



UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO EN INGENIERÍA

EN MECATRÓNICA

TEMA:

**“PLATAFORMA DE ROBÓTICA MÓVIL PARA EXPERIMENTOS A TRAVÉS  
DEL SISTEMA OPERATIVO DE ROBOTS (ROS): DESARROLLO DE LA  
TARJETA DE TIEMPO REAL”**

(MOBILE ROBOTICS PLATFORM FOR EXPERIMENTS USING ROS:

DEVELOPMENT OF THE REAL – TIME BOARD)

**AUTOR:** EDWIN MAURICIO LASCANO BEJARANO

**DIRECTOR:** XAVIER ROSERO

IBARRA – ECUADOR

2017



## UNIVERSIDAD TÉCNICA DEL NORTE

### BIBLIOTECA UNIVERSITARIA

### AUTORIZACIÓN DE USO Y PUBLICACIÓN

#### IDENTIFICACIÓN DE LA OBRA

La Universidad Técnica del Norte dentro del Proyecto Repositorio Digital Institucional determinó la necesidad de disponer de textos completos en formato digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la universidad.

Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto, para lo cual se pone a disposición la siguiente información:

DATOS DEL AUTOR	
<b>CEDULA DE IDENTIDAD</b>	172248424-1
<b>APELLIDOS Y NOMBRES</b>	LASCANO BEJARANO EDWIN MAURICIO
<b>DIRECCIÓN</b>	CAYAMBE (TERAN E INDEPENDENCIA)
<b>E-MAIL</b>	<a href="mailto:edwinlb20@gmail.com">edwinlb20@gmail.com</a>
<b>TELÉFONO MÓVIL</b>	0981071809
DATOS DE LA OBRA	
<b>TÍTULO</b>	“PLATAFORMA DE ROBÓTICA MÓVIL PARA EXPERIMENTO A TRAVÉS DEL SISTEMA OPERATIVO DE ROBOTS (ROS): DESARROLLO DE LA TARJETA DE TIEMPO REAL” (MOBILE ROBOTICS PLATFORM FOR EXPERIMENTS USING ROS: DEVELOPMENT OF THE REAL – TIME BOARD)
<b>AUTOR</b>	EDWIN MAURICIO LASCANO BEJARANO
<b>FECHA</b>	2017
<b>PROGRAMA</b>	PREGRADO
<b>TÍTULO POR EL QUE OPTA</b>	INGENIERO EN MECATRÓNICA
<b>ASESOR</b>	CARLOS XAVIER ROSERO CHANDI



**UNIVERSIDAD TÉCNICA DEL NORTE**  
**FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS**  
**AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD**

Yo, Edwin Mauricio Lascano Bejarano, con cédula de identidad Nro. 172248424-1, en calidad de autor y titular de los derechos patrimoniales del trabajo de grado descrito anteriormente, hago entrega del ejemplar respectivo en formato digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y uso del archivo digital en la biblioteca de la Universidad con fines académicos, para ampliar la disponibilidad del material y como apoyo a la educación, investigación y extensión; en concordancia con la Ley de Educación Superior Artículo 144.

**CONSTANCIAS**

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrollo sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

A handwritten signature in blue ink, which appears to read "Edwin Lascano", is written over a horizontal dotted line.

Edwin Mauricio Lascano Bejarano

C.I: 172248424-1



**UNIVERSIDAD TÉCNICA DEL NORTE**  
**FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS**

**DECLARACIÓN**

Yo, Edwin Mauricio Lascano Bejarano, con cédula de identidad N°. 172248424-1, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; y que éste no ha sido previamente presentado para ningún grado o calificación profesional.

A través de la presente declaración cedo los derechos de propiedad intelectual correspondientes a este trabajo, a la Universidad Técnica del Norte, según lo establecido por las Leyes de la Propiedad Intelectual, Reglamentos y Normativa vigente de la Universidad Técnica del Norte.

