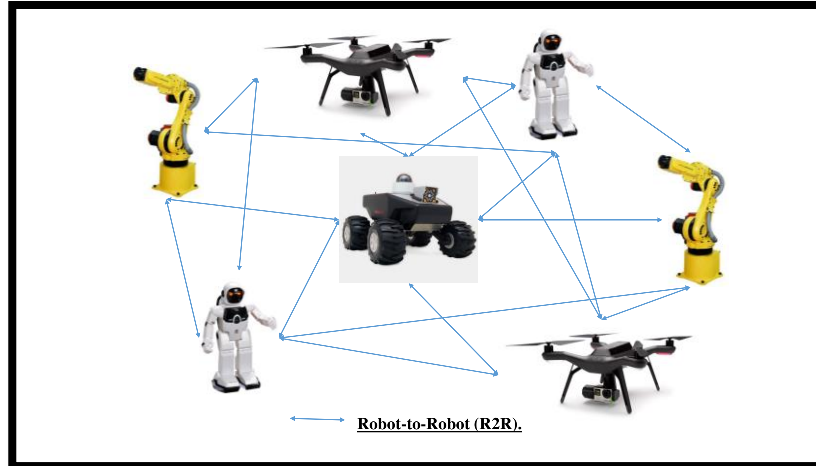


Figura 2.3: Diagrama de red robot-to-robot

2.4.2. R2C (Robot-to-Cloud)

En este escenario los robots comparten la información mediante un sistema alojado en la nube. A diferencia de *Networked Robotics*, se utiliza la red de Internet como plataforma de comunicación entre robots (Figura 2.4). Esto eleva las capacidades de cada robot debido al beneficio de la conexión a Internet. Este concepto se subdivide en las siguientes clases de estructura:

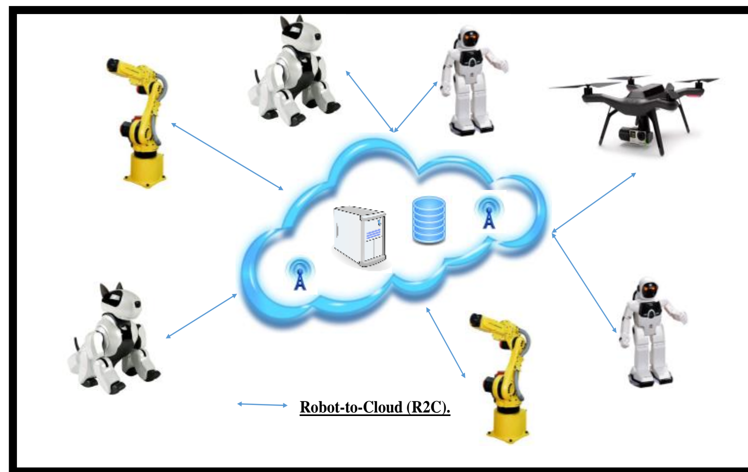


Figura 2.4: Diagrama de red robot-to-cloud

- **SaaS (Software as a Service).** Son aplicaciones o programas que funcionan en Internet, esto elimina la necesidad de instalar y ejecutar dicho *software* en el sistema del usuario (Koken, 2015). Es decir, se tiene acceso a un programa o aplicación determinada

mediante el uso de navegadores de Internet o una conexión remota, sin la necesidad de hacer una instalación en el dispositivo del usuario final.

- **PaaS (Platform as a Service).** Se refiere a una plataforma de cómputo con infraestructura en la nube. Ofrece a los desarrolladores tener un sistema en la nube que proporciona un ambiente seguro y recursos disponibles, durante el ciclo de vida del sistema (Koken, 2015). El usuario es quien administra su información y aplicaciones que están alojadas en la nube.

- **IaaS (Infrastructure as a Service).** Provee la infraestructura como un servicio. El cliente no necesita comprar servidores, unidades de almacenamiento o recursos en la red, puesto que los tiene disponibles en la nube. El cliente puede hacer uso del *hardware* y *software* para incorporarlos a su sistema, es decir, el servicio proporcionado incluye el *hardware* y *software* de base.

- **RaaS (Robot as a Service).** Se refiere a los robots que pueden ser dinámicamente coordinados para la ejecución de tareas específicas. RaaS tiene tres aspectos de sistema: estructura, interfaz y comportamiento. Puede haber varios tipos de unidades de nube, robots o dispositivos inteligentes, por ejemplo: robots de vigilancia, de servidumbre, de compañía y atención a pacientes. Estos robots se encuentran distribuidos en diferentes lugares y pueden ingresar a la plataforma de robótica en la nube (Koken, 2015).

2.5. Robot Operating System (ROS)

Aunque las siglas de ROS significan “Sistema Operativo Robot”, en realidad se trata de un Meta-Sistema Operativo, el cual ofrece las características de un SO, pero debe ejecutarse sobre otro SO. ROS se ejecuta sobre plataformas Unix de manera totalmente funcional, está probado en sistemas operativos Ubuntu y Mac OS X, también puede ser instalado sobre Microsoft Windows, aunque aún tiene errores.

2.5.1. Arquitectura

Existen tres niveles de conceptos en ROS: nivel del sistema de archivos, nivel de computación gráfica y el nivel de comunidad, se toma de (Willow Garage, s.f.) (García Cazorla , 2013) y (Labrador Fleitas, 2014) los siguientes elementos.

El nivel de Sistema de archivos se refiere a los recursos del programa:

- **Paquetes**, (Figura 2.6 inciso c), son la unidad principal de organización en ROS y pueden contener procesos ejecutables (nodos en ROS), una biblioteca dependiente, conjunto de datos, archivos de configuración o cualquier otra cosa que sea útil para una organización conjunta.

- **Manifiestos**, los cuales proporcionan datos sobre un paquete, incluyendo su información de licencia y dependencias, así como información del compilador.

- **Pilas**, (Figura 2.5), son un conjunto de paquetes que comparten una misma funcionalidad, son equivalentes a las librerías en otros lenguajes de programación.

- **Manifiestos de pilas**, los cuales proporcionan datos sobre una pila, incluyendo su información de licencia y sus dependencias en otras pilas.

- **Mensajes**, (Figura 2.6 inciso d), definen las estructuras de datos para los mensajes enviados en ROS, dado que los nodos se comunican mediante mensajes.

- **Servicios**, (Figura 2.6 inciso a), es el tipo de arquitectura encargada de realizar la comunicación entre nodos, hace uso de dos tipos de mensaje, el primero es para la solicitud y el segundo para la respuesta de los servicios requeridos por ROS.

El nivel de computación a nivel gráfico es la red ROS que se encarga de procesar todos los datos:

- **Nodos**, (Figura 2.6 inciso b y Figura 2.7), a son programas, que realizan funciones como puede ser publicar mensajes o hacer cualquier procesamiento. Son procesos que llevan a cabo cálculos. Por ejemplo, un nodo controla un sensor o un nodo calcula la velocidad de un motor.

- **Maestro**, proporciona registro de nombres y la búsqueda para el resto de los nodos. Sin el Maestro los nodos no serían capaces de encontrar mensajes entre sí, intercambiar o invocar los servicios, por lo cual son indispensables para la ejecución de cualquier programa.

• **Temas**, (Figura 2.6 inciso d), también llamados tópicos son los nombres que identifican el contenido de un mensaje y pueden ser publicador o suscriptor. Se puede pensar en un tema como un Bus de mensajes, donde cada Bus tiene un nombre y cualquier nodo puede conectarse al Bus para enviar o recibir Mensajes.

En el nivel de comunidad:

• **Distribución**, son las distintas versiones de ROS que se pueden instalar. Es similar a las versiones de Linux, por ejemplo, Ubuntu. La última versión es *ROS Melodic Morenia*.

• **Repositorios**, (Figura 2.5), son paquetes de *software* para desarrollo de sistemas robóticos, de los cuales algunos son administrados y alojados por *Willow Garage* y muchos otros son creados, compartidos y alojados en servidores por organizaciones, instituciones o aficionados y son parte del proyecto ROS.

• **Wiki de ROS**, se trata de un foro donde los usuarios pueden crear contenido y compartir información sobre ROS, cualquier persona puede contribuir al foro mediante un registro en el sitio web.

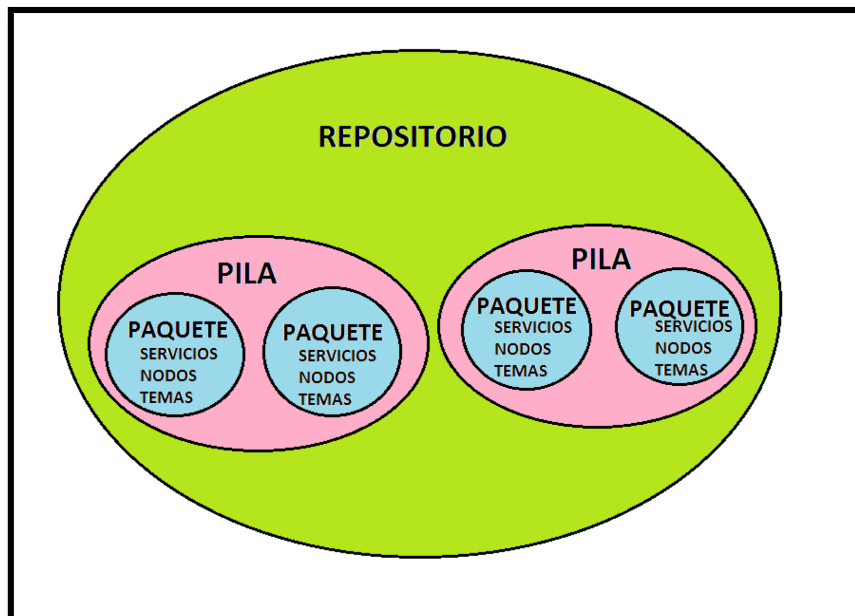


Figura 2.5: Estructura de un repositorio en ROS

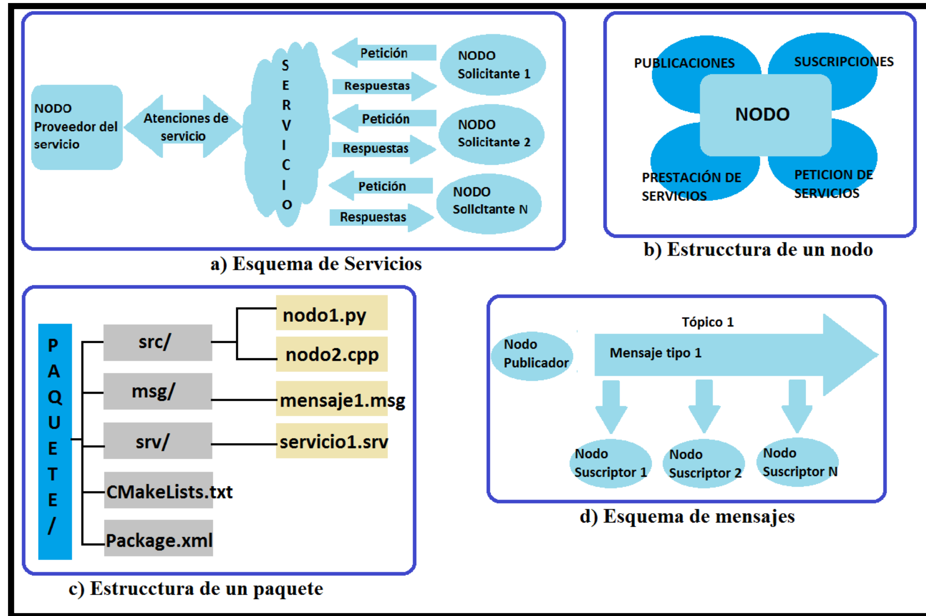


Figura 2.6: Estructuras y esquemas de conceptos. Se describen gráficamente algunos conceptos de ROS. a) Representa un servicio. b) Ilustra la estructura de un Nodo. c) Se muestran las partes que componen un paquete. d) Esquema de mensajes y temas

Después de la descripción de los tres niveles de conceptos en ROS, es preciso comprender la estructura del mismo, la cual se puede observar en la Figura 2.5.

La comunicación en ROS utiliza dos tipos de mensajes, el primero para la solicitud y el segundo para la respuesta a la solicitud del otro nodo. Existen nodos del tipo servidor y del tipo cliente, tanto el nodo servidor como el nodo cliente, pueden enviar solicitudes y respuestas, la Figura 2.7 muestra el mecanismo de comunicación entre nodos.

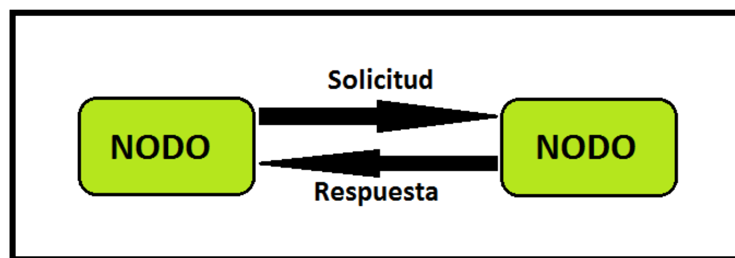


Figura 2.7: Estructura de comunicación entre nodos

Existe un tipo de archivo llamado ***BAG***, (nombrado así por su extensión ".bag"), el cual permite almacenar información de uno o varios temas para después poder consultar, procesar, analizar o visualizar los datos capturados.

La cantidad de usuarios de ROS está creciendo rápidamente, esto se debe a las ventajas que ofrece este SO tales como: ser un *software* libre, la gran compatibilidad con *hardware* (actuadores, sensores, cámaras, Arduino, Raspberry, entre otros), la reutilización del código, simuladores 3D, soporte multilenguaje (C, C++, Phyton o JAVA en modo experimental), por lo cual es aceptado y utilizado por investigadores, centros educativos y la industria.

Para el caso de este proyecto la plataforma que se utilizó fue ROS debido a las características descritas anteriormente y la aceptación que tiene alrededor del mundo, lo que permite contar con mucha información y soporte con respecto al desarrollo de *software* y funcionamiento del SO.

Capítulo 3: Ensamble, configuración y operación de los robots

3.1. Robot móvil de armado propio

Para llevar a cabo los experimentos en este trabajo se requerían dos robots para su interacción. Se construyó un robot móvil con la premisa de ser de bajo costo. Con base en un chasis de acrílico, dos motorreductores, un Arduino Uno y una Raspberri Pi 3 se comenzó el ensamble del robot propio. Sin embargo, el material antes mencionado no era suficiente para el armado del robot por lo que se fueron incorporando piezas conforme se avanzaba en el proyecto. En la parte de *software*, se tuvo el objetivo de utilizar *Open Source*, por este motivo se instaló Ubuntu, el IDE de Arduino y ROS para comenzar el proyecto.

3.1.1. Etapas de desarrollo del robot de armado propio

El desarrollo del robot móvil se dividió en etapas de construcción (Figura 3.1), durante las cuales se fue mejorando el *hardware* y *software* para lograr una correcta locomoción. Dichas etapas se describen a continuación.

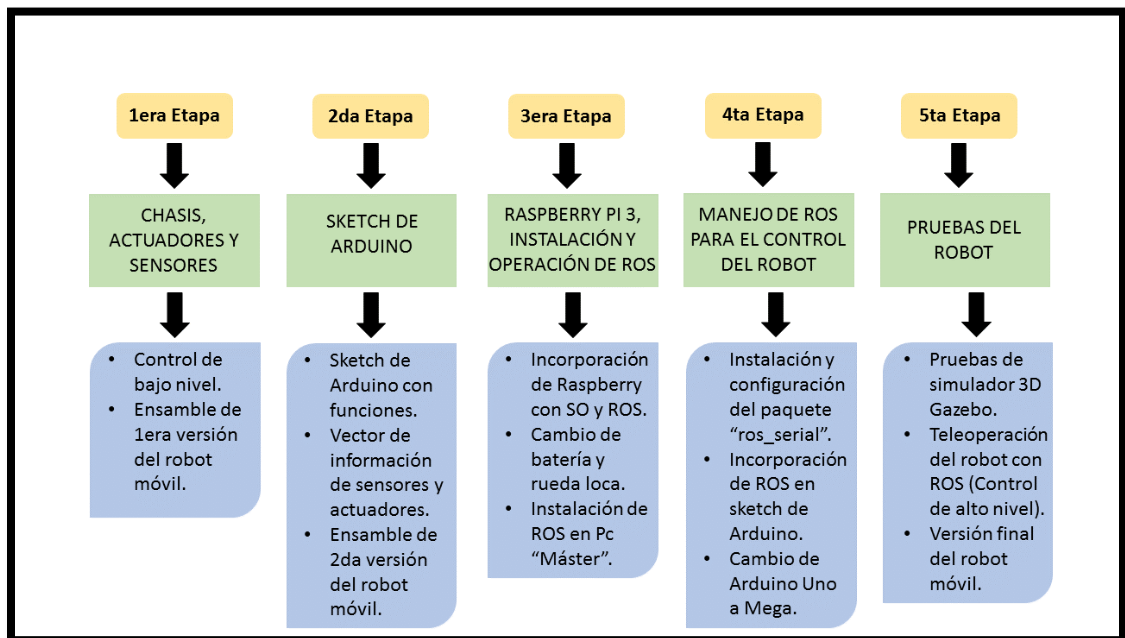


Figura 3.1: Diagrama de etapas de desarrollo del robot móvil

3.1.2. Primera etapa: Ensamble de chasis, actuadores y sensores.

En la primera etapa de desarrollo consistió en el ensamblado de los actuadores y sensores en el chasis del robot. Se ensambló lo que fue la versión 1 del robot (Figura 3.2). Consistiendo en los siguientes elementos:

- Chasis de acrílico para robot de 3 ruedas.
- Dos motorreductores de corriente continua.
- Fuente de alimentación.
- Rueda libre.
- Sensor de distancia.
- Arduino Uno.

Para alimentar de energía eléctrica al Arduino se ocupó la batería recargable de la marca Klip Xtreme. En el caso de los motores, se ocupó la batería recargable 12V 1.4 Ah Ácido-Plomo.

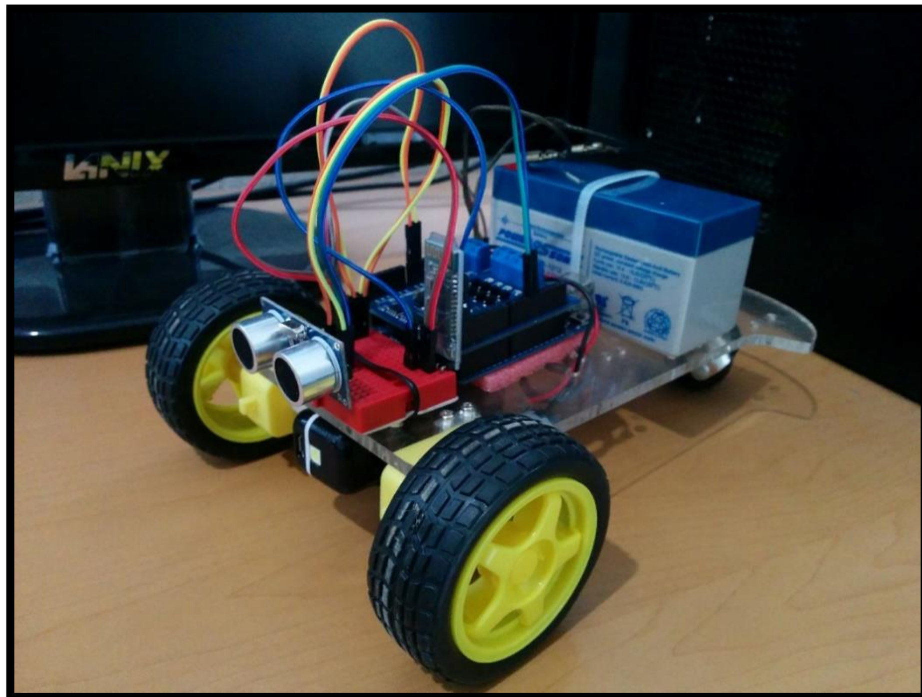


Figura 3.2: Versión 1 del robot de armado propio

3.1.3. Segunda etapa: Programación en Arduino del control básico

En esta segunda versión (Figura 3.3) se trabajó en el IDE de Arduino para lograr la locomoción del robot. El programa permite controlar la velocidad y la dirección de los motores conectados a las llantas, a través de un puente H con el circuito integrado L293D. Además, se programó la lectura de un sensor ultrasónico (HC-SR04) para medir la distancia con objetos al frente del robot.

En cuanto al suministro de energía. Durante las pruebas de desplazamiento se notó la dificultad que tenía el robot para moverse debido al peso que tiene la batería Ácido-Plomo.

En este *sketch* para Arduino se programaron en funciones que concentran las actividades activación y manejo de los motores (Anexo B *Sketch* de Arduino) para tener la ventaja de un código más claro y reducir el tamaño del mismo. Se generaron las funciones de Avance, Retroceso, Paro, Enclavamiento de la botonera, Enclavamiento del puerto serial, Velocidad y Sensor Ultrasónico.

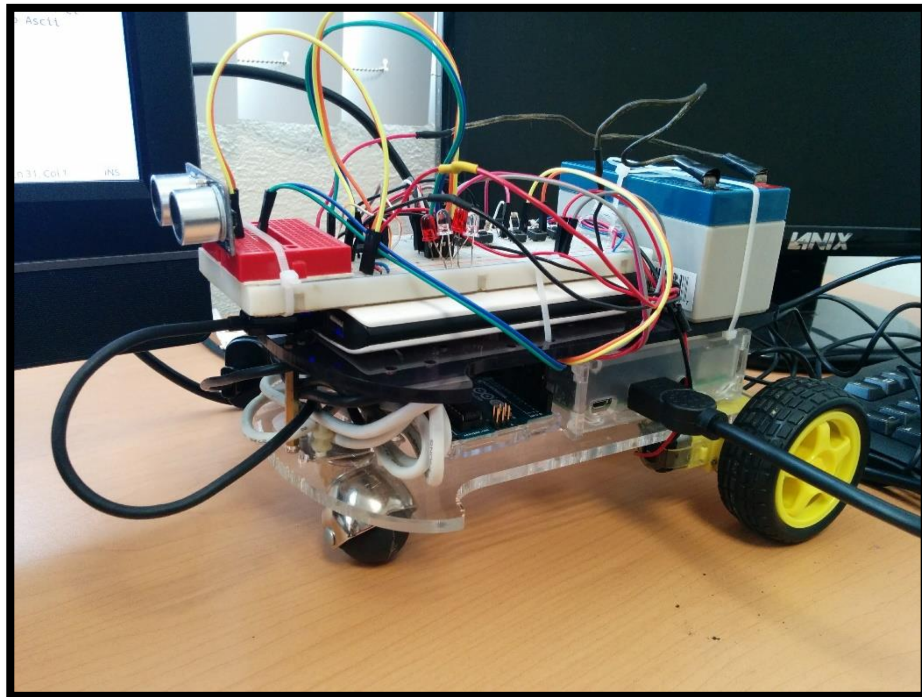


Figura 3.3: Versión 2 del robot de armado propio

Para el caso de las funciones Avance, Retroceso y Paro, como su nombre lo indica, se establecen valores que brindan al robot el estado de los motores, es decir, indican si las ruedas

deben o no girar y en qué sentido. Por otro lado, la función de velocidad lee y establece la velocidad a la que deberán girar las llantas del robot, la función del Sensor Ultrasónico obtiene datos desde el sensor para calcular distancias con objetos que se encentren al frente del robot y evitar colisiones.

Las funciones de Enclavamiento de Botonera y Puerto Serial, proporcionan la capacidad de obtener un pulso y liberar el canal de comunicación para poder recibir otro pulso con una instrucción diferente, en otras palabras, si el robot recibe la instrucción de Avance desde la botonera y el botón permanece oprimido, esto no impide que al presionar el botón Paro el robot responda a la instrucción y se detenga; lo mismo sucede con el Puerto Serial.