Edwin Mauricio Lascano Bejarano

C.I: 172248424-1



**UNIVERSIDAD TÉCNICA DEL NORTE**

**FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS**

**CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE GRADO A FAVOR DE  
LA UNIVERSIDAD TÉCNICA DEL NORTE**

Yo, Edwin Mauricio Lascano Bejarano , con cédula de identidad Nro. 172248424-1, manifiesto mi voluntad de ceder a la Universidad Técnica del Norte los derechos patrimoniales consagrados en la Ley de Propiedad Intelectual del Ecuador, artículos 4, 5, 6, en calidad de autor del trabajo de grado denominado: “MOBILE ROBOTICS PLATFORM FOR EXPERIMENTS USING ROS: DEVELOPMENT OF THE REAL-TIME BOARD” , que ha sido desarrollado para optar por el título de Ingeniero en Mecatrónica en la Universidad Técnica del Norte, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente. En mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la biblioteca de la Universidad Técnica del Norte.

Edwin Mauricio Lascano Bejarano

C.I: 172248424-1



**UNIVERSIDAD TÉCNICA DEL NORTE**  
**FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS**

**CERTIFICO**

Que la tesis previa a la obtención del título de Ingeniero en Mecatrónica con el tema: "MOBILE ROBOTICS PLATFORM FOR EXPERIMENTS USING ROS: DEVELOPMENT OF THE REAL-TIME BOARD", ha sido desarrollado y terminado en su totalidad por el Sr. Edwin Mauricio Lascano Bejarano, con cédula de identidad 172248424-1, bajo mi supervisión para lo cual firmo en constancia.

Carlos Xavier Rosero

Chandi

DIRECTOR



**UNIVERSIDAD TÉCNICA DEL NORTE**  
**FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS**

**AGRADECIMIENTO**

*Agradezco a Dios por guiar mis pasos y no dejarme vencer ante las adversidades de la vida, ya que con la confianza y la fe puesta en él he logrado alcanzar mis objetivos.*

*A mi madre que me han brindado su apoyo, su cariño y sobre todo su ejemplo de sobresalir ante cualquier adversidad, por confiar y creer en mí, a mi hermana y a todos aquellos que estuvieron presentes durante la elaboración de este trabajo manifestándome su apoyo.*

*Un especial y sincero agradecimiento al Ing. Xavier Rosero, que con su experiencia y conocimiento ha orientado mi trabajo, con motivación ha hecho posible que esta labor sea cumplida, con lo que se ha ganado mi admiración.*



**UNIVERSIDAD TÉCNICA DEL NORTE**  
**FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS**

**DEDICATORIA**

*La elaboración de este trabajo está dedicado principalmente a Dios, ya que me ha dado la salud y la fortaleza para cumplir mis metas; a mis padres que son las personas que quiero y admiro, ya que han sido el apoyo incondicional en mi vida, los que me han apoyado incondicionalmente y que con su ejemplo y labor han velado por mi bienestar y educación, también lo dedico a mi hijo, una de las razones de mi superación el que me han dado ánimo y aliento para seguir adelante para que le sirva de ejemplo, iguale y supere mis logros.*

*Por eso con mucho amor y cariño dedico este trabajo de investigación a mi familia.*



## RESUMEN

El campo de la robótica comprende un sinnúmero de tecnologías y sistemas para su operación y control. En la actualidad se pueden encontrar sistemas robóticos con mecanismos cada vez más complejos, capaces de realizar tareas que antes eran inalcanzables por robots convencionales. La rama de la robótica que más ha evolucionado en los últimos años es la robótica móvil. Los robots móviles son máquinas capaces de trasladarse en cualquier ambiente sin fijarse a una sola ubicación física. Además, realizan movimientos complejos que son realizados en tiempo real, que ocurren con una planificación predecible y que deben ser realizados en un plazo de tiempo definido.[1]

En la Universidad Técnica del Norte no se han desarrollado aún herramientas de robótica móvil que permitan visualizar, comprender y aplicar las teorías y fundamentos de las asignaturas de robótica y sistemas de control. En base a los aspectos antes mencionados nace la idea de realizar este trabajo titulado “Plataforma de Robótica Móvil para experimentos a través del Sistema Operativo de Robots (ROS): Desarrollo de la Tarjeta de Tiempo Real” mismo que se realizó con fines educativos para aportar al área de Ciencias Aplicadas de la Universidad.