Por otra parte, se incluyó un vector de información con la finalidad de poder visualizar el estado de los sensores y actuadores en el robot. El vector muestra en pantalla la velocidad de los motores, el estado de los motores, el cual puede ser avance, retroceso o paro y además la distancia que se obtiene desde el Sensor Ultrasónico (Figura 3.4).

Al mismo tiempo se agregó el segundo nivel de acrílico con el objetivo de colocar la Raspberry con su carcasa y los protoboard en el robot. En el protoboard de 830 puntos se elaboró una botonera (Anexo C Diagramas de conexión) con la finalidad de tener una manera de manipular al robot en caso de algún inconveniente y que no respondiera a las señales enviadas desde la computadora, en el mismo protoboard se colocaron dos micro switch para controlar la energía entregada a los sensores y actuadores, así como al Arduino y la Raspberry. El protoboard de 170 puntos se utilizó para conectar el Sensor Ultrasónico. Además, se cambió la batería de alimentación para el Arduino con la finalidad de aumentar el tiempo de operación del robot, la batería utilizada fue la CDP, modelo R-PB10k de 5V 10000 mAh.

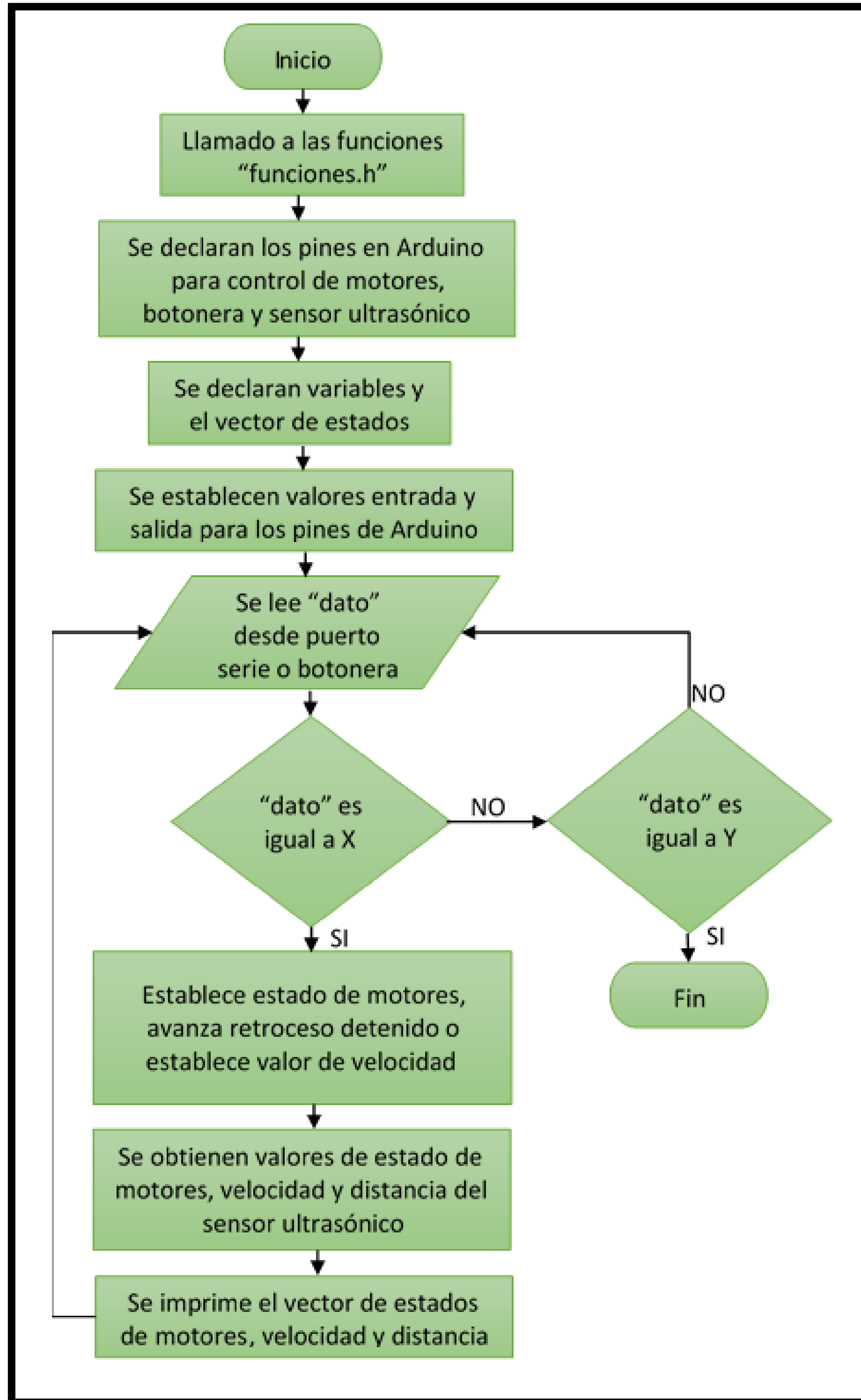


Figura 3.4: Diagrama de flujo del primer *sketch* de Arduino

3.1.4. Tercera etapa: Incorporación de Raspberry PI 3 e instalación y operación básica de ROS

La tercera etapa de construcción fue la configuración de la Raspberry PI 3, a la cual se le instaló el SO Ubuntu Mate (16.04.2 Xenial) y ROS en su versión Kinetic Kame (Figura 3.5). Para la instalación del SO se descargó el archivo .iso desde la página oficial (Ubuntu MATE Team, 2018), y se trabajó desde un ambiente Windows, siguiendo los pasos de (ETA PRIME, 2016).

De manera resumida, los pasos para la instalación de Ubuntu Mate en la Raspberry PI 3 fueron:

- Descargar el SO desde la página oficial.
- Tener una tarjeta micro SD de por lo menos 8GB de capacidad y de clase 10.
- Descargar el programa "Win32 Disk Imager".
- Conectar a la Pc la memoria micro SD y dar formato (Fat 32).
- Descomprimir el archivo .iso.
- Ejecutar "Win32 Disk Manager" y seleccionar la ruta de la tarjeta micro SD, después seleccionar el archivo .iso que contiene el SO Ubuntu Mate y por último dar clic en "write".
- Finalmente extraer la memoria micro SD de la Pc e insertarla en la ranura de la Raspberry para el primer inicio.

Una vez teniendo el SO funcionando, se procedió con la instalación de ROS siguiendo los pasos detallados en la "wiki de ROS" (Willow Garage, 2018):

Al finalizar con la instalación, se comprobó la funcionalidad de ROS ejecutando desde una terminal el comando:

```
$ roscore
```

Por otro lado, ya con la Raspberry y todos los componentes de *hardware* montados en el robot, se notó la dificultad que tenía para moverse, de manera que se sustituyeron algunas partes para hacer más ligero el robot y reducir su peso. Se optó por remplazar las baterías de Acido-Plomo y la batería CDP por una sola batería tipo LiPo junto a un regulador de voltaje

de corriente continua de 12-5V 5A. La batería LiPo entrega 11.V, los cuales son suficientes para mover los motores, sin embargo, el Arduino Uno y la Raspberry PI 3 necesitan 5V para funcionar y es por este motivo que se hizo uso del regulador.

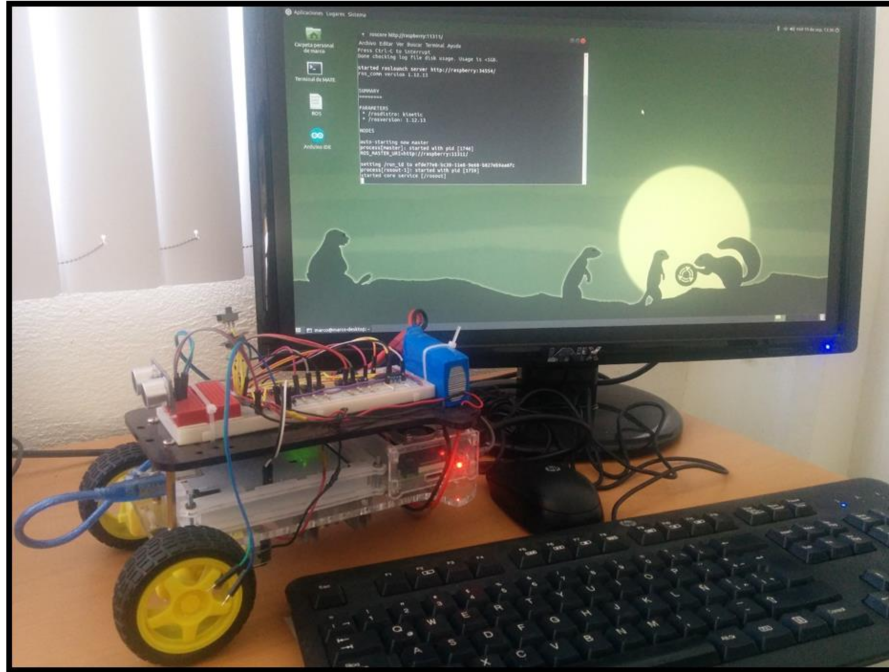


Figura 3.5: Robot de armado propio ejecutando ROS sobre Ubuntu Mate

Asimismo, se cambió la rueda loca tipo "carrito de súper mercado" debido a la resistencia que presentaba al hacer un giro, se suplió por una rueda tipo "bola" de metal para tener una mejor respuesta al movimiento.

Al mismo tiempo, se trabajó con la Pc que sirvió como "master" en ROS, para este caso la instalación de ROS se hizo en un SO Ubuntu 14.04 LTS (Trusty) 64 bits.

La distribución instalada de ROS fue *Indigo Igloo*. Se siguieron los pasos que la página de ROS indica en (Willow Garage, 2017).

Para comprobar el correcto funcionamiento de ROS en la Pc y con el objetivo de conocer mejor el manejo y la interfaz de ROS, se instaló un emulador 3D llamado "Gazebo" (Figura 3.6), mediante el cual es posible la virtualización entornos o robots e interactuar con los sensores que el robot contenga. Para su instalación se siguieron los pasos de (Willow Garage, 2016).

Una vez instalado Gazebo se ejecuta en una terminal el comando para correr ROS:

```
$ roscore
```

Después se ejecuta en una nueva terminal el comando para lanzar el simulador:

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

En una nueva terminal se ejecuta el comando para controlar el mundo virtual y la manipulación del Turtlebot mediante teclado y mouse:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

Logrando así experimentar con el uso y la interfaz de ROS para tareas básicas.

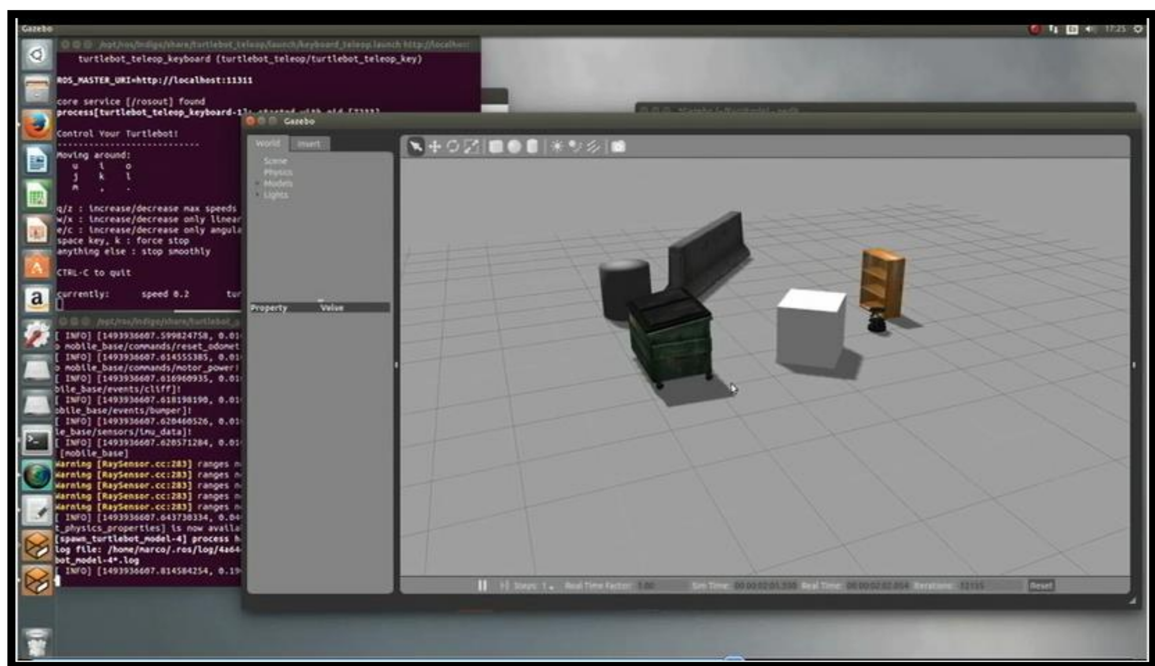


Figura 3.6: Emulador 3D Gazebo

3.1.5. Cuarta etapa: Ejecución de ROS para el control del robot

Durante esta etapa se instaló el paquete “ROSSERIAL” en la Pc, el cual es un protocolo para la comunicación de ROS con Arduino mediante el Puerto Serial. Los pasos para la descarga del paquete “ROSSERIAL_ARDUINO” fueron:

- Ejecutar la descarga del paquete con los comandos:

```
$ sudo apt-get install ros-indigo-rosserial-arduino

$ sudo apt-get install ros-indigo-rosserial
```

- Posteriormente se copia la librería de ROS “ros_lib” al ambiente de desarrollo de Arduino para lograr la compatibilidad de los programas. Se ejecutan los comandos:

```
$ cd <sketchbook>/libraries

$ rm -rf ros_lib

$ rosrun rosserial_arduino make_libraries.py
```

Para comprobar la correcta instalación y configuración, se ejecuta el ambiente de desarrollo de Arduino, se da clic en “Archivo”, “Ejemplos” y se debe observar la opción de “ros_lib”.

Una vez instalado el paquete rosserial se modificó el *sketch* para incluir las librerías de ROS y generar los nodos a utilizar (Figura 3.7). En esta etapa, Arduino Uno comenzó a generar errores debido al tamaño del programa y su capacidad de memoria (32 kb), por lo que se sustituyó con un Arduino Mega, el cual posee más memoria (256 kb).

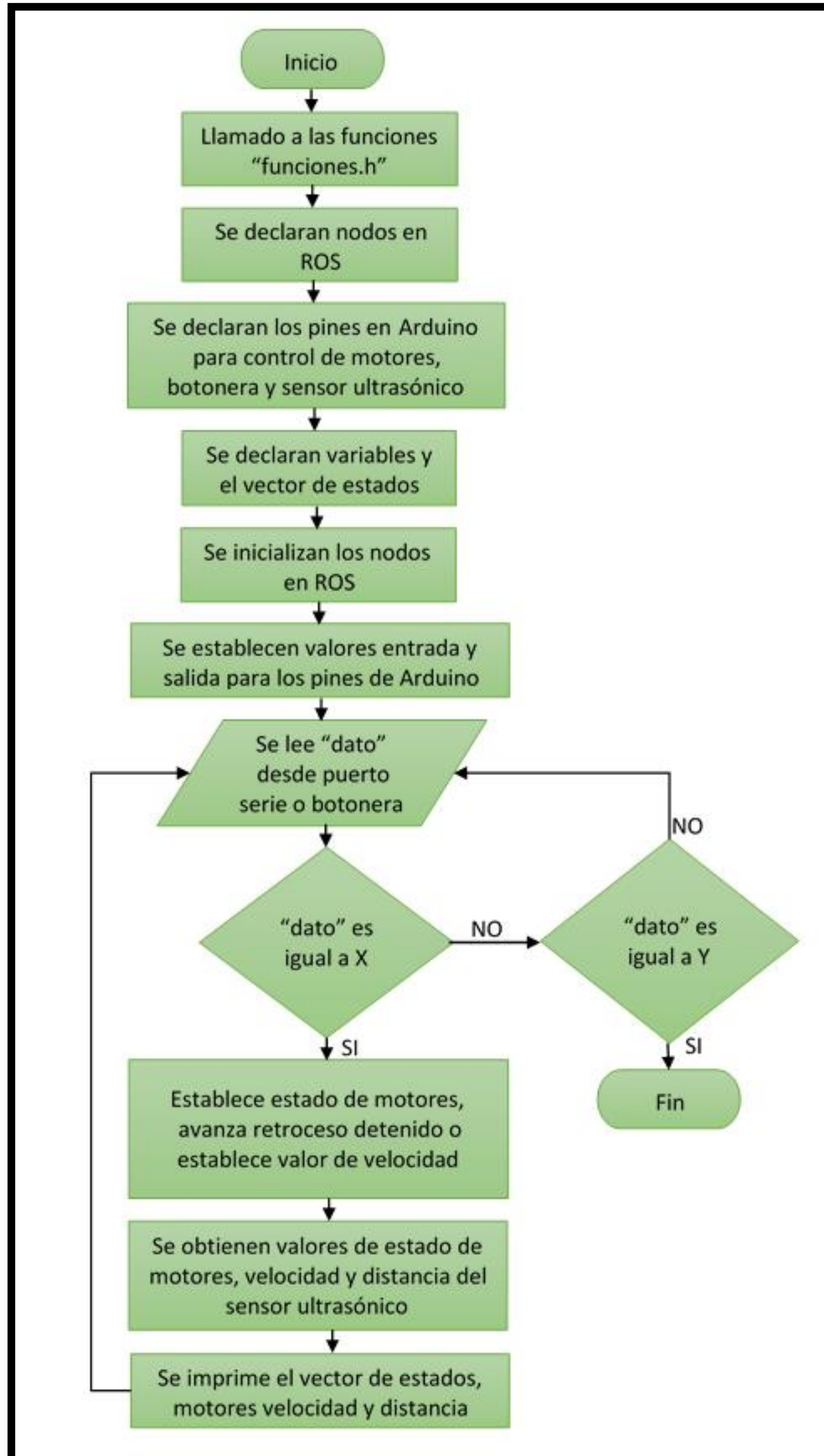


Figura 3.7: Diagrama de flujo del primer *sketch* de Arduino para ROS

3.1.6. Quinta etapa: Pruebas de funcionalidad del robot

Ya con el robot en su versión final (Figura 3.8), se realizaron pruebas de comunicación entre el robot y una Pc, logrando la tele operación del robot y recibiendo la información contenida en el vector de datos en un nodo de ROS (estado de motores, velocidad y distancia del sensor ultrasónico). El robot ya no presentó problemas de movimiento, ni de errores durante la manipulación.

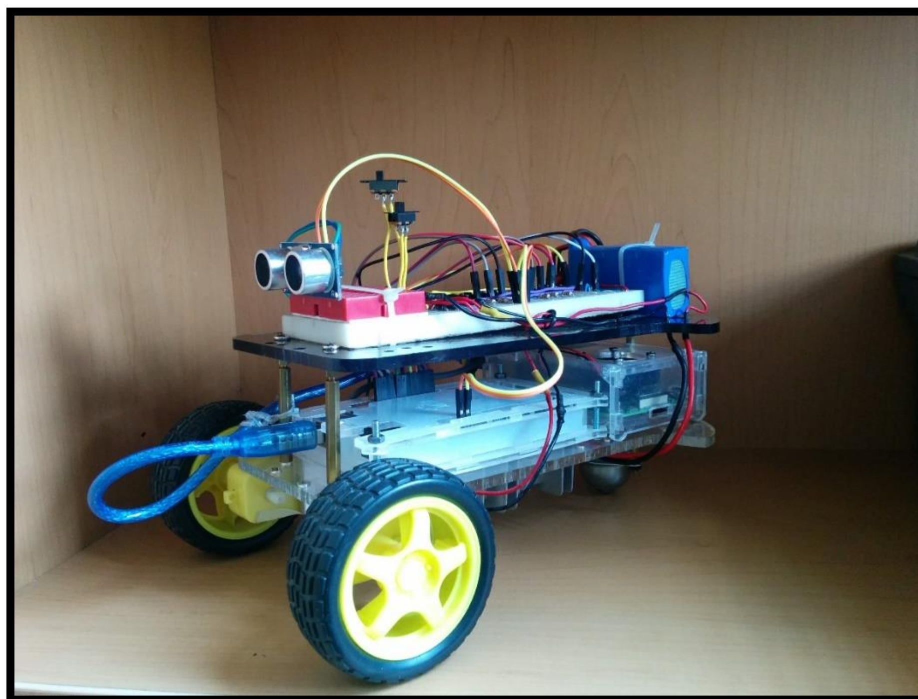


Figura 3.8: Versión final del robot de armado propio

3.2. Ensamble y operación del Turtlebot 3 Burger

Turtlebot es una familia de robots móviles de nueva generación, personalizables, modulares y compactos. Desarrollados por la compañía *Yujin Robots*, están disponibles los modelos *Burger*, *Waffle* y *Waffle Pi* (Robotis, 2018), todos son controlados por el sistema operativo ROS. Para este caso, el modelo utilizado es el *Burger* (Figura 3.9), cuenta con una computadora abordo, la *Raspberry pi3* y una tarjeta *OpenCR* que permite controlar sensores y actuadores entre los que destaca el sensor Lidar (*LDS Laser Distance Sensor*), es un escáner laser 2D capaz de recolectar datos en 360° con un rango que va de los 120 mm a 3,500 mm,

lo que le permite realizar tareas de mapeo (*SLAM Simultaneous Localization and Mapping*), el robot y sus principales características se pueden observar en la siguiente ilustración.

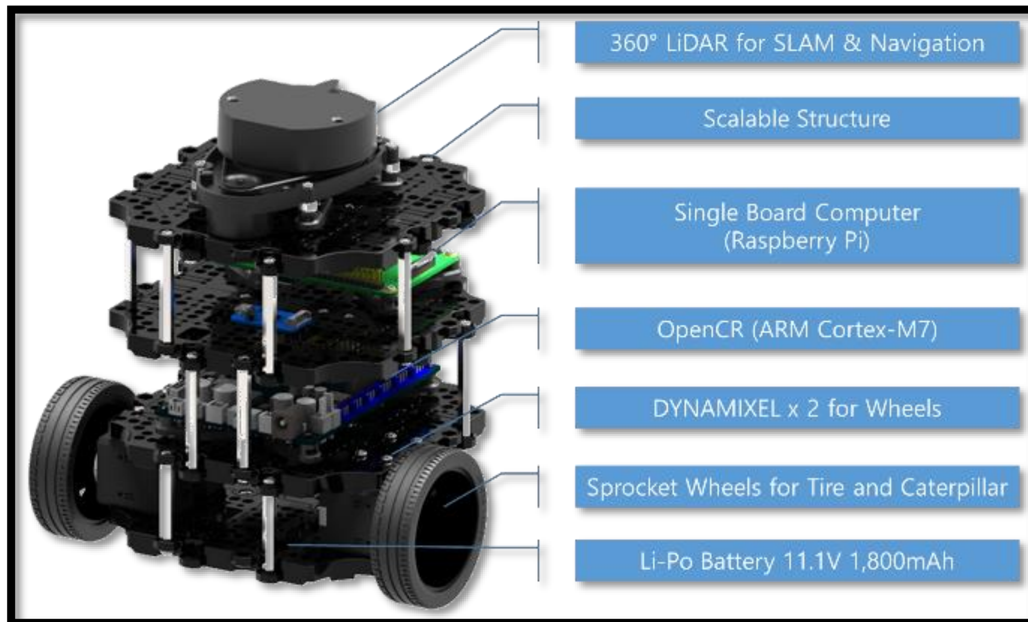


Figura 3.9: Turtlebot 3 Burger. Tomada de (Robotis, 2018)

3.2.1. Ensamble del robot comercial

Para el ensamble del robot se siguió el manual del fabricante, mismo que viene incluido junto con todas las piezas del robot y herramienta necesaria. El manual contiene instrucciones en japonés, chino, coreano e inglés con la descripción de todas las piezas incluidas, así como la guía paso a paso para el correcto ensamble del *hardware* del robot (Figura 3.10).

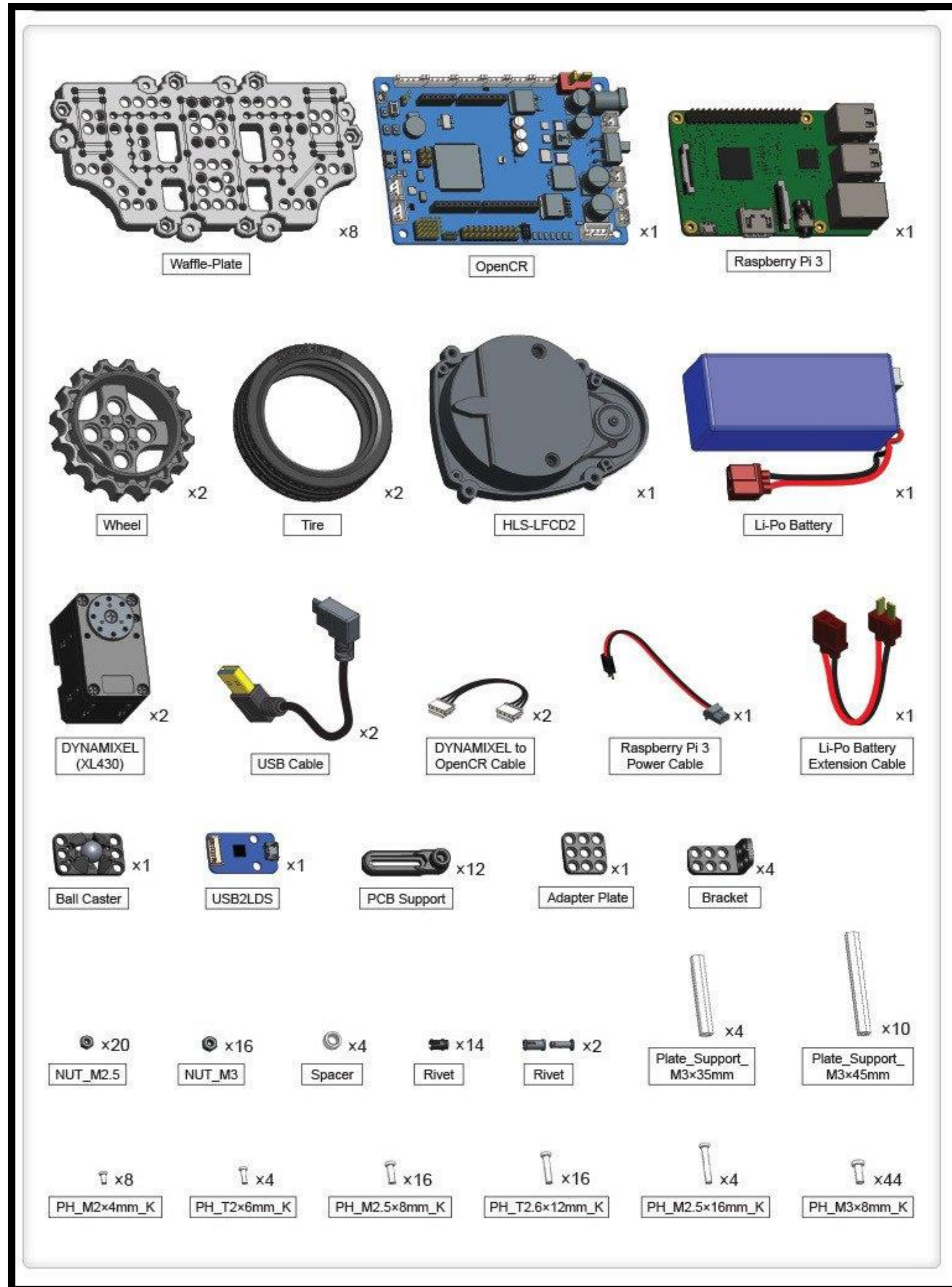


Figura 3.10: Lista de Partes del Turtlebot 3 Burger. Tomada de (Robotis, 2018)

3.2.2. Instalación y configuración del *software*

Después de ensamblar el *hardware*, se continuó con la instalación del *software* necesario. Se cargó el SO Ubuntu Mate (16.04 Xenial) en la Raspberry PI 3 además de ROS Kinetic Kame de manera similar que se hizo en el robot de armado propio.

3.2.3. Pruebas del Turtlebot 3 Burger

Una vez que el Turtlebot 3 se encontraba ensamblado, con SO y con ROS, se comprobó el correcto funcionamiento de ROS. En la Pc se ejecutó desde una nueva terminal el comando:

```
$ roscore
```

Ya con ROS ejecutándose en la Pc, se debe ejecutar en una nueva terminal del Turtlebot 3:

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Con estos pasos queda listo el robot para conectarse con la computadora remota, donde se debe ejecutar en una nueva terminal los comandos:

```
$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_bringup turtlebot3_remote.launch

$ rosrun rviz rviz -d `rospack find turtlebot3_description`/rviz/model.rviz
```

Después de introducir los comandos, el TurtleBot 3 se encuentra listo para ser manipulado, existen varias alternativas para controlar los movimientos del robot, puede ser operado mediante el teclado de la computadora remota, control de *XBOX 360*, *Play Station 3*, *Robotis RC100*, control remoto del *Wii* o incluso mediante una aplicación Android. Dependiendo el mando seleccionado se deben instalar los paquetes necesarios para la tele-operación, en este caso el control del *Play Station* fue el seleccionado debido a su disponibilidad en el laboratorio. En una nueva terminal se ejecuta el comando:

```
$ roslaunch teleop_twist_joy teleop.launch
```

Una vez que se han ejecutado correctamente los comandos anteriores, se debe ejecutar la instrucción para comenzar a generar el mapa (Figura 3.11). Es importante en este punto, asegurarse que los relojes, tanto del robot como el de la computadora remota, se encuentren en sincronía, ya que de no ser así el comando mostrará error y no se podrá generar la exploración.

```
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

El comando para guardar el mapa es:

```
$ rosrun map_server map_saver -f ~/map
```

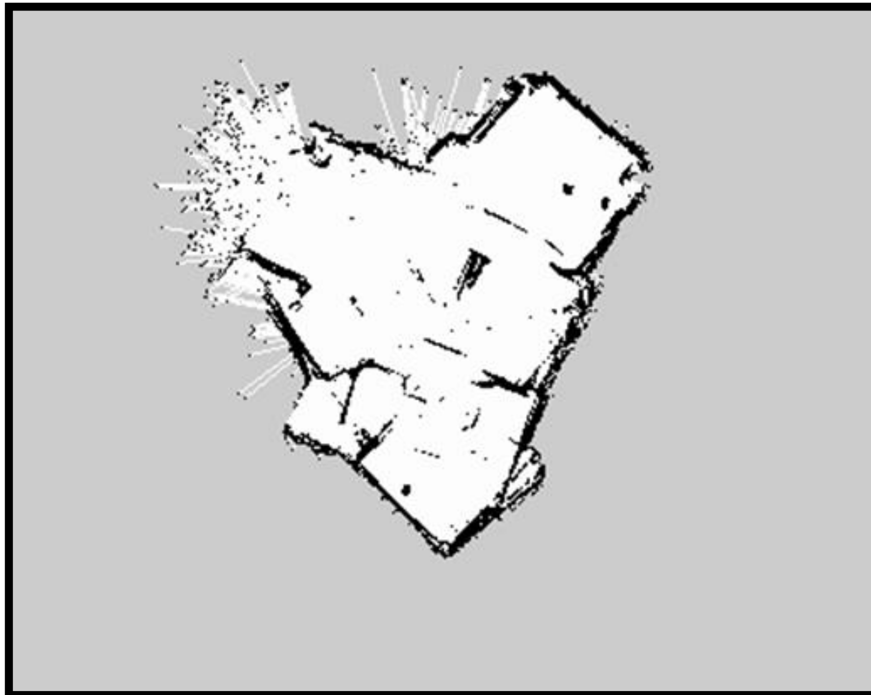


Figura 3.11: Mapa generado con Turtlebot 3 en las oficinas del Edificio F del CU UAEM VM

En la Figura 3.12 se muestra la estructura de nodos y tópicos en ROS que permiten la operación descrita. Aparecen los nodos de tele operación (*/joy_node* y */teleop_twist_joy*) y tópicos (*/joy* y */cmd_vel*). El nodo del Turtlebot 3 (*/turtlebot_core*) y su sensor Lidar (*/turtlebot3_lds*), del mapa que se está construyendo (*/turtlebot3_slam_gmapping*) y los tópicos que transmiten la información.

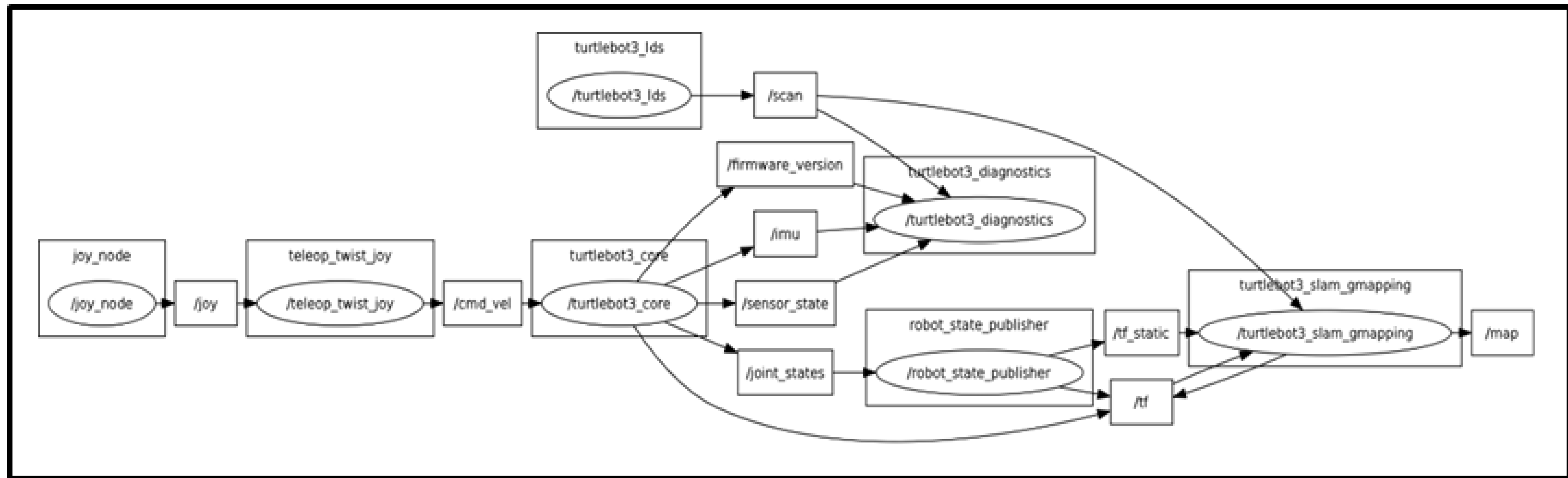


Figura 3.12: Diagrama de nodos y tópicos para generar un mapa con Turtlebot 3

Capítulo 4: Diseño y desarrollo experimental

4.1.Descripción de la plataforma

Mediante el uso de dos robots, uno comercial y otro de armado propio; se busca compartir información entre ambos a través de una computadora que hará la función de máster y brindará conexión a la nube. Dicho de otra manera, los robots podrán subir, consultar y descargar información mediante una red inalámbrica a través de una computadora que tendrá la función de servidor y proporcionará servicio de nube (Figura 4.1).

En primer lugar, se tiene la computadora que contiene el *software* necesario para interactuar con los robots (SO Ubuntu, librerías y paquetes de ROS, Arduino, entre otros). Mediante dicha computadora se ejecuta ROS, para posteriormente iniciar con la tele operación del robot comercial y a través de sus sensores, obtener datos acerca de su entorno.

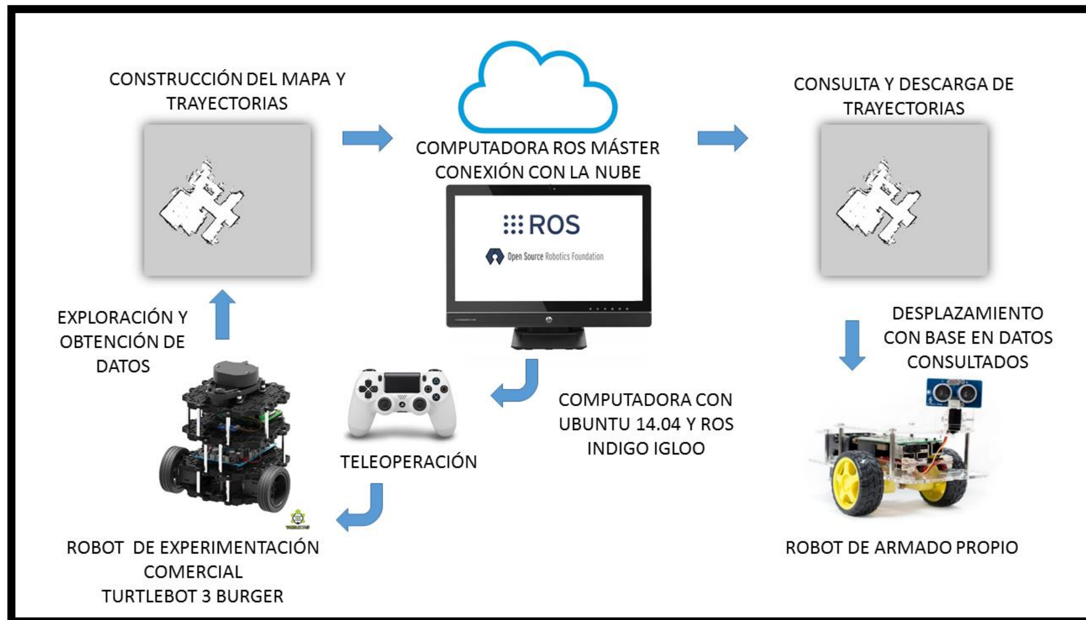


Figura 4.1: Diagrama de comunicación entre los robots, el máster y la nube

4.2. Creación e interpretación de datos del modelo de espacio de trabajo

En esta sección se describe la forma para interpretar los datos del sensor Lidar y los datos cinemáticos del Turtlebot 3, de tal manera que se construye el modelo del espacio del entorno del robot. En la primera parte se configurará las llamadas *bags* para recuperar los datos de los tópicos correspondientes al sensor Lidar y de velocidad, posteriormente se programa a través de Matlab un código para graficar la información guardada. Se presentan experimentos donde se representa el movimiento del robot y las lecturas del mismo a cada "paso".

4.2.1. Creación de la bolsa de datos del Lidar y datos cinemáticos

Para generar los experimentos de representación del entorno del robot, se generaron archivos *bags*, estos se generan al mismo tiempo de ejecutar una exploración del entorno del Turtlebot 3, donde se almacena información de los temas de principal interés (en este caso el tema del sensor Lidar */scan*, velocidad */cmd_vel* e IMU */imu*), para posteriormente procesarlos, consultarlos y generar el mapa en la computadora.

En la Pc se ejecuta:

```
$ roscore
```

En el Turtlebot 3 se ejecuta:

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Nuevamente en la Pc se ejecuta en una nueva terminal:

```
$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_bringup turtlebot3_remote.launch

$ rosrun rviz rviz -d `rospack find turtlebot3_description`/rviz/model.rviz
```

Para tele operar el robot se ejecuta en una nueva terminal de la Pc:

```
$ roslaunch teleop_twist_joy teleop.launch
```

Hasta este paso el robot comercial está listo para comenzar a explorar su entorno, pero antes se debe ejecutar el comando para generar el archivo *bag*, es importante tener un

directorio para alojar los datos generados. Para este caso se creó el directorio: *bagfiles* y se encuentra dentro de “Carpeta personal”.

Ahora se debe ejecutar en una nueva terminal en la Pc el comando para cambiar de directorio:

```
$ cd bagfiles
```

Ya dentro del directorio en la Pc, se introduce en una nueva terminal el comando para crear el archivo *bag*:

```
$ rosbag record -O NombreDeMiArchivoBag /tema1 /tema2 /temaN
```

Como se puede ver en el comando anterior, solo es necesario introducir el nombre del nuevo archivo y el tema o los temas a grabar dentro del *bag*. Para terminar con la captura de datos se presiona “ctrl+c”.

Una vez creado el archivo *bag* se procede con la extracción de los datos, para este caso se importaron los datos de los temas hacia un archivo *.txt*, en una nueva terminal en la Pc, dentro del directorio que contiene al archivo *bag* se introduce el comando:

```
$ rostopic echo -b NombreDelArchivo.bag -p /Tema > NombreDelTxt.txt
```

Este comando va a generar un archivo que contiene toda la información del tema seleccionado, el archivo será *.txt* y aparecerá dentro del directorio donde se encuentra el archivo *bag*.

4.2.2. Interpretación de la información de movimiento de translación lineal

Para lograr procesar los datos generados por el Turtlebot 3 en los archivos *bag*, se hizo uso del programa Matlab, debido a las herramientas que posee para trabajar con matrices de información.

Después de haber generado el archivo *txt*, se copian los datos y se pegan en el programa “Excel” mediante la opción de “pegado especial”, se debe seleccionar la opción “separar por comas” y por último guardar el archivo. Este procedimiento se realiza para cada *txt*, que corresponde a determinado tópico.

Luego de guardar el archivo “Excel”, se importan los datos a Matlab mediante el comando:

```
$ Nombre_variable=xlsread(‘nombre_archivo’)
```

Para este caso, el código quedo de la siguiente manera:

```
$ DATA=xlsread('lecturas.xlsx');

$ DATA2=xlsread('Velocidad.xlsx');
```

Donde “DATA” y “DATA2” son las variables correspondientes a “lecturas.xlsx” y “Velocidad.xlsx” respectivamente. El archivo “lecturas.xlsx” contiene los datos del tópico */scan*, que pertenece al sensor Lidar, mientras que “Velocidad.xlsx” es la información del tópico */cmd_vel* y tiene las lecturas de velocidad de los motores.

Para lograr generar el mapa se graficó (x,y) con una selección de datos del sensor Lidar que se encuentran dentro de la variable “fdata”, para el caso de “x” se multiplica por coseno y por la variable “i” que va desde 1 hasta 359, lo que representa los 360°. Después se convierte a radianes y se suma el valor de desplazamiento del robot contenido en “a”, multiplicado por la escala de la gráfica contenida en “vx”. Para el eje “y” se toman los valores de “fdata”, se multiplican por seno, la variable “i” y se convierten a radianes. Las ecuaciones en código quedan de la siguiente manera:

```
$ x(i)=fdata(j,i) * cos((i)* pi/180)+(a*vx);

$ y(i)=fdata(j,i) * sin((i) * pi/180);
```

La toma de datos para los experimentos se realizó dentro de la oficina de maestría en el edificio F del CU UAEM VM, por lo que las gráficas en ambos experimentos muestran el mapa de la oficina mencionada. En la Figura 4.2 se puede apreciar una línea bien definida en el eje x, que va desde el punto inicial (0,0) hasta el punto final (2.5,0), esto representa el desplazamiento del robot dentro de la oficina en forma de avance. Cada punto representa una muestra de 360° del robot en su camino hasta su destino final. Alrededor se aprecia el dibujo del mapa con sus muebles y habitantes.

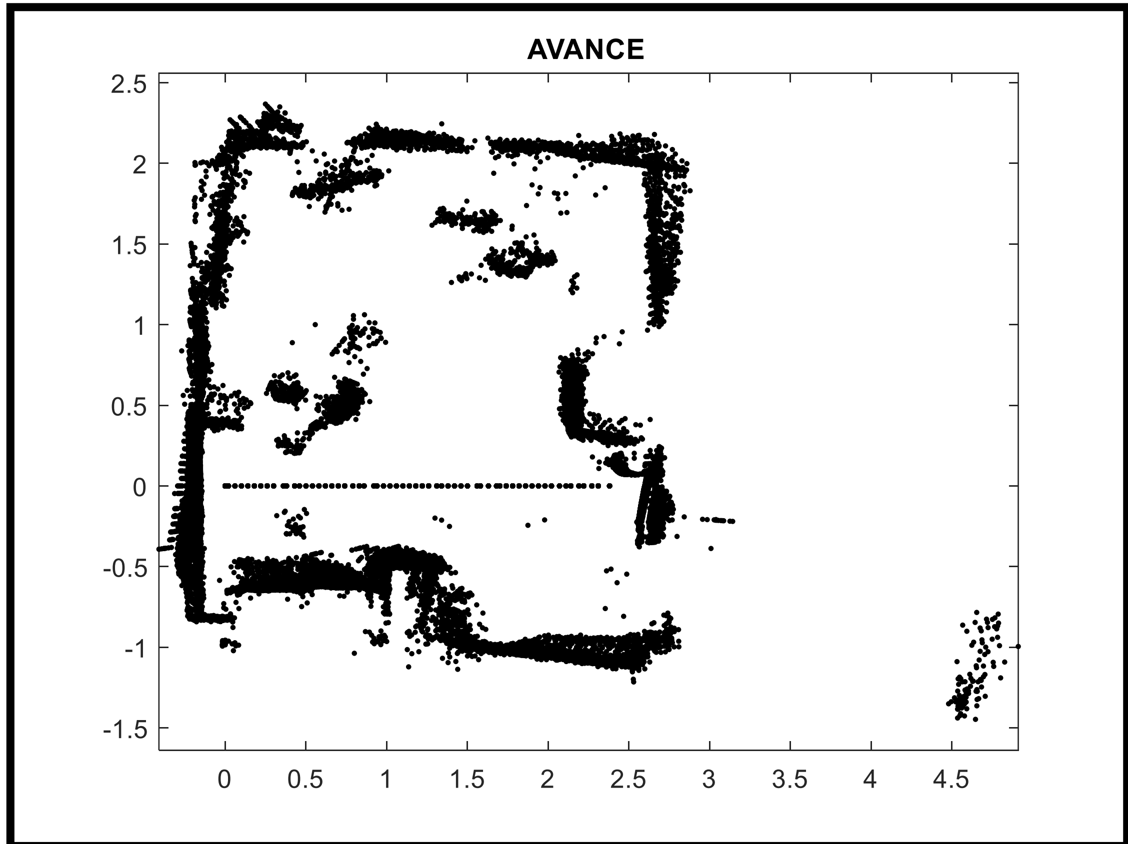


Figura 4.2: Grafica de datos obtenidos del Turtlebot 3 para avance del robot

4.2.3. Interpretación de la información de movimiento rotacional

Para el caso de un movimiento angular se puede observar la Figura 4.3, donde se representa el giro a la derecha del robot. En la figura se puede observar el punto (0,0) que representa al robot. A diferencia del ejemplo de avance, en este experimento se observa que el robot no se desplaza, sino que se mantiene en la misma posición recopilando datos de su ambiente mientras rota en su propio eje.

Para esta escenificación se generan de igual forma los archivos *xlsx* generados a partir del *txt*, que provienen a su vez del *bag*. Las variables quedan de la misma manera que el ejemplo anterior, es decir:

```
$ DATA=xlsread('lecturas.xlsx');

$ DATA2=xlsread('Velocidad.xlsx');
```

Figura 4.3: Grafica de datos obtenidos del Turtlebot 3 para giro del robot

De igual manera se grafica (x,y), con la diferencia de que en el movimiento rotacional se compensa el giro del robot sumando la variable “giro” a la variable “i” que representa los 360° y posteriormente convirtiendo a radianes. Las ecuaciones quedan de la siguiente forma:

```
$ x(i)=fdata(j,i) * cos((i+giro) * pi/180);