La conexión de sistemas integrados en tiempo real a ROS es una forma de aprovechar las capacidades de nivel superior de este meta-sistema operativo. Para la elaboración del presente proyecto se realizó la implementación del robot móvil a través de la plataforma robótica (ROS) que provee librerías, aplicaciones visuales de software y herramientas de comunicación. Éste fue ejecutado en un ordenador con aplicaciones de código abierto. Para la parte del sistema de tiempo real se empleó un microcontrolador y un sensor ultrasónico que están comunicados por nodos al ordenador.

El presente trabajo abarca los procesos de desarrollo, implementación y ejecución de un sistema de tiempo real en un robot móvil, su funcionalidad se centra en la utilización de hardware de fácil

accesibilidad y software de tecnología abierta. Se aplicaron conocimientos relacionados con control, robótica, y programación, adquiridos a lo largo de la carrera.

Como resultado se obtuvo un sistema funcional, flexible y capaz de incrementar los conocimientos de los estudiantes de la carrera de Ingeniería Mecatrónica además de dar a conocer las nuevas tecnologías que se están generando en el mundo.

Con la culminación de este proyecto se dejarán sentadas las bases para posteriores estudios en el área de robótica, con el fin de promover nuevos conocimientos académicos.

## ABSTRACT

The field of robotics comprises a number of technologies and systems for its operation and control. At present, robotic systems with increasingly complex mechanisms can be found capable of performing tasks previously unattainable by conventional robots. The robotics branch that has evolved the most in recent years is mobile robotics. Mobile robots are machines capable of moving in any environment without being fixed to a single physical location. In addition, they perform complex movements that are performed in real time, occur with predictable planning and must be performed within a defined time frame.

At the Universidad Técnica del Norte, no mobile robotics tools have yet been developed to visualize, understand and apply the theories and fundamentals of robotics subjects and control systems. Based on the aforementioned aspects, the idea was born to carry out this work entitled "Mobile Robotics Platform for Experiments through the Robots Operating System (ROS): Development of the Real Time Card", which was done for educational purposes to contribute to the area of Applied Sciences of the University.

Connecting real-time integrated systems to ROS is a way to take advantage of the higher-level capabilities of this meta-operating system. For the development of this project, the mobile robot was implemented through the ROS robotic platform, which provides libraries, visual software applications and communication tools. It was run on a computer with open source applications. For the part of the real time system a microcontroller and an ultrasonic sensor were used which are communicated by nodes to the computer.

This work covers the development, implementation and execution of a real-time system in a mobile robot, its functionality is focused on the use of easily accessible hardware and open

technology software. Knowledge related to control, robotics, and programming, acquired throughout the race, was applied.

As a result, a functional, flexible system was obtained, capable of increasing the knowledge of the students of the Mechatronics Engineering career as well as making known the new technologies that are being generated in the world.

With the culmination of this project will be laid the basis for further studies in the area of robotics, in order to promote new academic knowledge.



# ÍNDICE DE CONTENIDOS

	<b>Pág.</b>
IDENTIFICACIÓN DE LA OBRA .....	ii
AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD .....	iii
CONSTANCIAS .....	iii
DECLARACIÓN .....	iv
CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE GRADO A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE .....	v
CERTIFICO .....	vi
AGRADECIMIENTO .....	vii
DEDICATORIA .....	viii
RESUMEN .....	ix
ABSTRACT .....	xi
<b>CAPÍTULO I</b> .....	13
GENERALIDADES Y MOTIVACIÓN .....	13
INTRODUCCIÓN .....	13
1.1 PROBLEMÁTICA .....	15
1.2 JUSTIFICACIÓN .....	16
1.4 OBJETIVOS .....	17
1.4.1 OBJEIVO GENERAL .....	17
1.4.2 OBJETIVO ESPECÍFICO .....	17
1.5 ALCANCE .....	18
<b>CAPÍTULO II</b> .....	19
2. REVISIÓN LITERARIA .....	19
2.1 Tiempo Real .....	19
2.1.1 Definición de un Sistema en Tiempo Real .....	20
2.1.2 Clasificación de los Sistemas en Tiempo Real .....	21
2.1.3 Estructura General de un Sistema en Tiempo Real .....	22
2.1.4 Aplicaciones de Sistemas en Tiempo Real .....	23
2.2 Tiempo Real en los Robots .....	25
2.3 Robot Operating System (ROS) .....	27