$ y(i)=fdata(j,i) * sin((i+giro) * pi/180);
```

4.3. Generación de trayectoria

Existen en la literatura diversos algoritmos para generar trayectorias en mapas 2D, como puede ser A*, Dijkstra, DWA, Campos de potencial, por mencionar solo algunos. Es difícil seleccionar a alguno de estos algoritmos como el mejor, ya que cada uno posee ventajas y desventajas para determinados escenarios, sin embargo, para este caso, se utilizó DWA, el cual es el algoritmo por defecto que se utiliza en el Turtlebot 3 Burger.

DWA proviene del inglés *Dynamic Window Approach*, es un algoritmo propuesto por Dieter Fox (Fox, Thrun, & Burgard, 1997). Básicamente el algoritmo calcula velocidades (lineal y angular), para alcanzar el objetivo o la meta, tomando en cuenta los obstáculos y la dirección u orientación de la meta. DWA se basa en los ejes cartesianos (x,y) y los transforma en velocidades (v,w) las cuales se definen por la velocidad tangencial y angular.

Para elegir la velocidad correcta, se utiliza la siguiente función:

$$G(v, w) = \sigma(\alpha * heading(v; w) + \beta * dist(v; w) + \gamma * vel(v; w)), \tag{4.1}$$

En esta función, *heading* será la velocidad que maximice la orientación hacia la meta, la distancia *dist* deberá ser mayor entre la trayectoria y el obstáculo para evitar choque con algún obstáculo y *vel* representa la velocidad de avance, la cual deberá ser la más rápida posible. Por otra parte, los símbolos sigma (σ), beta (β) y gamma (γ) representan un peso, el cual para este caso se define por defecto de la siguiente manera (Zheng, 2017):

$$\sigma = 32.0 \tag{4.2}$$

$$\beta = 20.0 \tag{4.3}$$

$$\gamma = 0.02 \tag{4.4}$$

Después de contar con el mapa, el siguiente paso es generar la ruta que han de seguir ambos robots. Para lograrlo se ejecuta en la computadora "máster" el experimento que genera la trayectoria.

Este experimento se llevó a cabo de manera virtual con la herramienta rviz de ROS, la cual es un visualizador de datos que permite generar la trayectoria con ayuda del algoritmo DWA, simulando un Turtlebot 3 Burger desplazarse en un mapa de un entorno creado con

anterioridad. Al mismo tiempo se ejecutó de manera física, es decir, se colocó el robot comercial en el entorno controlado y con obstáculos para realizar la trayectoria de manera real.

La Figura 4.4 muestra la generación de trayectoria, para este caso se utilizó un mapa de un escenario con obstáculos representados en la imagen por las líneas y puntos en color negro. Se puede observar al Turtlebot 3 en la esquina superior derecha enmarcado por un rectángulo y una línea azul, la cual dibuja la trayectoria que llega hasta la meta establecida manualmente, representada por una flecha de color rojo. Los datos del sensor Lidar en el Turtlebot3 se observan en color verde y amarillo.

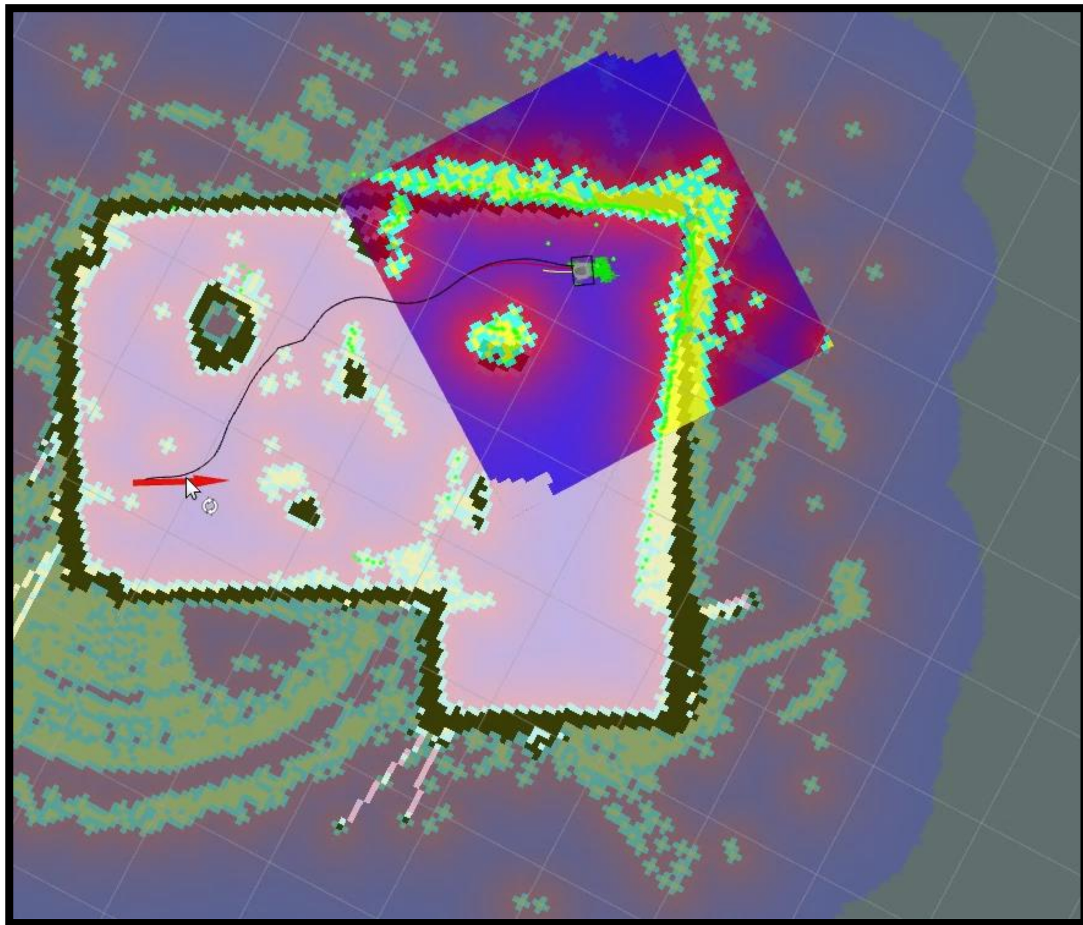


Figura 4.4: Generación de trayectoria en la computadora “máster” con la herramienta rviz con el algoritmo DWA

Los pasos para realizar trayectorias como lo descrito anteriormente son:

En una nueva terminal en el "máster" se introduce el comando:

```
$ roscore
```

En el Turtlebot 3 se ejecuta el comando:

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Nuevamente en la Pc se ejecuta en una nueva terminal:

```
$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_bringup turtlebot3_remote.launch

$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch
map_file:=/home/marco/map.yaml
```

Después de introducir los comandos, se abrirá el visualizador donde se debe hacer clic en la opción "2D Pose Estimate" para indicar la posición y orientación del robot dentro del mapa. Posteriormente se debe establecer la meta o punto final, para lograrlo se debe dar clic sobre la opción "2D Nav Goal" y entonces el robot comienza a desplazarse en el entorno evitando obstáculos haciendo uso del algoritmo DWA para evitar los obstáculos.

4.4. Desarrollo del segundo *sketch* de Arduino para el seguimiento de trayectoria

Para lograr el seguimiento de la trayectoria a partir de la información generada por el robot comercial, se programó un segundo *sketch* en el IDE de Arduino (Figura 4.5), el cual crea un nodo subscritor para el tópico *cmd_vel*, dicho tópico contiene las velocidades para cada motor (izquierdo y derecho) y también el sentido de giro de las ruedas para el desplazamiento de los robots. Con base en una ecuación y una escala de velocidades, se asignan valores para lograr la correcta locomoción en el robot de armado propio y ejecutar el seguimiento de una trayectoria de manera correcta, evitando obstáculos y alcanzando una meta.

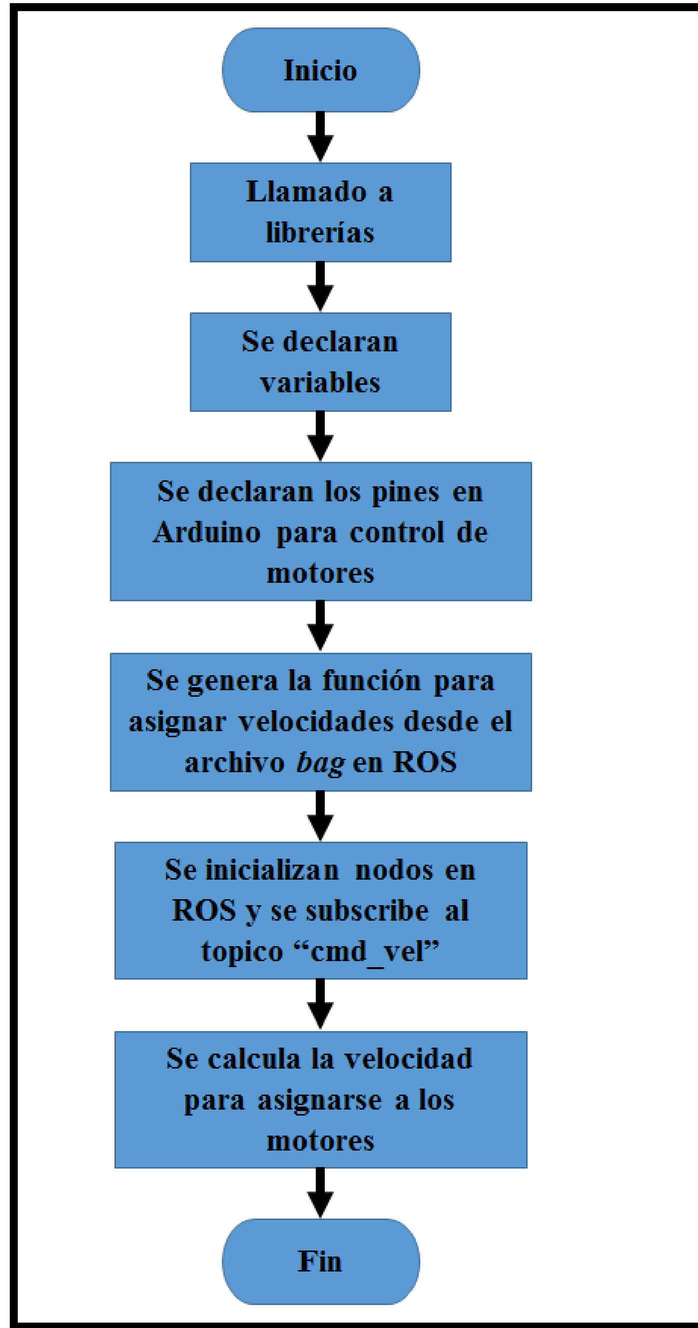


Figura 4.5: Diagrama de flujo del segundo *sketch* de Arduino para ROS

4.5. Calibración de movimientos: lineal y rotacional

Para realizar las pruebas de seguimiento de trayectoria, era necesario ajustar las velocidades del robot de armado propio. Esto se logró ajustando las velocidades PWM en cada motor (izquierdo y derecho), asemejando los movimientos como los hace el Turtlebot 3. Además, en esta etapa se hizo uso del segundo *sketch* de Arduino, el cual tiene la capacidad de suscribirse al tópico *cmd_vel* para asignar velocidades a los motores de los robots.

Dado que el robot propio funciona con motores de corriente continua, los cuales dependen de un valor PWM asignado desde Arduino y que dicho valor se encuentra dentro del rango [0, 255] donde cero representa el valor mínimo y doscientos cincuenta y cinco el valor máximo, se llevó a cabo un ajuste de la velocidad para que correspondiera a la que maneja el robot de experimentación comercial.

El ajuste se hizo mediante una escala de velocidades, partiendo de la fórmula de la pendiente:

$$M = \frac{Y_2 - Y_1}{X_2 - X_1} \tag{4.5}$$

Para hacer las escalas se tomaron en cuenta las velocidades del Turtlebot 3, las cuales son para el movimiento lineal de 1.5 y para el rotacional de 0.4 en la escala del mismo robot. Estos valores se multiplicaron por 10 para hacer su valor entero y lograr así un manejo más fácil dentro del *sketch* de Arduino. Derivado de esto, resultó que para el movimiento lineal la escala tendría 15 valores y para el rotacional serían 4 valores.

La fórmula quedó de la siguiente manera:

$$Vel = \frac{(Y_2 - Y_1)}{(X_2 - X_1)} * (valor\ -\ X_1) + Y_1 \tag{4.6}$$

Donde:

Vel es el valor PWM que se asigna a los motores.

Y_2 es la velocidad máxima PWM de desplazamiento.

Y_1 es la velocidad mínima PWM de desplazamiento.

X_2 es el valor máximo de la escala.

X_1 es el valor mínimo de la escala.

valor es la velocidad que proviene del Turtlebot 3 y que se multiplica por 10.

El valor para la escala PWM se fijó en 70, iniciando así las pruebas de desplazamiento en ambos robots para comprobar que su funcionamiento fuera lo más parejo posible. Durante los ensayos se incrementó y decremento el valor PWM para observar el comportamiento del robot de armado propio. Se encontró que con el valor de 60 PWM el robot propio se comportaba de manera semejante al robot comercial, por lo que las pruebas posteriores se ejecutaron con este valor.

4.5.1. Calibración lineal

Para llevar a cabo la calibración lineal se generó una bolsa de datos con el Turtlebot 3. En la Figura 4.6 se aprecia el escritorio de la computadora “master” ejecutando ROS y la bolsa de datos en diferentes terminales.

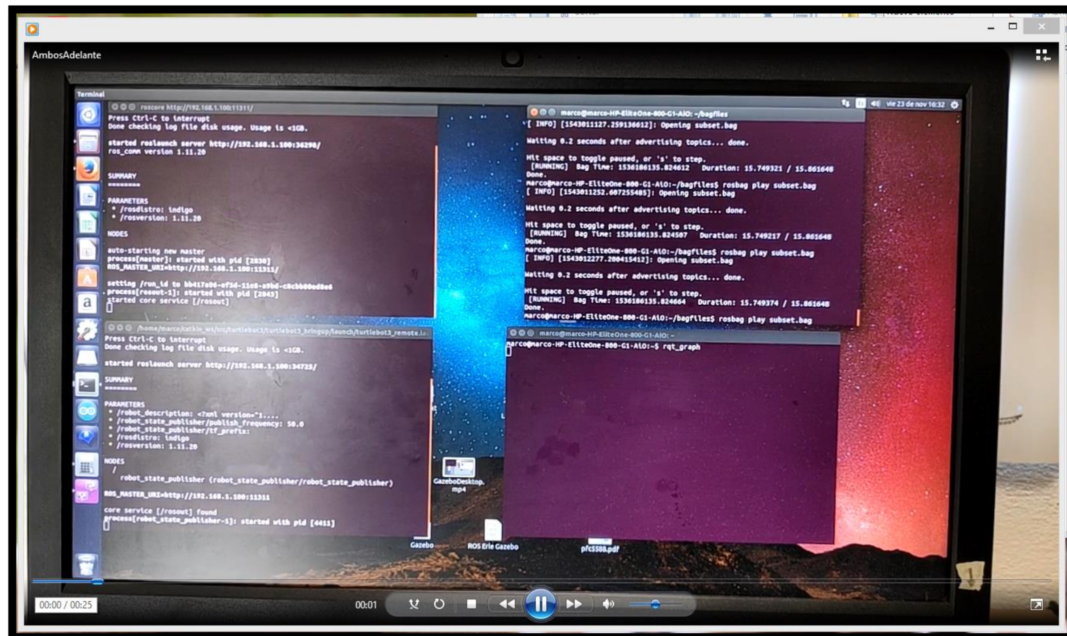


Figura 4.6: Calibración lineal de los robots. Secuencia 1 de 4

En la Figura 4.7 se observa a los robots colocados de manera paralela detrás de una marca de cinta roja en el piso. Los robots se encuentran listos para ejecutar los datos contenidos en la bolsa desde la computadora “master”.

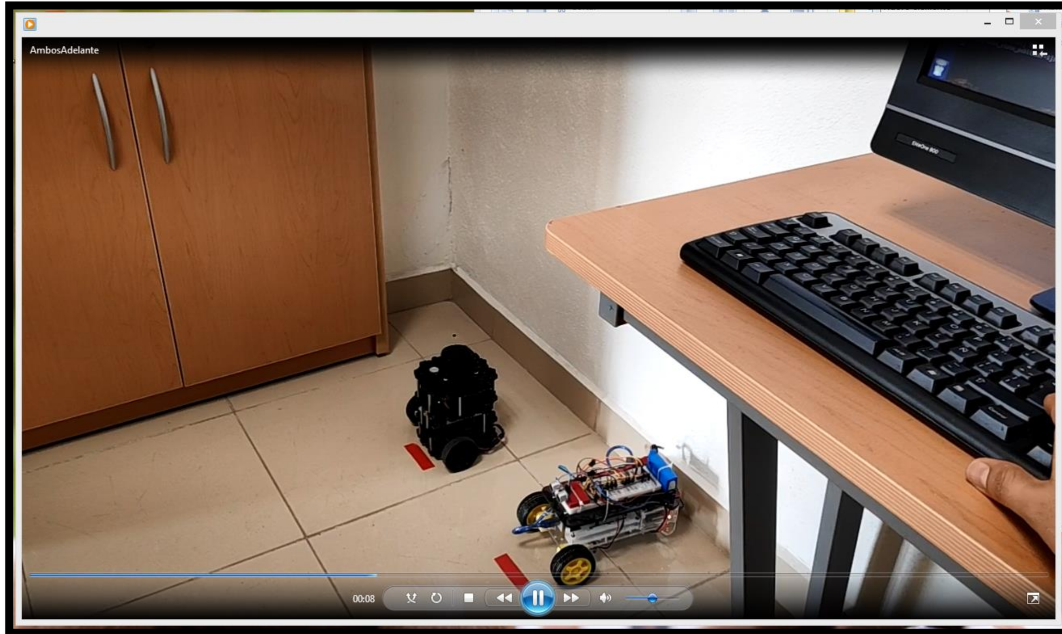


Figura 4.7: Calibración lineal de los robots. Secuencia 2 de 4

La Figura 4.8 muestra a los robots ejecutando los datos de la bolsa para calibrar el movimiento lineal. Para este experimento, ambos robots se desplazan hacia el frente con velocidades similares.

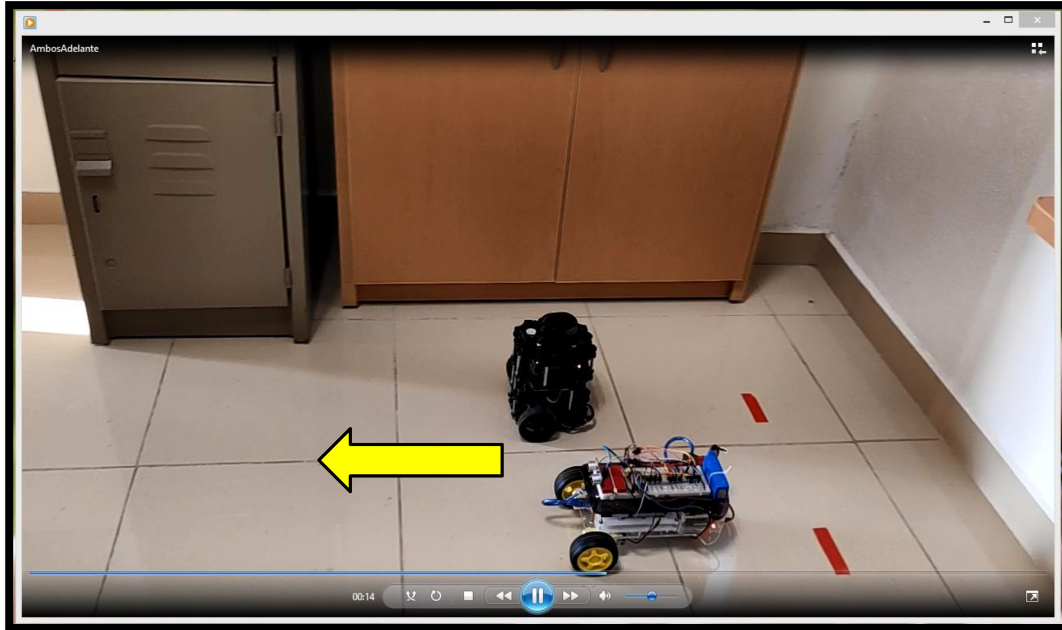


Figura 4.8: Calibración lineal de los robots. Secuencia 3 de 4

En la Figura 4.9 ambos robots continúan desplazándose con base en los datos de la bolsa generada para la calibración de velocidad. Se puede observar que presentan un avance similar y por lo tanto se consideran calibrados en su movimiento lineal.

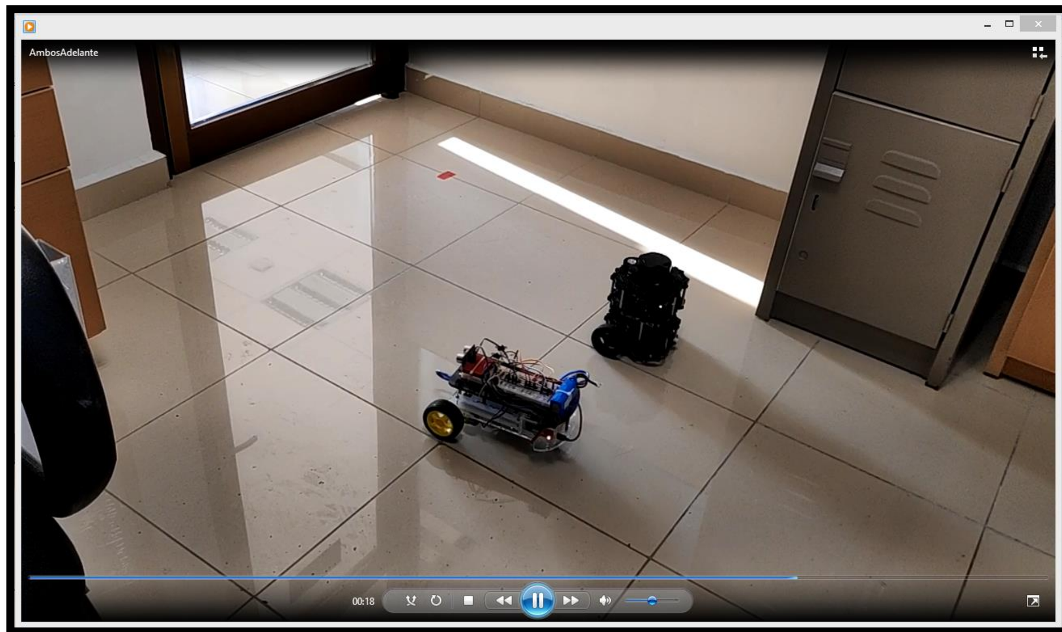


Figura 4.9: Calibración lineal de los robots. Secuencia 4 de 4

4.5.2. Calibración rotacional.

En el caso de la calibración rotacional, se hizo una bolsa de datos con el Turtlebot 3 para reproducirla después de manera simultánea en ambos robots. En la Figura 4.10 se puede ver el escritorio de la computadora “master” ejecutando ROS y la bolsa con los datos para el experimento.

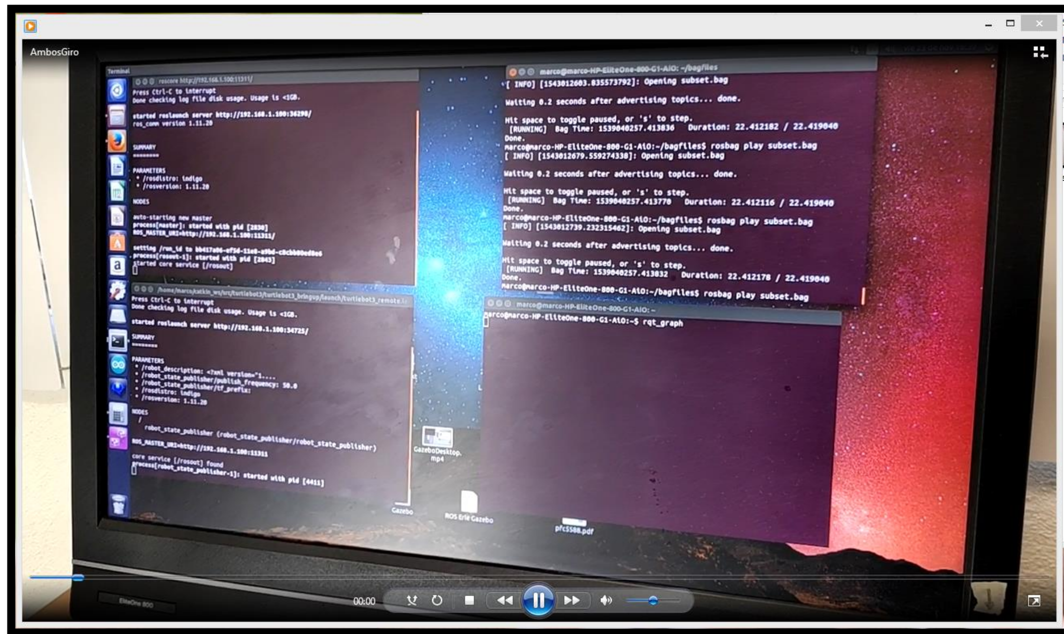


Figura 4.10: Calibración rotacional de los robots. Secuencia 1 de 4

En la Figura 4.11 se observa a los robots colocados de manera paralela, listos para ejecutar los datos desde la bolsa generada para este experimento de giro.

Figura 4.11: Calibración rotacional de los robots. Secuencia 2 de 4

Al ejecutar la bolsa desde la computadora “master”, los robots comienzan a reproducir las velocidades contenidas en ella. Para este experimento el giro que realizan ambos es hacia su derecha. En la Figura 4.12 se aprecia a los robots superando los 90° de giro.

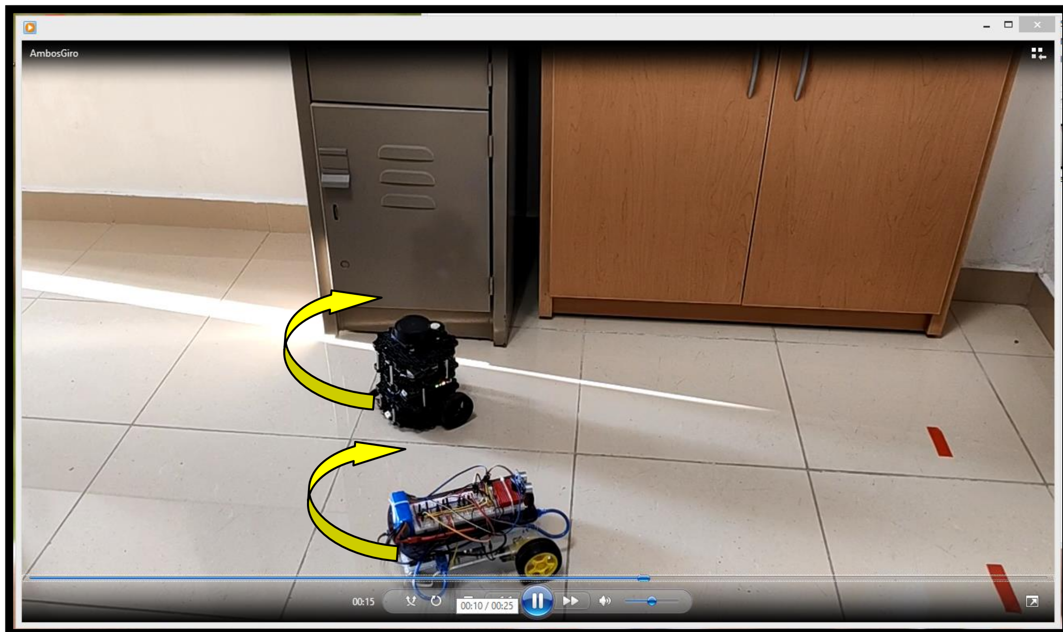


Figura 4.12: Calibración rotacional de los robots. Secuencia 3 de 4

Para finalizar, se presenta en la Figura 4.13 a los robots concluyendo el giro contenido en la bolsa para este propósito. Como se puede observar ambos terminan la reproducción de los datos de manera correcta y con esto se consideran calibrados en el movimiento rotacional.

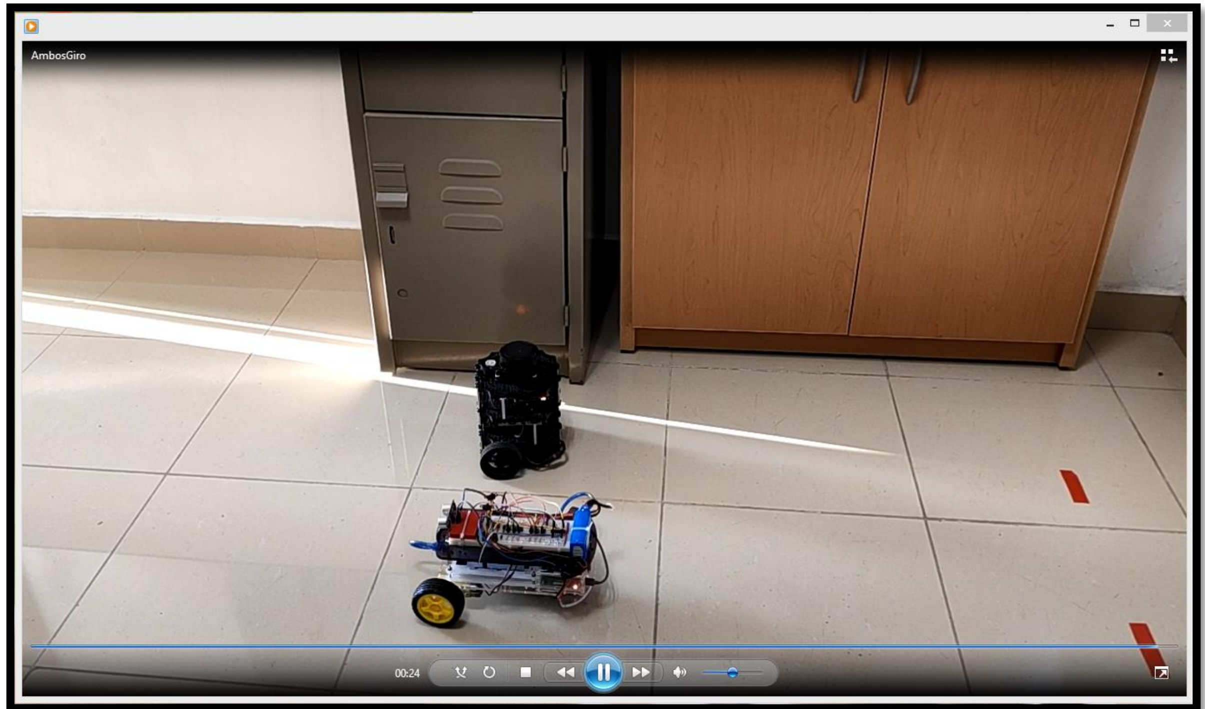


Figura 4.13: Calibración rotacional de los robots. Secuencia 4 de 4

4.6. Seguimiento de trayectorias

Para realizar las pruebas, se elaboraron dos entornos controlados con obstáculos en diferentes posiciones y distancias (Figura 4.14). Los entornos se exploraron con el Turtlebot 3 para recopilar información y generar los mapas correspondientes. Después se generaron las trayectorias a seguir con el algoritmo DWA ("trayectoria1.bag" y "trayectoria2.bag"), almacenándolas en la computadora "master" para posteriormente transmitirse a ambos robots en experimentos separados.

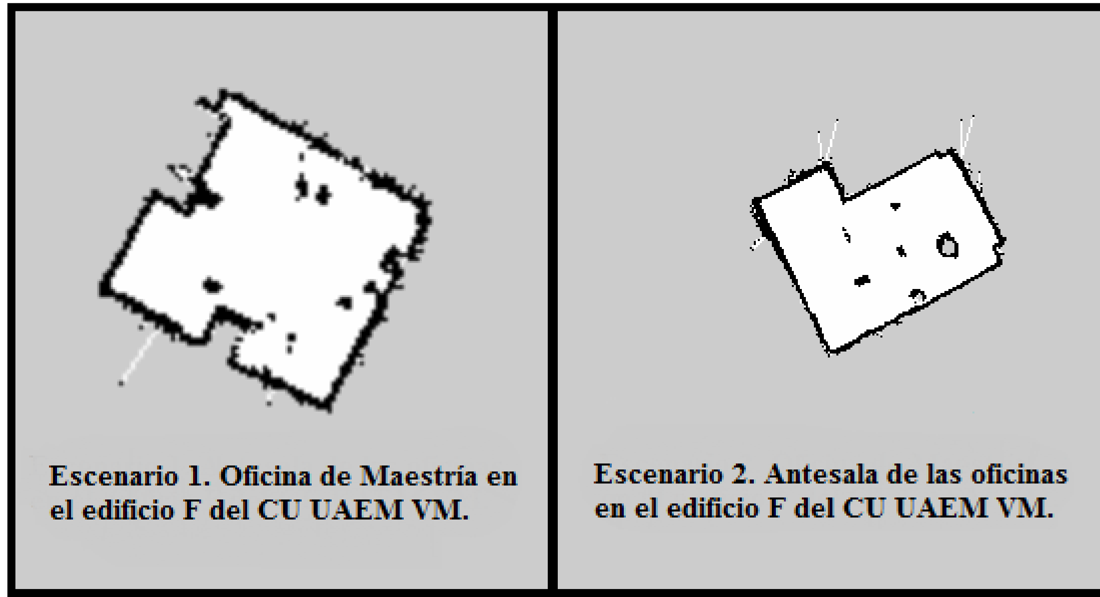


Figura 4.14: Mapas de los escenarios de pruebas de trayectorias

4.6.1. Caso de estudio uno

Con los robots calibrados y los mapas guardados, se realizaron experimentos de seguimiento de rutas y evasión de obstáculos, para los cuales se generaron bolsas de datos que se compartieron en ambos robots. El primer experimento se hizo en el escenario 1, primero con el robot comercial y enseguida con el robot propio.

Para ejecutar la trayectoria en el Turtlebot 3, se siguieron los siguientes pasos:

Se ejecuta en la computadora "máster" en una nueva terminal el comando:

```
$ roscore
```

En el Turtlebot 3 se ejecuta el siguiente comando en una terminal:

```
$ roslaunch turtlebot3_bringup turtlebot3_remote.launch
```

Nuevamente en el "máster" en una nueva terminal se ejecuta la bolsa que contiene la información de la ruta:

```
$ rosbag play trayectoria1.bag
```

La Figura 4.15 muestra los nodos y los tópicos activos en el experimento de seguimiento de trayectoria con el Turtlebot 3. La bolsa de datos está representada por el nodo

play_1542996637233921624, el cual se encuentra publicando a través del tópico */cmd_vel* hacia el nodo del Turtlebot 3 llamado */turtlebot3_core*. Los otros nodos de la figura son propios del robot comercial y para el experimento son irrelevantes ya que solo se ocupan los descritos para el seguimiento de la trayectoria.

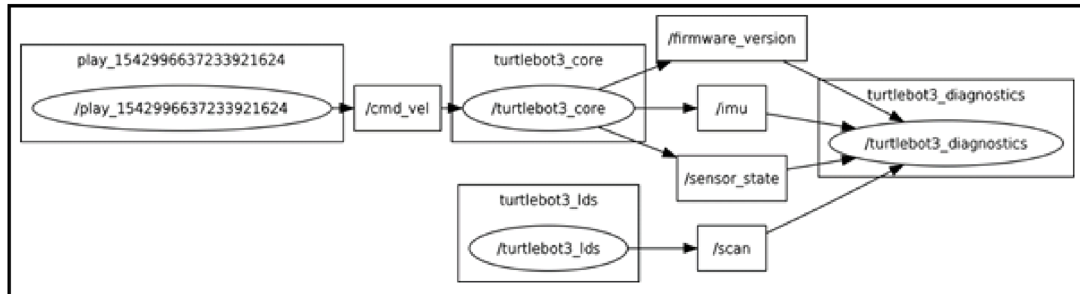


Figura 4.15: Diagrama de nodos y tópicos para seguimiento de trayectoria en Turtlebot 3

En el caso del robot propio, el experimento se corre de manera muy parecida al robot comercial, en la computadora "máster" se ejecuta:

```
$ roscore
```

Posteriormente en el robot de armado propio se introduce en una nueva terminal el comando:

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Después en la computadora "máster" se ejecuta el archivo bag que contiene la información de la trayectoria, en una terminal se introduce:

```
$ rosbag play trayectoria1.bag
```

La Figura 4.16 muestra el nodo de la bolsa *play_1543008380804222129* que transmite el tópico *cmd_vel*, al cual está subscrito el nodo del robot propio */serial_node* para tomar las velocidades y lograr desplazarse dentro del entorno.

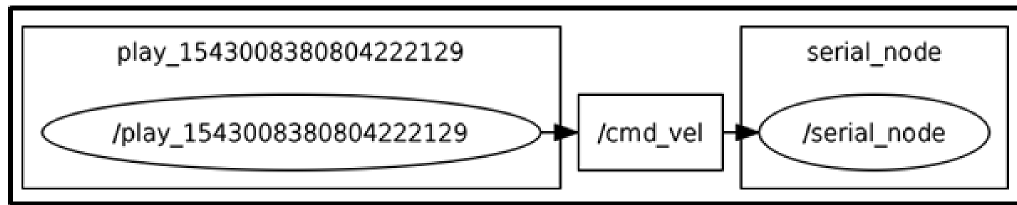


Figura 4.16: Diagrama de nodos y tópicos para seguimiento de trayectoria en robot propio

La Figura 4.17 muestra el escritorio de la computadora "master" ejecutando ROS y la bolsa de datos para el seguimiento de trayectoria en los robots de manera separada. El escritorio superior corresponde al Turtlebot 3 mientras que el inferior es del robot propio, en este caso se puede observar que está corriendo la gráfica de nodos y tópicos activos.

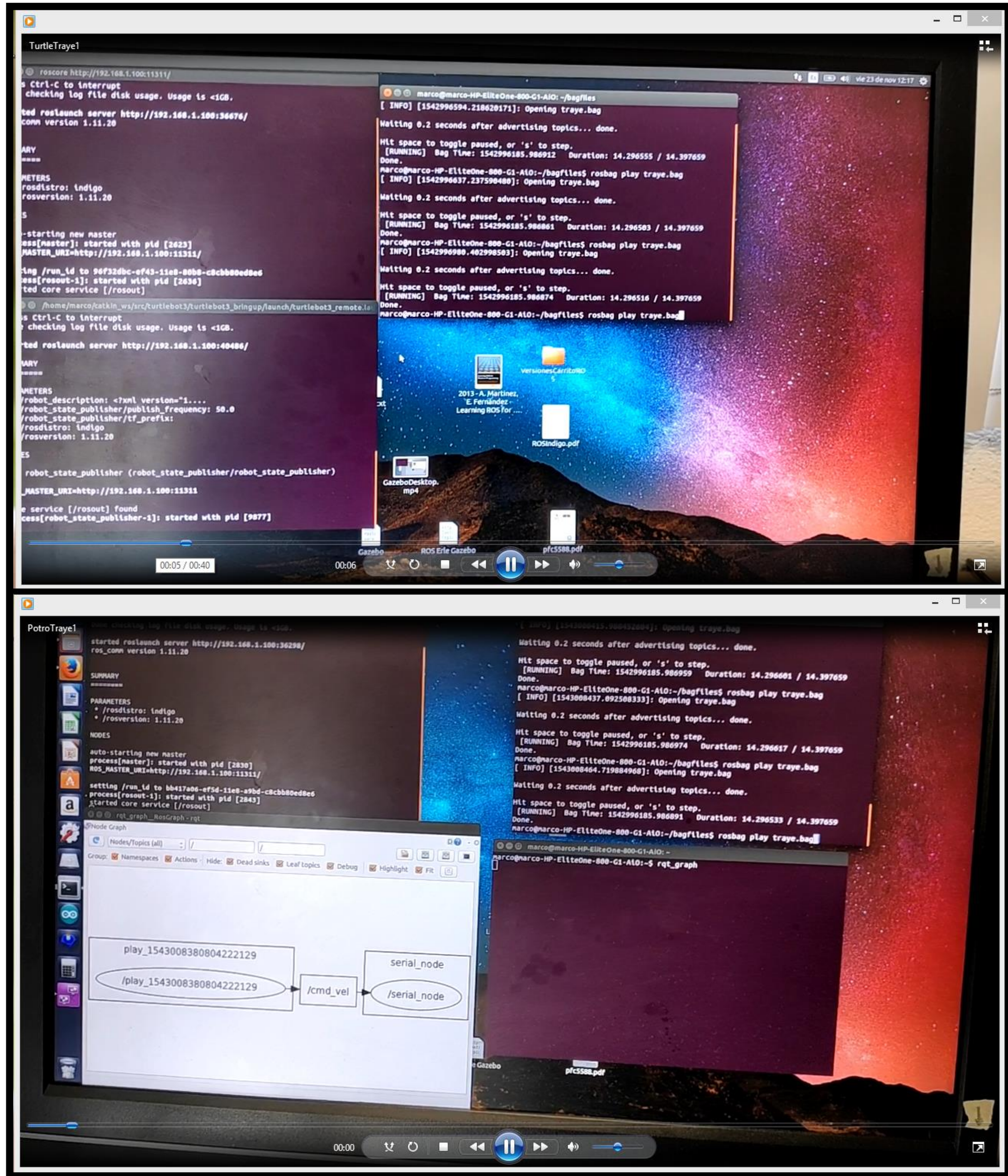


Figura 4.17: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 1 de 5

En la Figura 4.18 se puede observar a ambos robots colocados en el punto de inicio listos para comenzar a reproducir la trayectoria trasmitida desde la computadora “master”. En este caso se señaló el inicio y la meta con cinta adhesiva de color rojo.

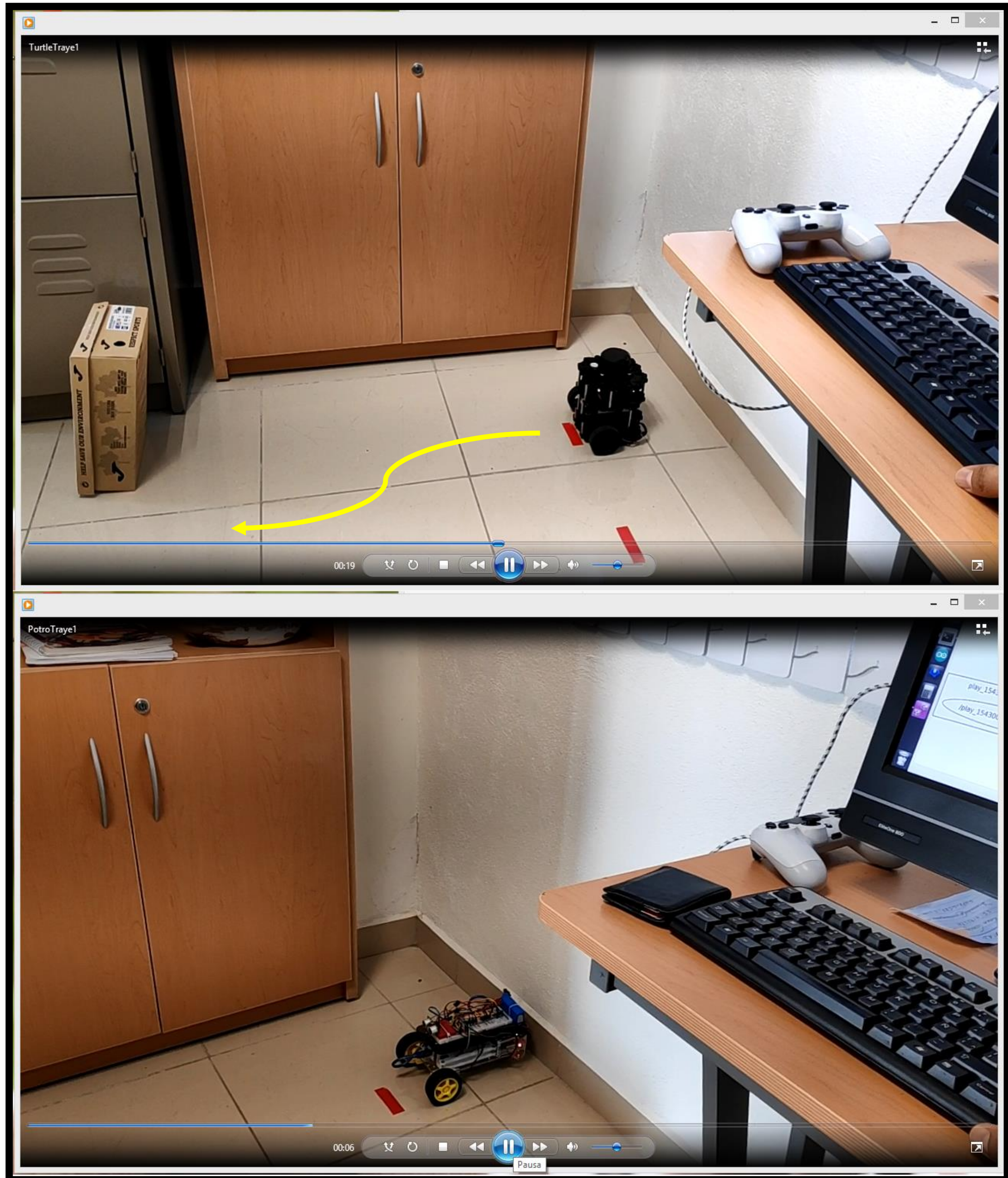


Figura 4.18: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 2 de 5

La Figura 4.19 muestra como los robots evaden el primer obstáculo con base en la información obtenida desde la bolsa. Las líneas amarillas representan una aproximación de la trayectoria.

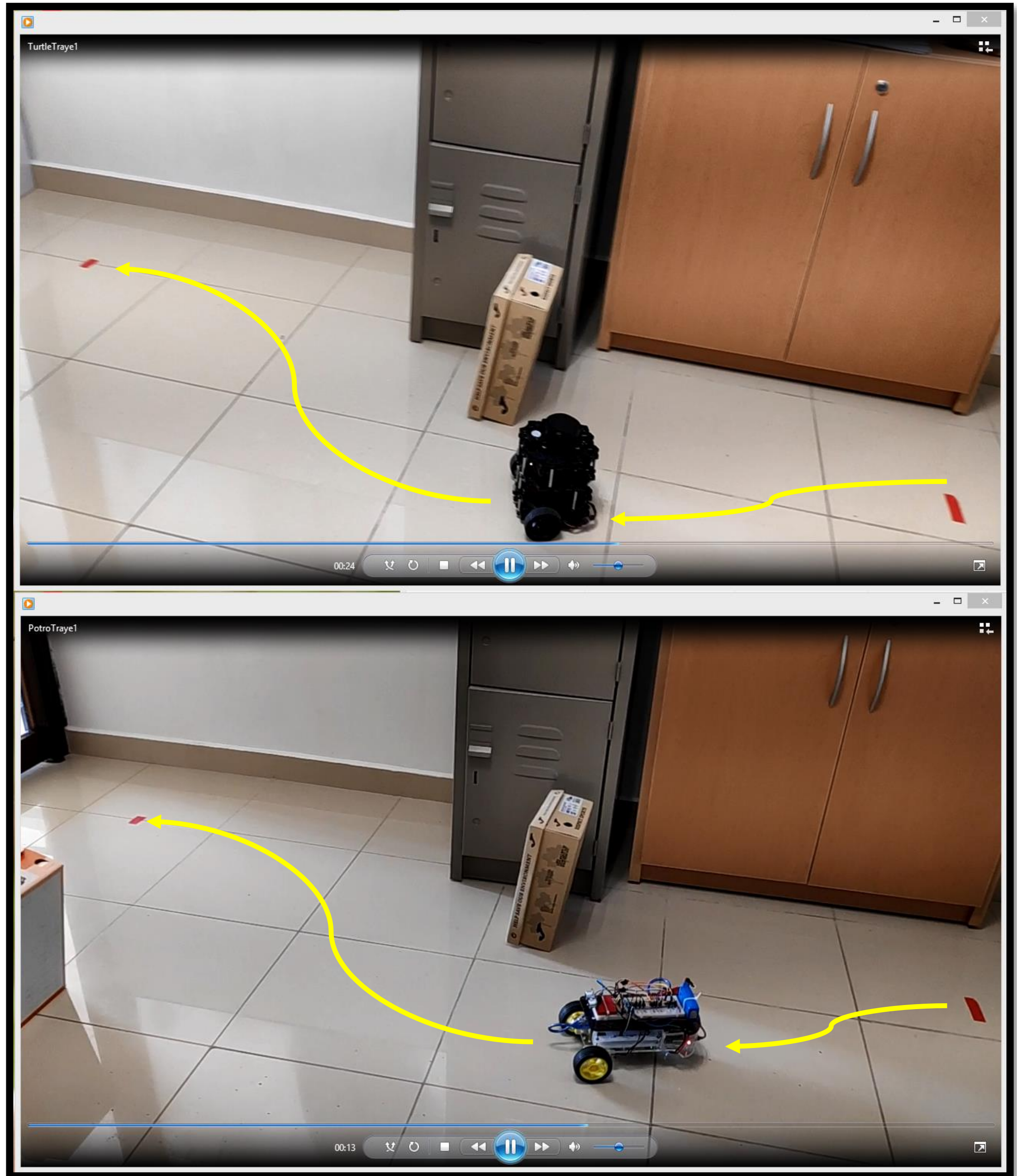


Figura 4.19: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 3 de 5

Posteriormente, se observa en la Figura 4.20 a los robots perfilarse hacia la meta marcada con la cinta roja y evadiendo el segundo obstáculo.

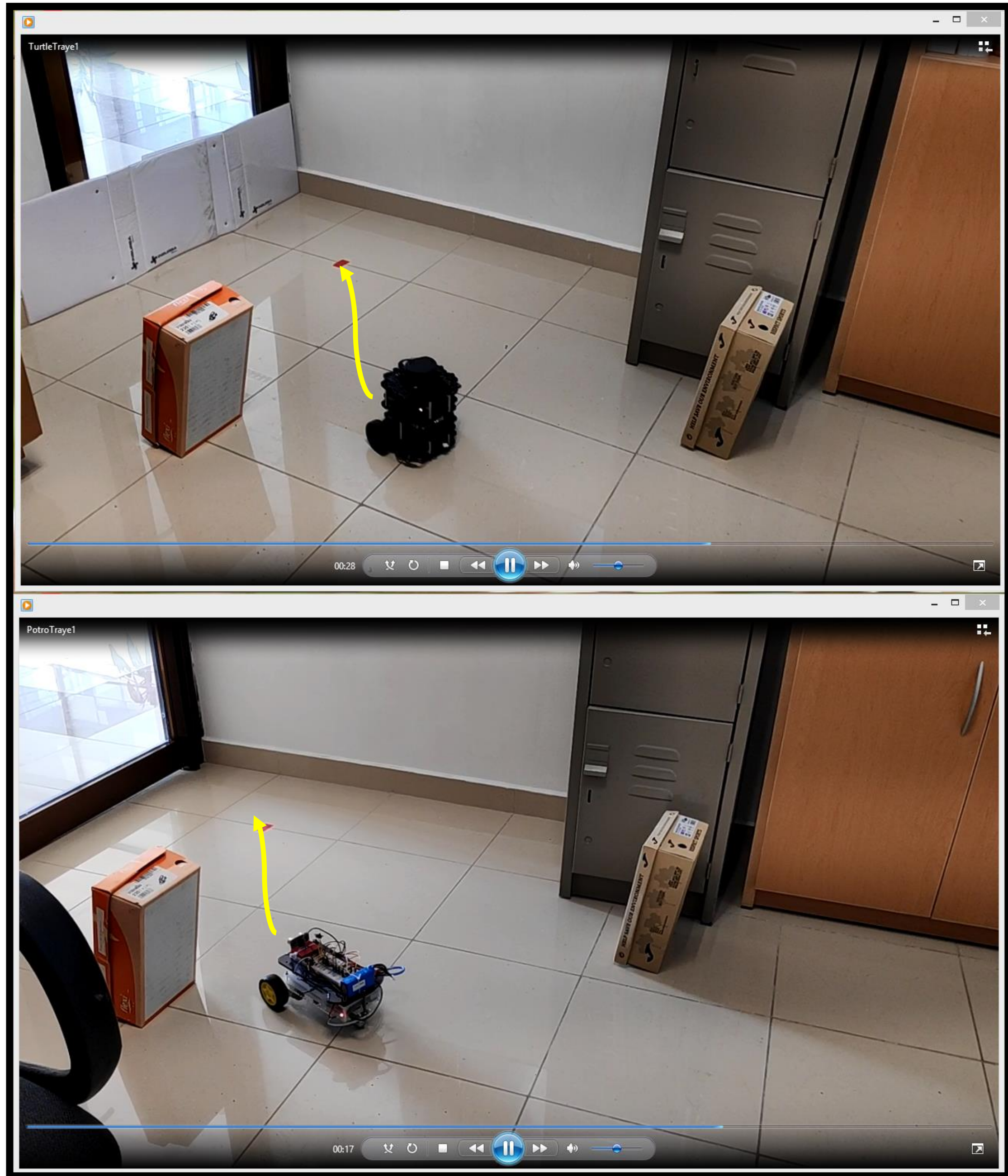


Figura 4.20: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 4 de 5

Finalmente se puede ver en la Figura 4.21 a los robots alcanzando la meta sin colisiones y desempeñándose de manera similar durante el recorrido.

Figura 4.21: Ejecución de trayectoria en el escenario 1 de ambos robots. Secuencia 5 de 5

4.6.2. Caso de estudio dos

Este experimento se realizó de manera similar al anterior, pero en el escenario dos. Se ejecutó de manera separada en los robots, en este caso el punto de inicio fue la puerta de la oficina de maestría y la meta fue la puerta de la antesala de las oficinas en el edificio F del CU UAEM VM.

Como primer paso se preparó la computadora “master” para la ejecución de la trayectoria. En una terminal se corrió ROS y en otra terminal la bolsa con las velocidades de la trayectoria. En la Figura 4.22 se observa a los robots colocados en el punto de salida, listos para comenzar la reproducción de los datos contenidos en la bolsa. La flecha roja representa la ruta que deberían seguir los robots para llegar a la meta.

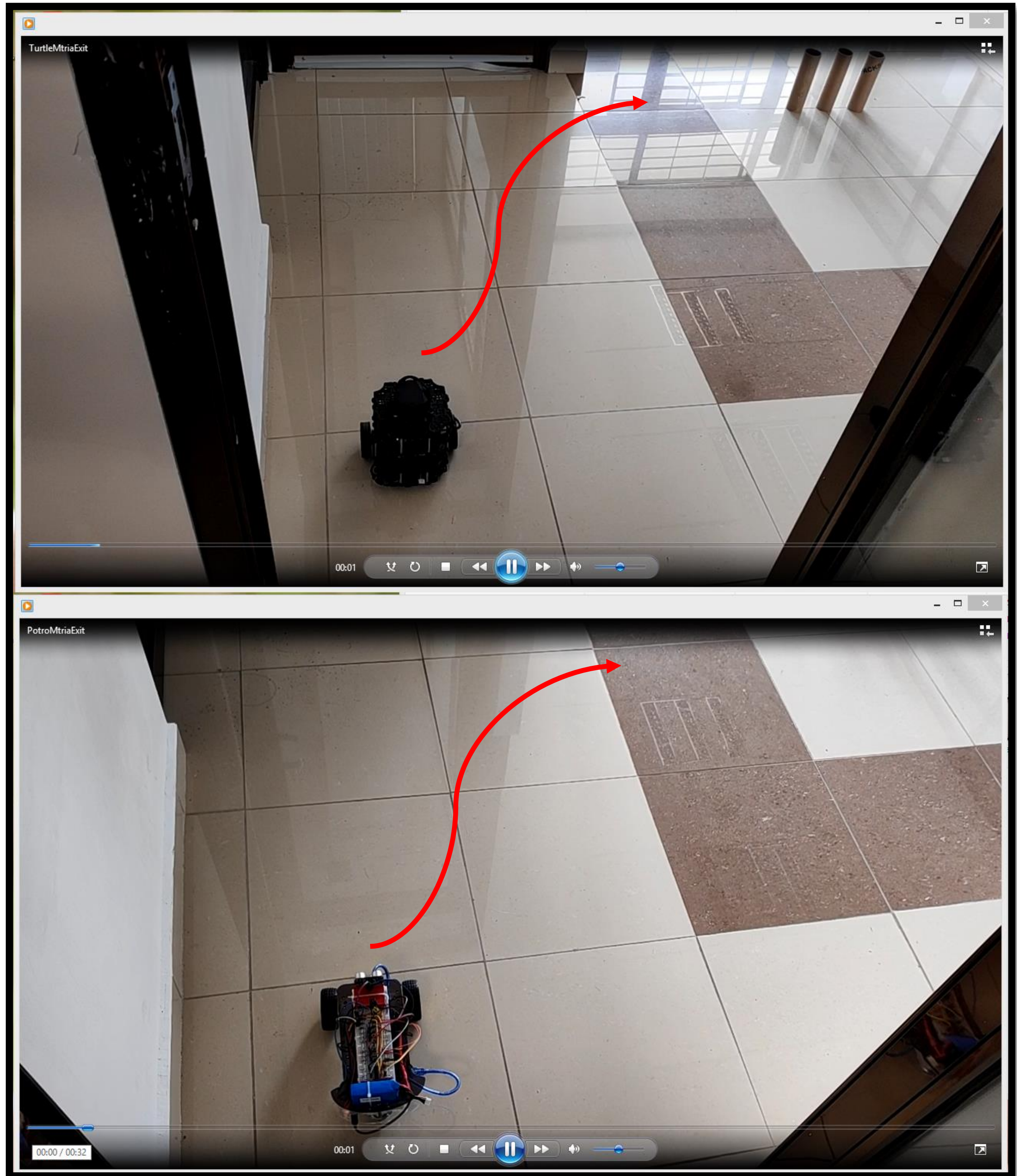


Figura 4.22: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 1 de 5

En la Figura 4.23 se observa a los robots ejecutando la trayectoria y perfilándose hacia la meta a través de los obstáculos.

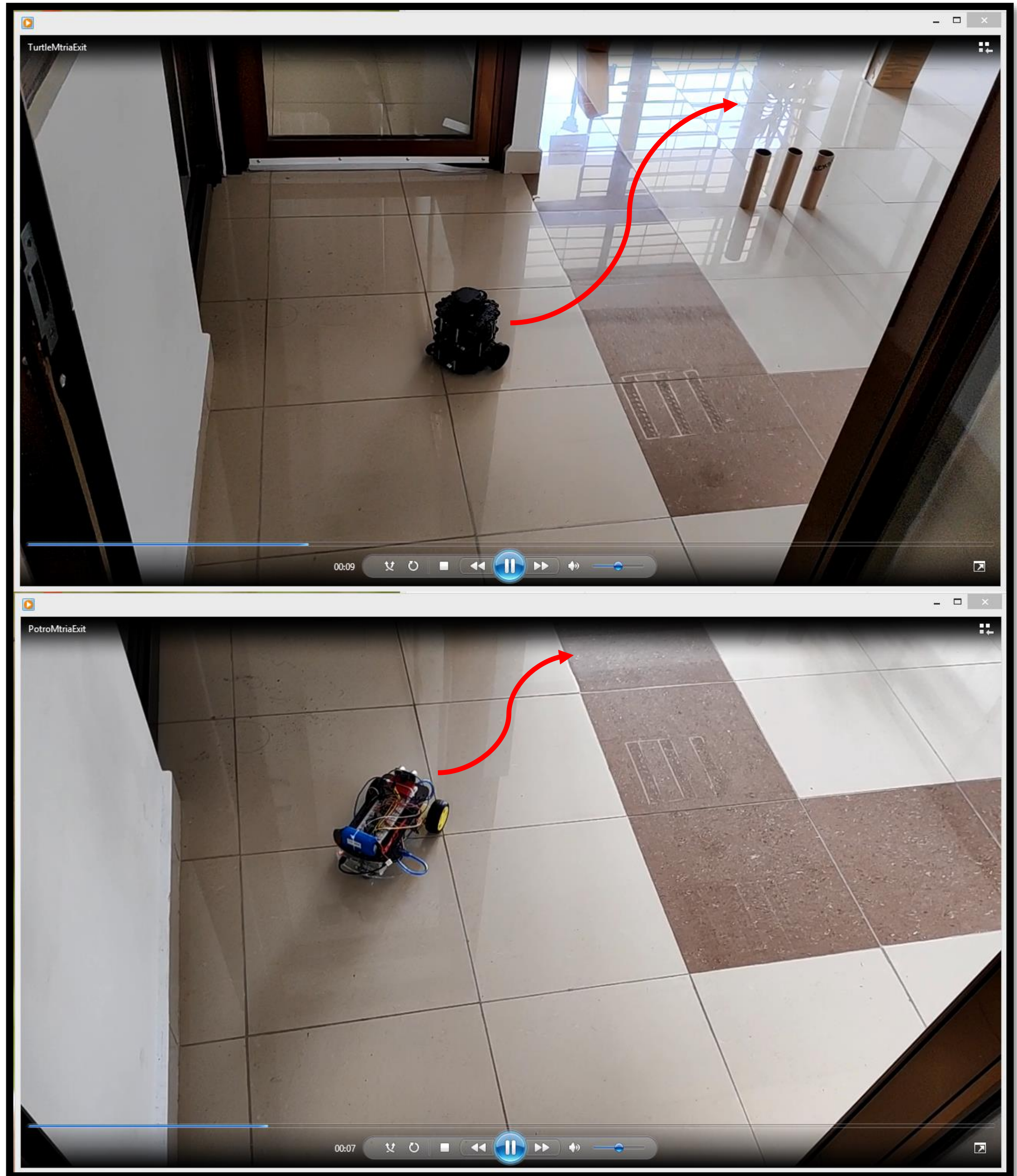


Figura 4.23: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 2 de 5

Posteriormente, en la Figura 4.24 se puede observar a los robots pasando en medio de los obstáculos sin colisiones y avanzando hacia la puerta de las oficinas.

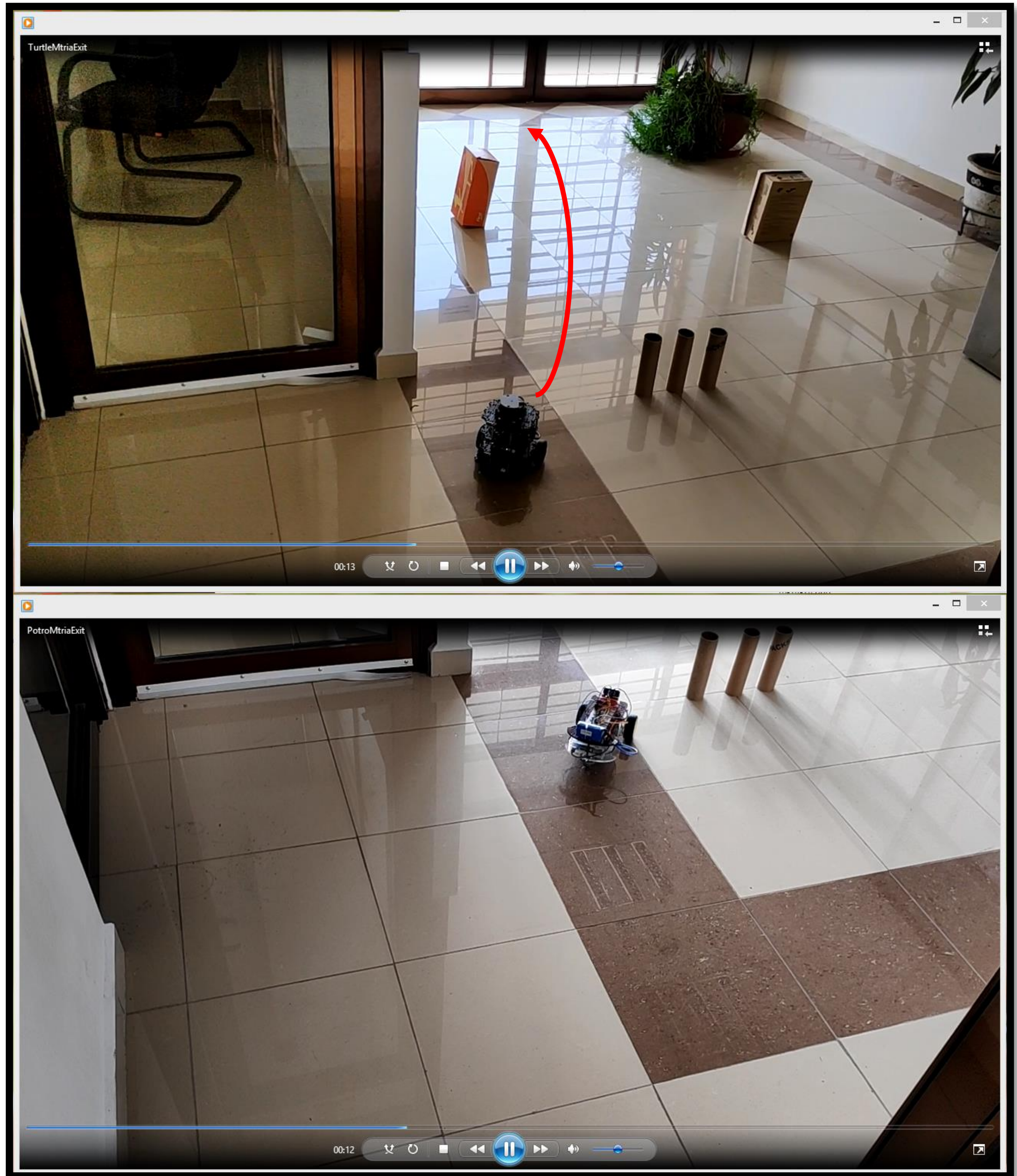


Figura 4.24: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 3 de 5

Ambos robots atraviesan el escenario evadiendo los obstáculos y acercándose a la meta como se puede ver en la Figura 4.25.

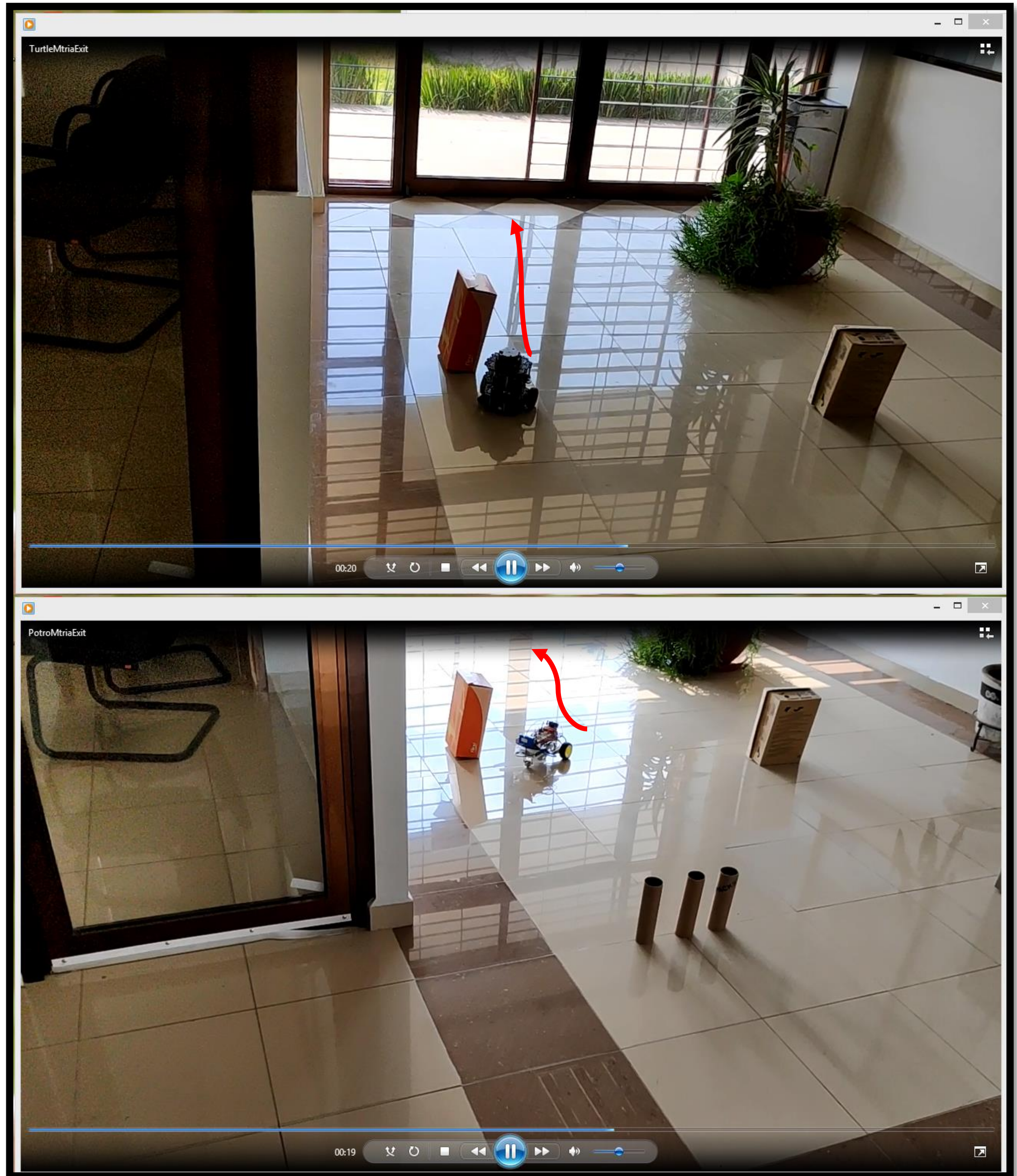


Figura 4.25: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 4 de 5

Después de llegar al otro lado de la antesala, se puede visualizar en la Figura 4.26 como los robots se acercan a su meta desempeñando movimientos y velocidades similares.

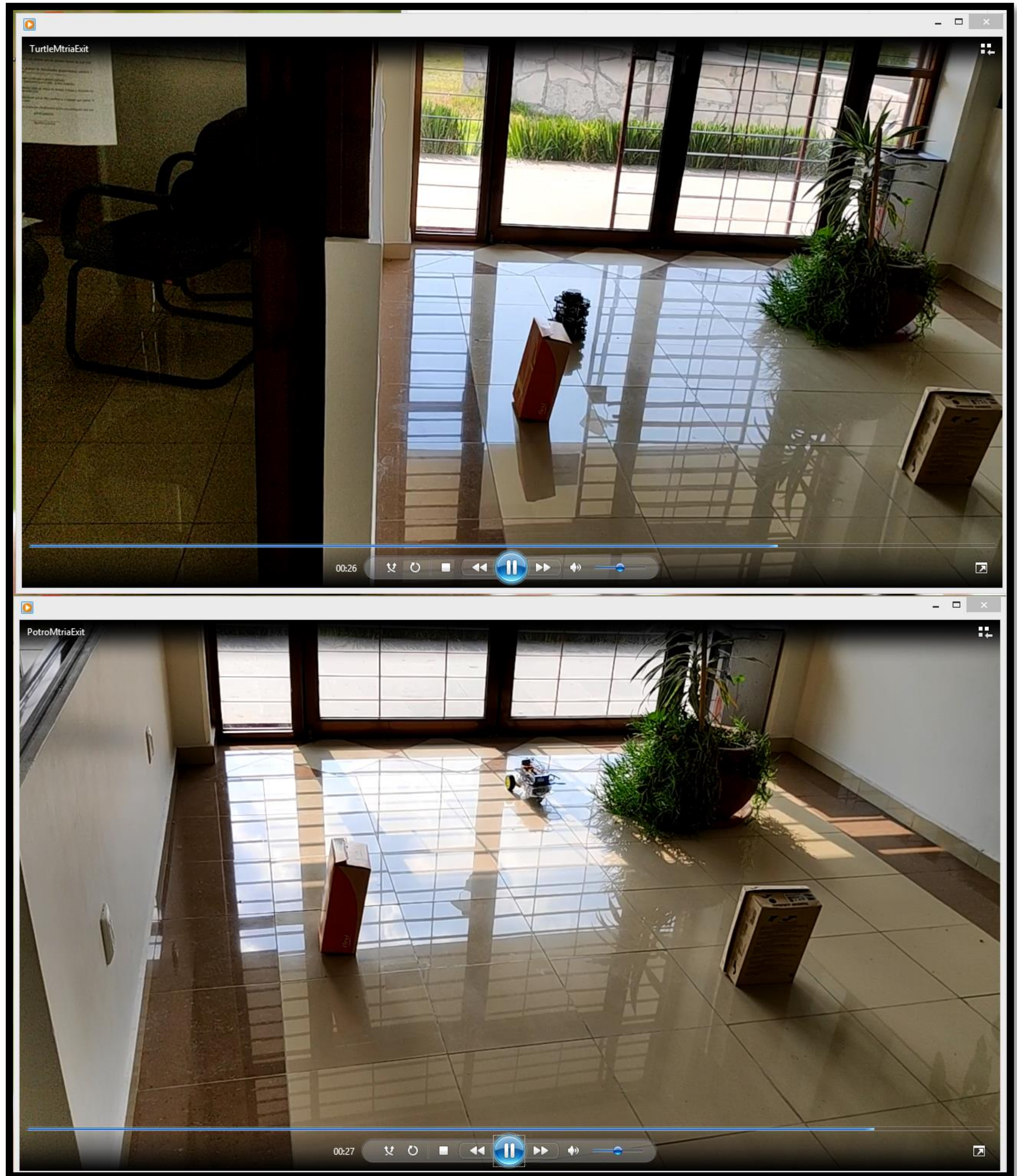


Figura 4.26: Ejecución de trayectoria en el escenario 2 de ambos robots. Secuencia 5 de 5

4.7. Discusión de resultados

Durante el desarrollo de este trabajo se hizo uso de ROS en ambos robots, lo que permitió generar mapas 2D y simularlos gracias a las herramientas que ofrece, también se generaron trayectorias con ayuda del algoritmo DWA y Turtlebot 3. Lo anterior fue posible por la

filosofía, el soporte y la comunidad que tiene ROS, su estructura permite realizar cosas que hasta hace unos años era complicado y requería de mucho tiempo para lograse, como lo realizado en este trabajo.

Después de los experimentos y los resultados obtenidos, se comprueba que es posible contar con robots de capacidades inferiores que "aprendan" tareas que por sí mismos no pudieran realizar. Las actividades para robots móviles con características de *hardware* y *software* similares a las de este proyecto pueden ser variadas, por ejemplo, en (Mohanarajah, 2014) se describe un caso de robots asistentes en hospitales, donde un robot con capacidad de generar mapas y trayectorias comparte dicha información haciendo uso de la plataforma *RoboEarth* para que después otro robot la consulte y se desplace dentro de una habitación asistiendo a un paciente en cama. Otro caso es descrito en (Kamei, Nishio, & Hagita, 2012), donde con base en información de mapas y trayectorias, un robot silla de ruedas asiste a personas de la tercera edad o con discapacidad transportándolos en un supermercado.

Robótica en la nube brindará a los robots muchos beneficios como los da el *Cloud Computing* a las personas actualmente, es cierto que el concepto todavía no es una realidad y que faltan muchos puntos por atender, como puede ser la seguridad de estas plataformas o la infraestructura para almacenar y procesar toda esta información que se creará, pero al día de hoy es interesante ver como estos desarrollos tecnológicos hacen que la robótica se acerque más a las personas.

Capítulo 5: Conclusiones

En la elaboración de este proyecto se logró analizar y comprender el concepto de robótica en la nube, para posteriormente ejemplificar su funcionamiento mediante una plataforma de comunicación entre dos robots.

El sistema de interacción entre robots que se presenta en este proyecto plantea el desarrollo de la forma de interacción y comunicación entre robots de diferente tecnología. Se ilustra como la información puede intercambiarse entre robots a través de una plataforma de comunicación y de esta forma aprovechar los datos para realizar tareas.

Durante el desarrollo de la tesis se hizo uso de ROS para la configuración y ensamble de un robot móvil, con el objetivo de presentar esta plataforma de desarrollo que ha venido ganando popularidad debido a sus capacidades. Además de que ROS permite la configuración y control del robot, su arquitectura basada en nodos hace natural la comunicación interna y externa al robot, es por esto que su fusión con modelos basados en la nube (Internet) potencializará la construcción de robots para todo uso.

Como resultado de este trabajo se presenta la configuración e integración de ROS en un robot móvil de armado propio. Dicho robot se construyó con base en los elementos que contaba la universidad (Raspberry Pi 3 y Arduino), y con la consigna de ser un robot de bajo costo. Se generaron varias versiones del robot en las cuales se agregaron mejoras de *hardware* y *software* necesarias para lograr el funcionamiento de ROS y de locomoción del robot.

Por otro lado, se adquirió un robot comercial de experimentación, con capacidades de percibir su entorno a través de sus actuadores, sensores y tarjetas de control. Este robot al ser de experimentación se debe configurar tanto el *hardware* y *software* para su correcto uso, donde ROS vuelve a ser la plataforma de funcionamiento para manejar y comunicar de manera eficiente los sensores con los que cuenta dicho robot. Resulta interesante la coincidencia de la configuración del robot de armado propio y el robot comercial, es decir, ambos robots cuentan con tarjetas de control Raspberry para el manejo de alto nivel y cuentan

con tarjetas para control de bajo nivel, en el caso del robot propio un Arduino Mega y en el caso de Turtlebot 3 una OpenCR.

La arquitectura de experimentación consiste en un robot de armado propio con un sensor ultrasónico para evitar colisiones y evasión de obstáculos, y otro robot de experimentación, con sensores de alta capacidad. Se generaron experimentos de modelado del entorno a partir de los datos generados con el apoyo de ROS. Se crearon programas que permitieron visualizar el entorno a través de la interpretación de los datos.

La información del ambiente y el algoritmo DWA permiten que de forma dinámica se cree una trayectoria que pueda ser seguida por ambos robots, presentando los resultados obtenidos.

En ambos escenarios se plantearon puntos de inicio y puntos meta para el desplazamiento de los robots. En el caso del Turtlebot 3, el seguimiento de la trayectoria se cumplió de manera eficiente, logrando llegar a la meta sin dificultades. Para el robot de armado propio, el seguimiento de la ruta se dio de manera correcta, alcanzando la meta a través de los obstáculos, sin embargo, el movimiento no fue exactamente el mismo en ambos robots, debido a diversos factores como son: distinto tipo de llantas, diferente estructura, tipo de superficie (loseta) o peso de los robots.

Aún con las diferencias antes mencionadas, la interacción entre ambos se logró de manera correcta, alcanzando la meta en ambos casos sin dificultades y evadiendo los obstáculos planteados en los escenarios.

En resumen, se obtuvieron los siguientes resultados:

- Conocimiento del desarrollo de robots usando el entorno de ROS.
- Integración de un robot móvil de armado propio.
- Puesta en marcha de un robot y análisis de la información de un sensor Lidar para modelado del entorno.
- Configuración de la comunicación entre robots con el paradigma de que el intercambio de información potencializa sus capacidades.

Dada la plataforma de experimentación y los resultados obtenidos se concluye que con el uso de plataformas para el desarrollo de robots (como lo es ROS), permitirá que la robótica este más al alcance de la comunidad técnica para brindar soluciones a problemas, permitiendo una rápida transferencia de los nuevos desarrollos.

5.1. Trabajo futuro

Con lo realizado hasta ahora se ha podido experimentar con las capacidades de ROS, que hacen posible la construcción de robots con grandes capacidades por medio de una plataforma de desarrollo común, queda como trabajo futuro las siguientes actividades:

1. Experimentar la transferencia de la trayectoria al robot de armado propio, es decir, descargar la información de la trayectoria para poder consultarla en cualquier momento.
2. En este modelo de experimentación, la comunicación entre robots es a través de una estación de trabajo local. Se necesita tener acceso a un esquema basado en la nube, por ejemplo, *RoboEarth*.
3. Evaluar las capacidades para que la información de un robot que se almacena y procesa en la nube pueda ser aprovechada por otros robots, permitiendo la transmisión de conocimiento entre robots. Lo anterior permitiría la aplicación masiva de robots en muchas actividades.

Si robótica en la nube funcionará de forma similar a las redes sociales para las personas, dotaría a los robots de nuevas habilidades para desempeñar tareas para las que no fueron hechos originalmente, pero que con la ayuda de otros robots más equipados de sensores pueden llegar a realizar.

Anexo A. Material utilizado en la construcción del robot propio

El material para la construcción del robot propio fue:

- 1 Raspberry Pi 3 Model B
- 1 Arduino Uno
- 1 Chasis de acrílico de dos niveles para robot con tornillería y separadores.
- 1 Cable USB tipo AB de 1.8 metros.
- 2 Motorreductor Recto 48:16 con rueda de plástico.
- 2 Switch Mini Deslizable 2 Pasos 3 Pin.
- 8 Diodo rectificador IN4005.
- Cable calibre 22 rojo, amarillo y negro.
- 2 Circuito Integrado L293D.
- Resistencia de 10 K Ohm 1/4W 5%
- 6 Micro botón push switch interruptor 2 pines.
- 1 Protoboard de 830 puntos.
- Cinchos de plástico.
- Cable micro USB a USB.
- Batería recargable marca Klip Xtreme, modelo KBH-120 de 5.3V 1000mAh.
- Batería recargable marca CDP, modelo R-PB10k de 5V 10000 mAh.
- 1 Paquete de 80 cables tipo dupont macho-macho y macho-hembra de 15 cm.
- 1 Batería recargable 12V 1.4 Ah Ácido-Plomo.
- 1 Mini protoboard de 170 puntos.
- 1 Sensor ultrasónico HC-SR04.
- 1 Carcasa de acrílico para Raspberry PI 3.
- 1 Rueda loca tipo “carrito de súper mercado”.
- 1 Batería recargable LiPo 11.1v 1200mAh.
- 1 Cable conector macho tipo “T” para batería LiPo.
- 1 Arduino Mega.

- 1 Carcasa de acrílico para Arduino Mega.
- 1 Rueda loca tipo “bola” de metal.
- 1 regulador de voltaje corriente continua 12-5V 5A.

Anexo B. Código desarrollado

B.1 *Sketch* de Arduino para tele operación y visualización de estados de sensores y actuadores en el robot (primer *sketch*).