2.3.1	Características Principales del Sistema Operativo de Robots (ROS)	27
2.3.2	Conceptos Básicos del Sistema Operativo de Robots (ROS)	28
2.4	Tiempo real en ROS	30
2.5	Problemática y la Necesidad del Sensor Ultrasónico	31
2.6	Módulo de Tiempo Real con el Sistema Operativo de Robots (ROS)	32
2.7	Diagrama General del Sistema	34
2.7.1	Sistema de Plataforma Robótica (Kobuki)	34
2.7.1.1	Nodo Actuador (motor)	37
2.7.1.2	Nodo Sensores.	38
2.7.1.2.1	Sensores de Impacto	39
2.7.1.2.2	Codificador de Cuadratura	39
2.7.2	Sistema de Control (Ordenador)	40
2.7.2.1	Roscore	41
2.7.2.2	Rosserial	41
2.7.3	Sistema de Tiempo Real	41
<b>CAPÍTULO III</b>		<b>43</b>
3.	Caracterización del Módulo de Tiempo	43
3.1	Diagrama de Bloques del Robot Kobuki ejecutando el Sistema en Tiempo Real	43
3.1.1.1	Mobile_Base_Nodelet_Manager (Administrador de nodo de base móvil)	44
3.1.1.2	Mobile_Base (base móvil)	44
3.1.1.3	Diagnostic Aggregator	44
3.1.1.4	Rosout	45
3.1.2.	Sistema en Tiempo Real	45
3.1.2.1	Nodo Serial	45
3.1.2.2	Instalación del Software	46
3.1.2.3	Instalación de la Librería ROS_lib en el Entorno de Arduino.	47
3.1.2.4	Ejemplo de su Uso	48
<b>CAPÍTULO IV</b>		<b>51</b>
4.	IMPLEMENTACIÓN	51
4.1	Programa del Microcontrolador (Arduino)	51
4.1.1	Programa de Medición de Distancia y Publicación de Datos	52
4.1.2	Código Explicado	54

4.1.3 Diagrama de flujo del programa ultrasonido .....	58
4.2 Pasos para la Implementación y Verificación .....	59
CAPÍTULO V .....	67
5. Análisis y Resultados.....	67
5.1 Prueba de conexión de los nodos en ejecución.....	67
5.1.1 Pruebas de conexión del nodo Kobuki .....	68
5.1.2 Pruebas de Conexión del Nodo Rosserial.....	69
5.2 Pruebas de Respuesta del Sensor Ultrasonico Hacia un Objeto (pared) .....	70
5.2.1 Tiempo de Muestreo en $t=1s$ y Distancia Física de 0.50m.....	70
5.2.2 Tiempo de Muestreo en $t=1s$ y Distancia Física de 0.70m.....	72
5.2.3 Tiempo de Muestreo en $t=2s$ y Distancia Física de 0.70m.....	73
5.2.4 Tiempo de Muestreo en $t=1s$ y Distancia Física de 0.94m.....	74
5.2.5 Tiempo de Muestreo en $t=1s$ y Distancia Física de 1.20m.....	75
CONCLUSIONES Y RECOMENDACIONES .....	78
CONCLUSIONES .....	78
RECOMENDACIONES.....	79
ANEXOS .....	80
BIBLIOGRAFÍA .....	83



## ÍNDICE DE FIGURAS

<b>Figura N°</b>	<b>Pág.</b>
Figura 1. Sistema de Tiempo Real.....	18
Figura 2. Diagrama de un Proceso en Tiempo Real[9] .....	22
Figura 3. Prioridad de Control por Ordenador [9].....	23
Figura 4. Aplicaciones y su Tiempo de Respuesta[11] .....	25
Figura 5. Robot y su Interacción con el Entorno [12] .....	26
Figura 6. Comunicación por Tópicos .....	30
Figura 7. Sensor Ultrasónico [16] .....	32
Figura 8. Microcontrolador Arduino [18].....	33
Figura 9. Esquema General del Sistema [19] .....	34
Figura 10. Plataforma Robótica Kobuki [19] .....	35
Figura 11. Ubicación de los Motores y Ruedas en el Robot Móvil [21] .....	38
Figura 12. Sensor de Impacto [19] .....	39
Figura 13. Codificador de Cuadratura[19].....	40
Figura 14. Sistema de Tiempo Real Implementado [19].....	43
Figura 15. Diagrama de Bloques de los Sistemas en Tiempo Real [19] .....	43
Figura 16. Instalación de la Librería ros-lib [28].....	48
Figura 17. Diagrama de Flujo del Programa Ultrasónico [19] .....	58
Figura 18. Visualización de Carga de Programa hacia el Arduino Uno [19] .....	59
Figura 19. Ejecución de Roscore [19] .....	60
Figura 20. Comunicación entre el Ordenador y Kobuki [19].....	61
Figura 21. Teleoperación del Robot Kobuki [19].....	62
Figura 22. Verificación de puertos activos [19] .....	63
Figura 23. Activación del Puerto Serial [19] .....	63
Figura 24. Lista de Temas a los que se Puede Suscribirse [19] .....	64
Figura 25. Datos Obtenidos del Sensor Ultrasónico [19].....	65
Figura 26. Preparado para ejecutar Rviz [19].....	65
Figura 27. Recepción de datos desde el sensor ultrasónico en tiempo real [19] ....	66
Figura 28. Diagrama de nodos en ejecución [19] .....	67

Figura 29. Interacción del Nodo Kobuki con los Demás Nodos [19] .....	69
Figura 30. Comunicación del Nodo Serial con los Demás Nodos [19].....	70
Figura 31. Medición de Datos a una Distancia de 0.50m [19] .....	70
Figura 32. Medición de Datos a una Distancia de 0.70m[19] .....	72
Figura 33. Medición de Datos a una Distancia de 0.94m [19] .....	74
Figura 34. Medición de Datos a una Distancia de 1.20m [19] .....	75

## ÍNDICE DE TABLAS

<b>Tabla N°</b>	<b>Pág.</b>
Tabla 1. Tipos de Mensajes de ROS.....	29
Tabla 2. Elementos de la Plataforma Robótica Kobuki [20] .....	36
Tabla 3. Características Principales del Actuador [22] .....	37
Tabla 4. Datos de Sensor Ultrasónico t=1s y d=0.50m [29].....	711
Tabla 5. Datos de Sensor Ultrasónico t=1s y d=0.70m [29].....	722
Tabla 6. Datos de Sensor Ultrasónico t=2s y d=0.70m [29].....	733
Tabla 7. Datos de Sensor Ultrasónico t=1s y d=0.94m [29].....	744
Tabla 8. Datos de Sensor Ultrasónico t=1s y d=1.20m [29].....	766



# CAPÍTULO I

## GENERALIDADES Y MOTIVACIÓN

### INTRODUCCIÓN

La robótica está experimentando un crecimiento explosivo propulsado por los avances en computación, sensores, electrónica y software[1]. Los robots están ya revolucionando los procedimientos que se emplean en la medicina, agricultura, minería y el transporte, al solventar la mayoría de sus necesidades de control y automatización. De esta manera, existen innumerables aplicaciones que día a día mejoran el diario vivir de la sociedad.

En la actualidad existen varios Framework de Desarrollo en Robótica tales como Player/Stage/Gazebo, Yarp, Orocos, OpenRave entre otros, todos ellos de código abierto, sin embargo, Sistema operativo de robots (ROS) ha logrado agrupar las mejores características de todos estos proyectos dando una solución integral y muy uniforme al problema de desarrollo de sistemas robóticos, se caracteriza por ser una plataforma multi-lenguaje (c++, python, java), peer2peer, orientado a herramientas, ligero y de código abierto (OpenSource)[2]. Un sistema construido utilizando (ROS) provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes[3].