```
#include "funciones.h" //Llamado del archivo con la implementación de funciones

//Robot Operating System
  #include <ros.h>
  #include <std_msgs/String.h>
  #include <std_msgs/Char.h>

  ros::NodeHandle  nh;
  std_msgs::String str_msg;
  ros::Publisher chatter("chatter", &str_msg);

// Designación de los puertos físicos para los botones de control
  //Motor 1
  #define Avance1 12
  #define Retroceso1 13
  #define Paro1 8

  //Motor 2
  #define Avance2 4
  #define Retroceso2 2
  #define Paro2 7

//Designación de los puertos físicos para señal PWM
  //Motor 1
  #define PWMA1 10
  #define PWMB1 11

  //Motor 2
  #define PWMA2 5
  #define PWMB2 6

//Sensor Ultrasónico
  #define PinTrig 9
  #define PinEcho 3

//Variables de velocidad
  byte vel1; //M1
  byte vel2; //M2

//Variables para detectar el motor para el cambio de velocidad
  byte velm1; //M1
  byte velm2; //M2
//Designación de las variables para detectar el flanco de subida en botones
  //Motor 1
  byte DisAvan1;
  byte DisRetr1;
  byte DisParo1;
  //Motor 2
  byte DisAvan2;
  byte DisRetr2;
  byte DisParo2;

//Variables para enclavamiento de avance y retroceso
  //Motor 1
```

```
  byte HoldAvan1;
  byte HoldRetr1;
  int pasado1[3]; //Arreglo de pasados para Motor 1
  //Motor 2
  byte HoldAvan2;
  byte HoldRetr2;
  int pasado2[3]; //Arreglo de pasados para Motor 2

//Variables para el control automático del motor
  //Motor 1
  byte AvanAuto1;
  byte RetrAuto1;
  byte ParoAuto1;
  //Motor 2
  byte AvanAuto2;
  byte RetrAuto2;
  byte ParoAuto2;

  //Alto para Motor1 y Motor2
  byte ParoAuto3;

//Variable para Sensor Ultrasónico
  int distancia1;

  byte dato; //Variable para leer puerto serial
  int pas[9]; //Arreglo de pasados para Detector de flanco del puerto serial

  unsigned long ini; //Variables para medir el tiempo
  int periodo = 1000;
  unsigned long tiempo;

//Variables para monitoreo del estado del Motor 1
  //Estado de la rueda Motor 1
  byte edoMot1 = 3;
  byte actualM1;
  String moniM1[3] = {"Avanzando", "Reversa", "Detenido"};
  //Velocidad Motor 1
  byte actualVM1;

//Variables para monitoreo del estado del Motor 2
  byte edoMot2 = 3;
  byte actualM2;
  String moniM2[3] = {"Avanzando", "Reversa", "Detenido"};
  //Velocidad Motor 2
  byte actualVM2;

//Distancia
  int actualDist1;

//Vector de estados
  char vEdos[100];

//Robot Operating System
  void messageCb( const std_msgs::Char& caracter_msg){
    dato = caracter_msg.data;
  }
  ros::Subscriber<std_msgs::Char> sub("comando", &messageCb);

void setup() {

//Robot Opertaing System
  nh.initNode();
  nh.subscribe(sub);
```

```
  nh.advertise(chatter);

//Configuración como salidas PWM
  //Motor 1
  pinMode(PWMA1, OUTPUT);
  pinMode(PWMB1, OUTPUT);
  //Motor2
  pinMode(PWMA2, OUTPUT);
  pinMode(PWMB2, OUTPUT);

//Configuración de botones de control
  //Motor 1
  pinMode(Avance1, INPUT);
  pinMode(Retroceso1, INPUT);
  pinMode(Paro1, INPUT);
  //Motor2
  pinMode(Avance2, INPUT);
  pinMode(Retroceso2, INPUT);
  pinMode(Paro2, INPUT);

  //Sensor Ultrasónico
  pinMode(PinTrig, OUTPUT);
  pinMode(PinEcho, INPUT);
}

void loop() {

switch(dato) {  //Switch para determinar las órdenes para los motores
  case '1':     // '1' corresponde a 49 en código ASCII Adelante M1
    AvanAuto1 = 1;
    edoMot1 = 1;
    dato = 0;
  break;
  case '2':     //Atras M1
    RetrAuto1 = 1;
    edoMot1 = 2;
    dato = 0;
  break;
  case '3':     //Paro M1
    ParoAuto1 = 1;
    edoMot1 = 3;
    dato = 0;
  break;
  case '4':     //Adelante M2
    AvanAuto2 = 1;
    edoMot2 = 1;
    dato = 0;
  break;
  case '5':     //Atras M2
    RetrAuto2 = 1;
    edoMot2 = 2;
    dato = 0;
  break;
  case '6':     //Paro M2
    ParoAuto2 = 1;
    edoMot2 = 3;
    dato = 0;
  break;
  case 'z':
    ParoAuto3 = 1;  //Paro de ambos motores simultáneamente
    edoMot1 = 3;
    edoMot2 = 3;
    dato = 0;
```

```
     break;
     case 'A':  //Baraja de velocidades para M1 desde A=0 hasta K=250 con
intervalos de 25 MAYÚSCULAS
     case 'B':
     case 'C':
     case 'D':
     case 'E':
     case 'F':
     case 'G':
     case 'H':
     case 'I':
     case 'J':
     case 'K':
       velm1 = 1;
     break;
     case 'a':  //Baraja de velocidades para M2 desde a=0 hasta k=250 con
intervalos de 25 minúsculas
     case 'b':
     case 'c':
     case 'd':
     case 'e':
     case 'f':
     case 'g':
     case 'h':
     case 'i':
     case 'j':
     case 'k':
       velm2 = 1;
     break;
     default:
       ;
     break;
   }

   //Enclavamiento Motor 1
     AvanAuto1 = DetectorFlanco(AvanAuto1, pas[0]);
     pas[0] = AvanAuto1;
     RetrAuto1 = DetectorFlanco(RetrAuto1, pas[1]);
     pas[1] = RetrAuto1;
     ParoAuto1 = DetectorFlanco(ParoAuto1, pas[2]);
     pas[2] = ParoAuto1;

   //Enclavamiento Motor 2
     AvanAuto2 = DetectorFlanco(AvanAuto2, pas[3]);
     pas[3] = AvanAuto2;
     RetrAuto2 = DetectorFlanco(RetrAuto2, pas[4]);
     pas[4] = RetrAuto2;;
     ParoAuto2 = DetectorFlanco(ParoAuto2, pas[5]);
     pas[5] = ParoAuto2;

   //Enclavamiento de paro para ambos motores simultáneamente
     ParoAuto3 = DetectorFlanco(ParoAuto3, pas[6]);
     pas[6] = ParoAuto3;

   //Enclavamiento de velocidad en Motor 1
     velm1 = DetectorFlanco(velm1, pas[7]);
     pas[7] = velm1;

   //Enclavamiento de velocidad en Motor 2
     velm2 = DetectorFlanco(velm2, pas[8]);
     pas[8] = velm2;

     if (velm1 == 1){
```

```
      vel1 = Velocidadm(dato); //Si "velm1" toma el valor de 1 entonces se invoca
la función "Velociadadm" y se le pasa el valor de "dato" desde el puerto serial
    }

    if (velm2 == 1){
      vel2 = Velocidadm(dato); //Si "velm2" toma el valor de 1 entonces se invoca
la función "Velociadadm" y se le pasa el valor de "dato" desde el puerto serial
    }

  //Generadores de pulso de señal motor 1
    DisAvan1 = DetectorFlanco(digitalRead(Avance1),pasado1[0]);
    pasado1[0] = digitalRead(Avance1);
    DisRetr1 = DetectorFlanco(digitalRead(Retroceso1),pasado1[1]);
    pasado1[1] = digitalRead(Retroceso1);
    DisParo1 = DetectorFlanco(digitalRead(Paro1),pasado1[2]);
    pasado1[2] = digitalRead(Paro1);

  //Retención de pulso avance. Enclavamiento
    HoldAvan1 = (AvanAuto1 || DisAvan1 || HoldAvan1) && (!DisParo1 && !ParoAuto1
&& !ParoAuto3);
    if (HoldAvan1 == 1){
      AvanceM(PWMA1, PWMB1, vel1);
      HoldRetr1 = 0;
      edoMot1 = 1;
    }

  //Retención de pulso retroceso. Enclavamiento
    HoldRetr1 = (RetrAuto1 || DisRetr1 || HoldRetr1) && (!DisParo1 && !ParoAuto1
&& !ParoAuto3);
    if (HoldRetr1 == 1){
      RetrocesoM(PWMA1, PWMB1, vel1);
      HoldAvan1 = 0;
      edoMot1 = 2;
    }

  //Retención de pulso paro. Enclavamiento
    if ((HoldAvan1 == 0) && (HoldRetr1 == 0)){
        ParoM(PWMA1, PWMB1);
        edoMot1 = 3;
    }

  //Generadores de pulso de señal motor 2
    DisAvan2 = DetectorFlanco(digitalRead(Avance2),pasado2[0]);
    pasado2[0]= digitalRead(Avance2);
    DisRetr2 = DetectorFlanco(digitalRead(Retroceso2),pasado2[1]);
    pasado2[1]= digitalRead(Retroceso2);
    DisParo2 = DetectorFlanco(digitalRead(Paro2),pasado2[2]);
    pasado2[2] = digitalRead(Paro2);

  //Retención de pulso avance. Enclavamiento
    HoldAvan2 = (AvanAuto2 || DisAvan2 || HoldAvan2) && (!DisParo2 && !ParoAuto2
&& !ParoAuto3); //Condiciones de arranque && Condiciones de paro
    if (HoldAvan2 == 1){
      AvanceM(PWMA2, PWMB2, vel2);
      HoldRetr2 = 0;
      edoMot2 = 1;
    }

  //Retención de pulso retroceso. Enclavamiento
    HoldRetr2 = (RetrAuto2 || DisRetr2 || HoldRetr2) && (!DisParo2 && !ParoAuto2
&& !ParoAuto3);
    if (HoldRetr2 == 1){
      RetrocesoM(PWMA2, PWMB2, vel2);
```

```
        HoldAvan2 = 0;
        edoMot2 = 2;
      }

   //Retención de pulso paro. Enclavamiento
      if ((HoldAvan2 == 0) && (HoldRetr2 == 0)){
        ParoM(PWMA2, PWMB2);
        edoMot2 = 3;
      }

   //Sensor Ultrasónico
      distancia1 = SensorUS(PinTrig,PinEcho);

   //Valores del robot para Monitor Serie
      if (millis() > ini + periodo){

         actualM1 = edoMot1;
         actualM2 = edoMot2;
         actualVM1 = vel1;
         actualVM2 = vel2;
         actualDist1 = distancia1;
         tiempo = millis() / 1000;

        sprintf(vEdos, "M1: %d, M2: %d, V1: %d, V2: %d, D1: %d, T: %d", actualM1,
actualM2, actualVM1, actualVM2, actualDist1, tiempo);

   //Robot Opertaing System
        str_msg.data = vEdos;
        chatter.publish( &str_msg );
        nh.spinOnce();

        ini = millis();
      }
   }

   FUNCIONES.H

   //Prototipo de funciones y su implementación
   void AvanceM(const int,const int, int);
   void RetrocesoM(const int,const int, int);
   void ParoM(const int,const int);
   int DetectorFlanco(int, int);
   void DetectorFlancoSerial(int*, int);
   int Velocidadm (int);
   int SensorUS (int, int);

   //Función de Avance
     void AvanceM(const int PWMA, const int PWMB, int vel){
       digitalWrite(PWMA, LOW);
       analogWrite(PWMB, vel);
     }
   //Función de Retroceso
     void RetrocesoM(const int PWMA, const int PWMB, int vel){
       digitalWrite(PWMB, LOW);
       analogWrite(PWMA, vel);
     }
   //Función de Paro
     void ParoM(const int PWMA, const int PWMB){
       digitalWrite(PWMA, LOW);
       digitalWrite(PWMB, LOW);
     }
   //Función de Enclavamiento de botonera
     int DetectorFlanco(int EstadoPuerto, int pasado){
```

```
      return EstadoPuerto && !pasado;
    }
  //Función de Enclavamiento de puerto serial
    void DetectorFlancoSerial(int *Activa, int pasado){
      *Activa = 1;
      if(pasado == 1 && *Activa == 1){
        *Activa = 0;
      }
    }
  //Función de Velocidad
    int Velocidadm(int dato){
      if (dato >= 65 && dato <= 75) // A partir de ((y2 - y1) / (x2 - x1)) *
(dato - x1) + y1;
        return ((250 - 0) / (75 - 65)) * (dato - 65) + 0;
      else if (dato >= 97 && dato <= 107)
        return ((250 - 0) / (107 - 97)) * (dato - 97) + 0;
      else
        return 0;
    }
  //Función de Sensor Ultrasónico
    int SensorUS (int PTrig, int PEcho){
    long distancia=0;
    long NumDatos=20;
    int i;
      for(i=0;i<NumDatos;i++){
        digitalWrite(PTrig, LOW);
        delayMicroseconds(2);
        digitalWrite(PTrig, HIGH);
        delayMicroseconds(10);
        digitalWrite(PTrig, LOW);
          distancia = pulseIn(PEcho, HIGH)*0.017+distancia;
      }
        distancia=distancia/20;
        if (distancia > 200)
          return -1;
        else
          return distancia;
    }
```

B.2 *Sketch* de Arduino para seguimiento de trayectorias (Segundo *sketch*).