Está basado en una arquitectura de punto a punto donde el procesamiento toma lugar en los nodos que pueden recibir, entregar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros.

La realización de este proyecto se fundamenta en una necesidad la cual consiste, que en la actualidad la Universidad Técnica del Norte y particularmente a la Facultad de Ingeniería en Ciencias Aplicadas, no existen robots móviles que se utilicen como equipo para enseñanza e investigación, situación que repercute en las asignaturas relacionadas con robótica y control, convirtiéndolas netamente teóricas.

En base a la problemática descrita, se ve imperiosa la implementación de un robot móvil para experimentos de robótica que sea utilizada como equipo didáctico y de investigación en los laboratorios de la FICA.

El presente tema de tesis está compuesto por cuatro capítulos a continuación un breve resumen de cada uno:

**Capítulo 1:** Da a conocer los factores que motivaron el desarrollo de este trabajo de tesis, la introducción, los antecedentes, los objetivos, y el alcance.

**Capítulo 2:** Detalla la información necesaria para la realización de este proyecto además los elementos que se emplearon para su buen funcionamiento.

**Capítulo 3:** En este capítulo se describen los bloques de los que está constituido la plataforma robótica móvil además el vínculo de éste con el nodo de tiempo real.

**Capítulo 4:** Se muestra la implementación del software y el hardware, necesario para el funcionamiento óptimo del sistema de tiempo real con la plataforma robótica Kobuki.

**Capítulo 5:** Describe las diferentes pruebas que se realizó al proyectos los resultado y análisis de cada uno, por último conclusiones, recomendaciones, con el fin de mejorar futuras investigaciones.

## 1.1 PROBLEMÁTICA

Con el transcurrir del tiempo, el control y la automatización pasaron a ser parte de muchos de los procesos relacionados con el hombre. Hoy en día muchas de las necesidades humanas han sido solventadas a través de aplicaciones robóticas complejas. Así, existen innumerables aplicaciones basadas en plataformas de desarrollo en robótica (RSF) que día a día mejoran el diario vivir de la sociedad con aplicaciones de diferente topología.

En la actualidad existen varios entornos de desarrollo para robótica tales como Player/Stage/Gazebo, Yarp, Orocos, OpenRave, entre otros, todos ellos de código abierto y de desarrollo dependiente. Sin embargo, hasta hace pocos años atrás, no existía una plataforma que logre agrupar las mejores características de los proyectos mencionados en una solución integral y uniforme.[4]

Con la aparición de Sistema Operativo de Robots (ROS) se ha logrado resolver el problema de desarrollo de sistemas robóticos, proporcionando una plataforma integral, multi-lenguaje, orientada a herramientas, ligera y de código abierto. Un sistema robótico que usa (ROS), posee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. (ROS) está basado en una arquitectura de punto a punto donde el procesamiento toma lugar en los nodos, pueden recibir, entregar y multiplexar mensajes de sensores, controladores, actuadores, así como mensajes de estado y planificación. [5]

En lo concerniente a la realidad de la Universidad Técnica del Norte y particularmente a la Facultad de Ingeniería en Ciencias Aplicadas, no existen robots móviles que se utilicen como

equipo para enseñanza e investigación, situación que repercute negativamente en las asignaturas relacionadas con robótica y control, convirtiéndolas en netamente teóricas.

Es así que se considera necesario el desarrollo de una plataforma robótica basada en (ROS) para ser usada como base para la realización de innumerables investigaciones orientadas al mejoramiento del propio robot. Luego de adquirir experticia en el tema, se puede usar la misma plataforma para el desarrollo de sistemas con propósitos industriales y/o de producción.

## **1.2 JUSTIFICACIÓN**

Desde siempre, el ser humano ha utilizado los recursos a su disposición como herramientas para ayudarlo en la realización de las tareas de la forma más efectiva, rápida y segura. A medida que la tecnología avanza, estas herramientas se convierten en maquinaria cada vez más compleja y capaz de realizar trabajos complicados de manera precisa y muchas veces, mejor de lo que podría haberlo hecho un humano.

Para realizar el proyecto se implementará el Sistema Operativo Robótico (ROS) por todas sus ventajas como: procesamiento de imágenes, transformación de coordenadas, navegación (Path Planning), control distribuido, bajo costo, escalabilidad, repetitividad, etc. Bajo este concepto, se necesita un autómatas que muestre una interfaz sencilla y fraterna con el operador y posea suficiente versatilidad para que, con ligeras modificaciones de hardware y firmware, consienta su empleo en cualquier tipo de aplicación de automoción. Se requiere de un equipo con índice costo – beneficio aceptable, vida útil larga, mantenimiento barato y repuestos accesibles en nuestro medio.



La relevancia del dispositivo propuesto consiste en que servirá como plataforma para el posterior diseño de sistemas robóticos de automoción, y como indicador para exponer el alto nivel de conocimientos y el gran potencial innovador de los estudiantes de la Universidad Técnica del Norte.

La implementación de este sistema se basa en nociones, habilidades, capacidades, destrezas y aptitudes, vinculadas a nuestra competencia profesional y conocimientos adquiridos.

## **1.4 OBJETIVOS**

### **1.4.1 OBJETIVO GENERAL**

- ✓ Desarrollar la placa de tiempo real para la detección de obstáculos de un robot móvil basado en ROS (Robot Operating System).

### **1.4.2 OBJETIVO ESPECÍFICO**

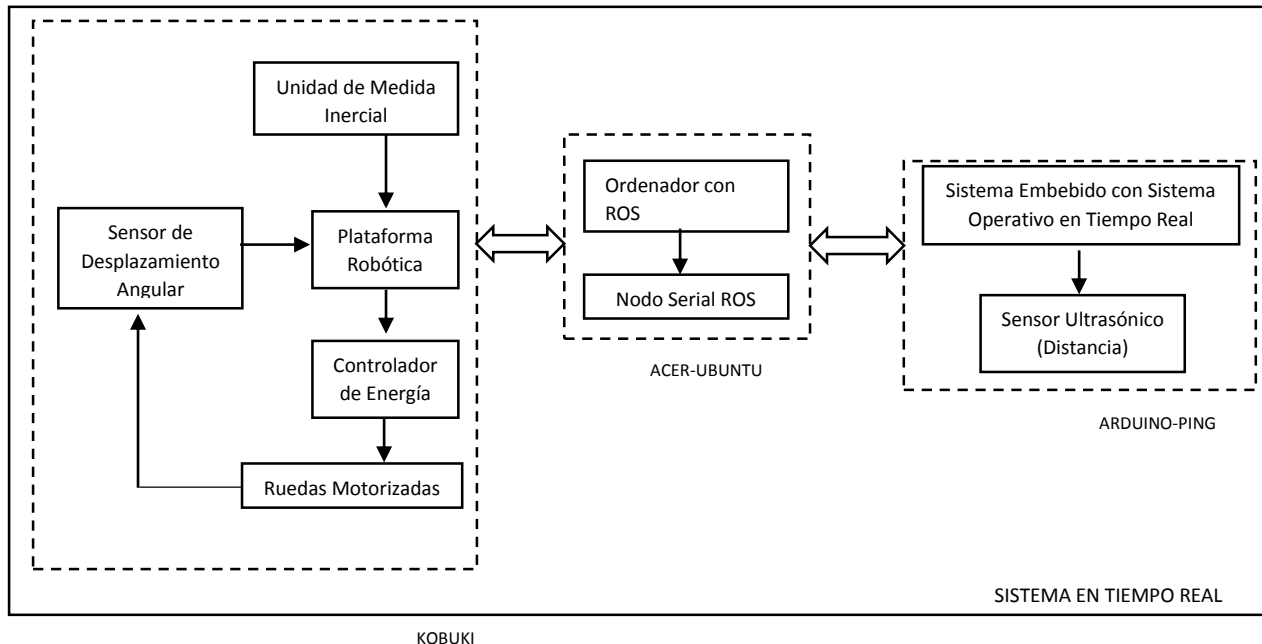
- ✓ Determinar los requerimientos del sistema.
- ✓ Detallar la configuración del sistema operativo útil para el desarrollo del sistema.
- ✓ Implementar el sistema de tiempo real sobre la plataforma.
- ✓ Realizar pruebas, análisis de resultados.