```
#include <ros.h>
#include <math.h>
#include <geometry_msgs/Twist.h>
ros::NodeHandle nh;

float linx;
float angz;
int num = 10;
int lx;
int az;
int abslx;
int absaz;
//Designación de los puertos fisicos para señal PWM
  //Motor 1
  #define PWMA1 10
  #define PWMB1 11
  //Motor 2
  #define PWMA2 5
  #define PWMB2 6

void velCb(const geometry_msgs::Twist& vel){
  linx = vel.lineal.x;
  angz = vel.angular.z;
}

ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel", &velCb);

void setup(){
  nh.initNode();
  nh.subscribe(sub);
  //Configuracion como salidas PWM
  //Motor 1 Izquierdo
  pinMode(PWMA1, OUTPUT);
  pinMode(PWMB1, OUTPUT);
  //Motor2 Derecho
  pinMode(PWMA2, OUTPUT);
  pinMode(PWMB2, OUTPUT);
}

void loop(){                         //70 para lineal & angular
  abslx=abs(linx);                  //60 para trayectoria
  lx=(15/15.0)*((abslx*num)-0)+60; //((y2 - y1) / (x2 - x1)) * (dato - x1) +
y1;
  lx=ceil(lx); //floor

  absaz=abs(angz);
  az=(10/4.0)*((absaz*num)-0)+100; //((y2 - y1) / (x2 - x1)) * (dato - x1) +
y1;
  az=ceil(az); //floor

  if ((linx > 0) && (angz == 0)){ //Ambos motores ADELANTE
  digitalWrite(PWMA1, LOW);
  analogWrite (PWMB1, lx);
  digitalWrite(PWMA2, LOW);
  analogWrite(PWMB2, lx);
}
else if ((linx < 0) && (angz == 0)){ //Ambos motores ATRAS
  analogWrite(PWMA1, lx);
  digitalWrite(PWMB1, LOW);
  analogWrite(PWMA2, lx);
```

```
  digitalWrite(PWMB2, LOW);
}
else if ((linx == 0) && (angz > 0)){ //Giro a la DERECHA
  digitalWrite(PWMA1, LOW);
  analogWrite(PWMB1, az);
  analogWrite(PWMA2, az); //az
  digitalWrite(PWMB2, LOW);
}
else if ((linx == 0) && (angz < 0)){ //Giro a la IZQUIERDA
  analogWrite(PWMA1, az);  //az
  digitalWrite(PWMB1, LOW);
  digitalWrite(PWMA2, LOW);
  analogWrite(PWMB2, az);
}
else if ((linx > 0) && (angz > 0)){ //Giro a la DERECHA hacia ENFRENTE
  digitalWrite(PWMA1, LOW);
  analogWrite(PWMB1, az);
  digitalWrite(PWMA2, LOW);
  analogWrite(PWMB2, lx);
}
else if ((linx > 0) && (angz < 0)){ //Giro a la IZQUIERDA hacia ENFRENTE
  digitalWrite(PWMA1, LOW);
  analogWrite(PWMB1, lx);
  digitalWrite(PWMA2, LOW);
  analogWrite(PWMB2, az);
}
else if ((linx < 0) && (angz > 0)){ //Giro a la DERECHA hacia ATRAS
  analogWrite(PWMA1, lx);
  digitalWrite(PWMB1, LOW);
  analogWrite(PWMA2, az);
  digitalWrite(PWMB2, LOW);
}
else if ((linx < 0) && (angz < 0)){ //Giro a la IZQUIERDA hacia ATRAS
  analogWrite(PWMA1, az);
  digitalWrite(PWMB1, LOW);
  analogWrite(PWMA2, lx);
  digitalWrite(PWMB2, LOW);
}
else if ((linx == 0) && (angz == 0)){ //Paro
  digitalWrite(PWMA1, LOW);
  digitalWrite(PWMB1, LOW);
  digitalWrite(PWMA2, LOW);
  digitalWrite(PWMB2, LOW);
}
  nh.spinOnce();
  delay(1);
}
```

B.3 Código de Matlab para generar mapas con movimiento angular.

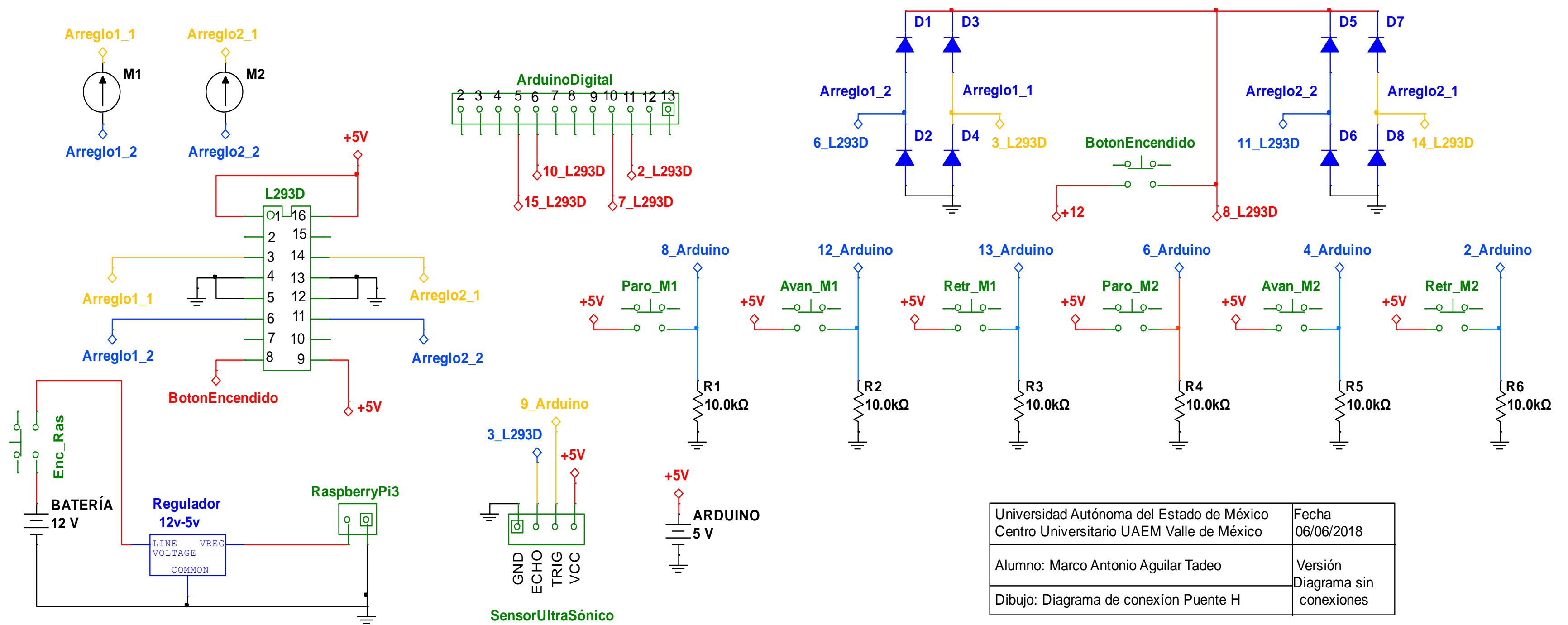
```
%Programa para seleccionar los datos del lidar
clear all
close all
DATA=xlsread('lecturas.xlsx');
DATA2=xlsread('Velocidad.xlsx');
[nl,nc]=size(DATA); %nl=111 y nc=731
fdata=DATA(:, 12:371); %se seleccionan únicamente los valores de rango

conteo2=[];
for i=1:nl; %número de muestras por tiempo 102
    timeLidar=DATA(:,1);
    timeVel=DATA2(:,1);
    compara=timeVel - timeLidar(i,1);
    pasos=sign(compara);
    conteo=find(pasos<0);
    conteo2=[conteo2 size(conteo,1)];
end

for i=1:(nl-1) %102-1=101
    pasos2(i)=conteo2(1,i+1)-conteo2(1,i);
end

pasos2=[0 pasos2]; %para recorrer el número de saltos
b=-4.5; %-4.5 grados
giro=0;
for j=1:nl %por escaneo 102 // es de 1:nl
    if pasos2(1,j)>0 %Se comprueba si hay movimiento de las ruedas
        giro=giro+b; %Se genera el valor para compensar el giro
        for i=1:359 %por revolucion de lidar 359
            x(i)=fdata(j,i) * cos((i+giro) * pi/180);
            y(i)=fdata(j,i) * sin((i+giro) * pi/180);

        end
        figure(1);
        title('GIRO DERECHA')
        plot(x,y,'.k')
        hold on
        axis('equal')
        %axis([-4.5 4.5 -3.5 3.5])
        pause(0.2);
        drawnow
        %hold off
    end
end
```

B.4 Código de Matlab para generar mapas con movimiento lineal.

```
%Programa para seleccionar los datos del lidar
clear all
close all
DATA=xlsread('lecturas.xlsx');
DATA2=xlsread('Velocidad.xlsx');
[nl,nc]=size(DATA); %nl=79 y nc=731
fdata=DATA(:, 12:371); %se seleccionan únicamente los valores de rango

conteo2=[];
for i=1:nl; %número de muestras por tiempo 79
    timeLidar=DATA(:,1);
    timeVel=DATA2(:,1);
    compara=timeVel - timeLidar(i,1);
    pasos=sign(compara);
    conteo=find(pasos<0);
    conteo2=[conteo2 size(conteo,1)];
end

for i=1:(nl-1) %79-1=78
    pasos2(i)=conteo2(1,i+1)-conteo2(1,i);
end

pasos2=[0 pasos2]; %para recorrer el número de saltos
vx=0.01; %positivo porque el robot avanza
a=pasos2(1:1); %Inicializa a en el primer valor del vector pasos2
for j=1:nl %por escaneo 82
    a=sum(pasos2(1:j)); %tiempo corrido de desplazamiento
    for i=1:359 %por revolución de lidar 359
        x(i)=fdata(j,i) * cos((i)* pi/180)+(a*vx);
        y(i)=fdata(j,i) * sin((i) * pi/180);
    end

    %grafica de los datos
    figure(1);
    title('AVANCE')
    plot(x,y,'.k')
    hold on
    axis('equal')
    %axis([-3 2.5 -1.5 2.5])
    pause(0.2);
    drawnow
    %hold off
end
```

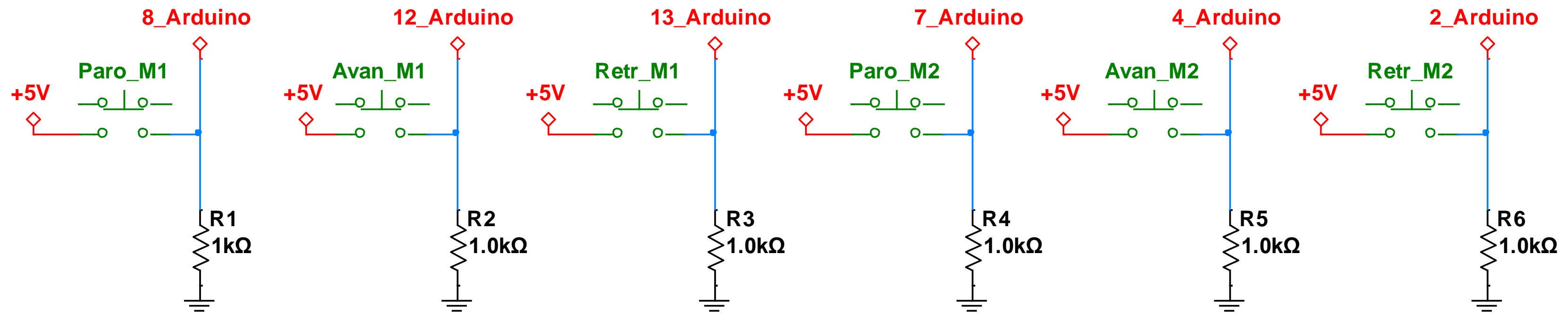
Anexo C. Diagramas de conexión

C.1 Diagrama de conexión general



Universidad Autónoma del Estado de México Centro Universitario UAEM Valle de México	Fecha 06/06/2018
Alumno: Marco Antonio Aguilar Tadeo	Versión Diagrama sin conexiones
Dibujo: Diagrama de conexíon Puente H	

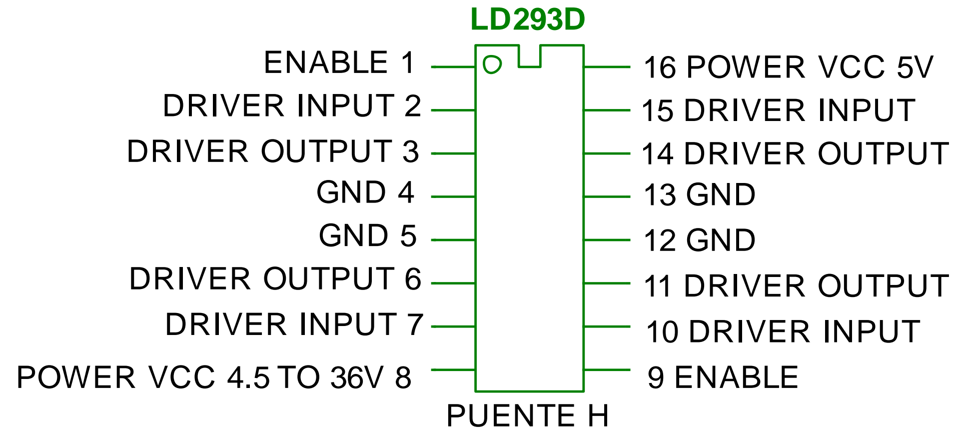
C.2 Diagrama de conexión de la botonera



Universidad Autónoma del Estado de México Centro Universitario UAEM Valle de México	Fecha 05/03/2018
Alumno: Marco Antonio Aguilar Tadeo	Versión Diagrama de conexiones
Dibujo: Diagrama de conexíon de la botonera	

C.3 Diagrama de conexión del puente H L293D

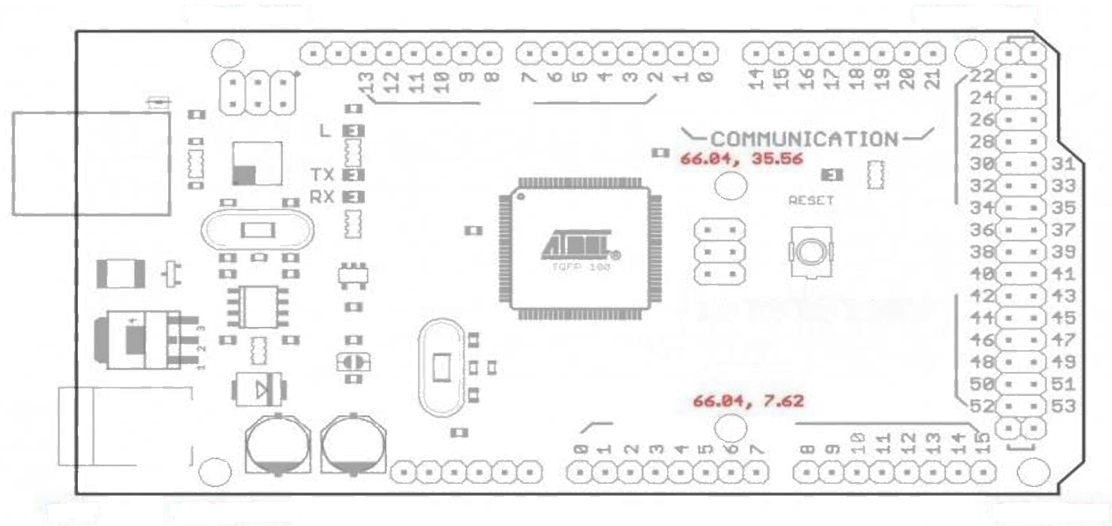
No. PIN	CONEXIÓN A
1	+5v
2	11 Arduino
3	Motor 1
4	GND
5	GND
6	Motor 1
7	10 Arduino
8	+12v
9	+5v
10	6 Arduino
11	Motor 2
12	GND
13	GND
14	Motor2
15	5 Arduino
16	+5v



Universidad Autónoma del Estado de México Centro Universitario UAEM Valle de México	Fecha 05/03/2018
Alumno: Marco Antonio Aguilar Tadeo	Versión Diagrama de conexiones
Dibujo: Diagrama de conexíon CI L293D	

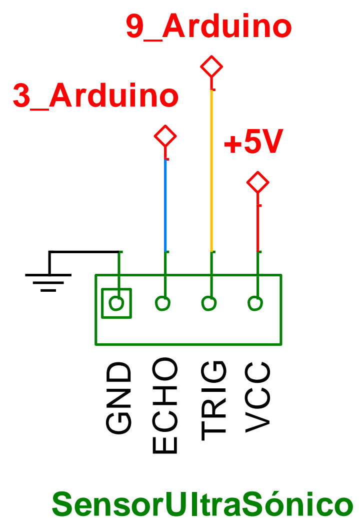
C.4 Diagrama de conexión de Arduino Mega

DIGITAL PWMK	Conexión a
D2	-Retr_M2
D3	Echo SenUltraS
D4	-Avan_M2
D5	Pin 15 L293D
D6	Pin 10 L293D
D7	-Paro_M2
D8	-Paro_M1
D9	Trigger SenUltraS
D10	Pin 7 L293D
D11	Pin 2 L293D
D12	-Avan_M1
D13	-Retr_M1



Universidad Autónoma del Estado de México Centro Universitario UAEM Valle de México	Fecha 05/03/2018
Alumno: Marco Antonio Aguilar Tadeo	Versión Diagrama de conexiones
Dibujo: Conexíones de Arduino Mega	

C.5 Diagrama de conexión del sensor ultrasónico



Universidad Autónoma del Estado de México Centro Universitario UAEM Valle de México	Fecha 05/03/2018
Alumno: Marco Antonio Aguilar Tadeo	Versión Diagrama de conexiones
Dibujo: Diagrama de conexíon Sensor Ultrasónico	

Referencias

Alcalá Tomás, F., Celorio Aponte, A., & Montoya Álvarez, C. (2013). Plataforma de simulación reconfigurable basada en Microsoft Robotics Developer Studio. Recuperado el 24 de agosto de 2017, de https://eprints.ucm.es/22468/1/Plataforma_de_Simulaci%C3%B3n_reconfigurable_basada_en_MRDS.pdf

Alonso, D., Pastor, J. Á., Sánchez, P., Álvarez, B., & Vicente Chicote, C. (2012). Generación automática de software para sistemas de tiempo real: Un enfoque basado en componentes, modelos y frameworks. *Revista Iberoamericana de Automática e Informática industrial, 9*(2), 170-181.

Bravo Sánchez, F. Á., & Forero Guzmán, A. (2012). La robótica como un recurso para facilitar el aprendizaje y desarrollo de competencias generales. *Teoría de la Educación. Educación y Cultura en la Sociedad de la Información, 13*(2).

Britanica, E. (s.f.). *Encyclopaedia Britannica*. Recuperado el 15 de Septiembre de 2016, de https://global.britannica.com/technology/robot-technology

Brooks, A., Kaupp, T., Makarenko, A., & Moser, M. (s.f.). *Orca*. Recuperado el 8 de junio de 2017, de http://orca-robotics.sourceforge.net/index.html

Bruyninckx, H. (s.f.). *The Orocos Project*. Recuperado el 9 de junio de 2017, de http://www.orocos.org/content/history

Center, D. R. (s.f.). *ROCK the robot construction kit*. Recuperado el 9 de junio de 2017, de https://www.rock-robotics.org/index.html

da Silva Gillig, J. U. (27 de octubre de 2016). *miniBloq*. Recuperado el 8 de junio de 2017, de http://blog.minibloq.org/

ETA PRIME. (26 de abril de 2016). *Youtube*. Recuperado el 17 de enero de 2018, de https://www.youtube.com/watch?v=KVSFBNZn_lE

Fedor, C., & Simmons, R. (s.f.). *CARMEN Robot Navigation Toolkit*. Recuperado el 6 de junio de 2017, de http://carmen.sourceforge.net/

Fox, D., Thrun, S., & Burgard, W. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine, 4*(1), 23-33.

García Cazorla , A. (2013). ROS: Robot Operating System.

Gerkey, B., Vaughan, R., & Howard, A. (16 de febrero de 2014). *The player project*. Recuperado el 6 de junio de 2017, de http://playerstage.sourceforge.net/index.php?src=index

Goldberg, K. (1994). Beyond the web: Excavating the real world via mosaic. *Second International WWW Conference*, (págs. 1-12). Chicago, IL, USA.

Hu, G., Tay, W. P., & Wen, Y. (2012). Cloud robotics: architecture, challenges and applications. *IEEE network, 26*(3).

Kamei, K., Nishio, S., & Hagita, N. (2012). Cloud Networked Robotics. *IEEE Network*.

Koken, B. (2015). Cloud Robotics platforms. *Interdisciplinary Description of Complex Systems: INDECS, 13*(1), 26-33.

Kuffner, J. (2010). Cloud-enabled robots. *International Conference Humanoid Robot.* Nashville, TN, USA.

Kumar, V., Rus, D., & Sukhatme, G. S. (2008). Networked robots. En *Springer Handbook of Robotics* (págs. 943-958). Berlin: Springer.

Labrador Fleitas, A. (2014). Desarrollo de un sistema basado en ROS (Robotic Operating System) para tele operar un vehículo agrícola e integración de sensor filoguiado para navegación autónoma.

Makarenko Alexei, B. A. (2006). Orca: Components for robotics. *International Conference on Intelligent Robots and Systems (IROS).* Beijin.

Mell, P., & Grance, T. (September de 2011). The NIST definition of Cloud Computing. *National Institute of Standars and Tecnology, Special Publication*.

Microsoft. (2017). *Microsoft*. Recuperado el 7 de junio de 2017, de https://www.microsoft.com/en-us/download/details.aspx?id=29081

Mizerany, B. (s.f.). *Sinatra*. Recuperado el 5 de junio de 2017, de http://sinatrarb.com/

Mohanarajah, G. (29 de julio de 2014). *Youtube*. Obtenido de https://www.youtube.com/watch?v=mgPQevfTWP8

myrobotlab. (s.f.). *myrobotlab*. Recuperado el 8 de junio de 2017, de http://myrobotlab.org/

Newman, P. M. (2008). MOOS - Mission Orientated Operating Suite.

Newman, P. (s.f.). *MOOS*. Recuperado el 8 de junio de 2017, de http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/HomePage

OpenCV team. (2017). *OpenCV*. Recuperado el 8 de junio de 2017, de https://opencv.org/

Roboearth. (s.f.). *Roboearth*. Recuperado el 14 de Septiembre de 2016, de http://roboearth.ethz.ch/

Robohow. (s.f.). *Robohow*. Recuperado el 12 de Septiembre de 2016, de http://robohow.eu/

Robotis. (2018). *Robotis*. Recuperado el 17 de febrero de 2018, de http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

Rüf Stiftung, G. (2014). *EEROS*. Recuperado el 6 de junio de 2017, de http://eeros.org/wordpress/

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). Autonomous Mobile Robots. MIT press.

Tapia García, M. R., & López Hernández, J. M. (2017). Robótica Móvil. *Jóvenes en la ciencia, 3*(2), 2526-2530.

The Hybrid Group. (2014). *artoo Ruby on robots*. Recuperado el 05 de junio de 2017, de http://artoo.io/

Ubuntu MATE Team. (2018). *Ubuntu Mate*. Recuperado el 8 de octubre de 2017, de https://ubuntu-mate.org

Willow Garage. (14 de 01 de 2016). *ROS*. Recuperado el 28 de noviembre de 2016, de http://wiki.ros.org/turtlebot_gazebo/Tutorials/indigo/Gazebo%20Bringup%20Guide

Willow Garage. (24 de agosto de 2017). *ROS*. Recuperado el 16 de noviembre de 2016, de http://wiki.ros.org/indigo/Installation/Ubuntu

Willow Garage. (13 de septiembre de 2018). *ROS*. Recuperado el 28 de agosto de 2017, de http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi

Willow Garage. (s.f.). *ROS*. Recuperado el 12 de Septiembre de 2017, de http://www.ros.org/

Zheng, K. (2017). ROS Navigation Tuning Guide. *arXiv preprint arXiv:1706.09068*.