## 1.5 ALCANCE

El proyecto a implementar, se desarrollará en base a una nueva tecnología de aplicaciones y/o procesos en el ámbito académico e investigativo.

El proyecto contará de un sistema embebido con funcionamiento en tiempo real para optimizar el uso del procesador que se comunicará entre el ordenador y el Sistema Operativo de Robots (ROS), este sistema consistirá en un sensor de rango ultrasónico acoplado a un microcontrolador el que se encuentra trabajando en tiempo real.

Además, se implementará un nodo serial (ROSSerial) para la comunicación entre la placa de tiempo real y el ordenador que a su vez esta comunicado con la plataforma robótica. Se pondrá énfasis en el desarrollo de este último nodo.



**Figura 1. Sistema de Tiempo Real**

## CAPÍTULO II

### 2. REVISIÓN LITERARIA

En el siguiente capítulo se registra la información obtenida de la investigación de los temas necesarios para la elaboración del presente proyecto.

#### 2.1 Tiempo Real

A medida que la tecnología avanza nos ha permitido disponer de ordenadores más pequeños, rápidos, fiables y baratos, así también una minimización en los dispositivos electrónicos y un incremento en la capacidad de procesamiento.

Las nuevas herramientas tecnológicas que interaccionan con el mundo real, estarán sometidos a las restricciones de tiempo que imponga su entorno. Puede ocurrir que después de haber obtenido unos resultados correctos a partir de los datos de entrada, estos ya no sean útiles y pueden ser peligrosas porque se han obtenido demasiado tarde. Por ello se han implementado los “sistemas en tiempo real”. [6] Aparte de ser confiables deben cumplir otro rol muy importante que es reaccionar a tiempo antes los cambios externos que se pudieran suscitar.

Es decir, un sistema es el conjunto de componentes que trabajan en conjunto para alcanzar un propósito común. Los sistemas de tiempo real son sistemas basados en un ordenador que debe resolver diferentes aspectos de forma simultánea, rápida respuesta, reacciona ante estímulos, fallo en los componentes o en sus conexiones y posibles necesidades de adaptarse a lo largo del tiempo (mientras está en funcionamiento) ante los cambios de requerimiento y circunstancias.

### **2.1.1 Definición de un Sistema en Tiempo Real**

Existen algunas definiciones de sistemas de tiempo real, como por ejemplo:

Un sistema en tiempo real debe producir unas salidas como respuesta a unas entradas dentro de unos límites de tiempo específicos.[7]

Un sistema en tiempo real es aquel que debe producir respuestas correctas dentro de unos límites de tiempo.[8]

En todas las definiciones que existen se destacan dos aspectos muy importantes:

El primer aspecto a destacar de dichos sistemas es su tiempo de respuesta que es una parte clave. Es decir, no importa solo que sea capaz de generar un resultado correcto sino que éste se produzca en un tiempo determinado. Un ejemplo claro es el de un robot que necesita tomar una pieza de una banda sin fin. Si el robot llega tarde, la pieza ya no estará donde debía recogerla, por lo que es incorrecto su funcionamiento, otro caso sería si el robot llega antes que la pieza, la pieza aun no estaría ahí y el robot no terminaría su trabajo para el que fue programado.[8]

El segundo aspecto importante es la definición de un sistema de tiempo real, es que debe responder ante estímulos generados por el entorno dentro de un periodo de tiempo finito. Es decir, un sistema en tiempo real interactúa con el entorno (mundo físico real) adquiriendo datos del entorno y generando una acción sobre dicho entorno.

**Un sistema en tiempo real debe cumplir tres condiciones básicas:**

- ✓ Interactúa con el mundo real (proceso físico)
- ✓ Emite respuestas correctas
- ✓ Cumple restricciones temporales[8]

### **2.1.2 Clasificación de los Sistemas en Tiempo Real**

Los sistemas en tiempo real se diferencian por los plazos de tiempo que estos deben cumplir en una determinada aplicación, se clasifican así:

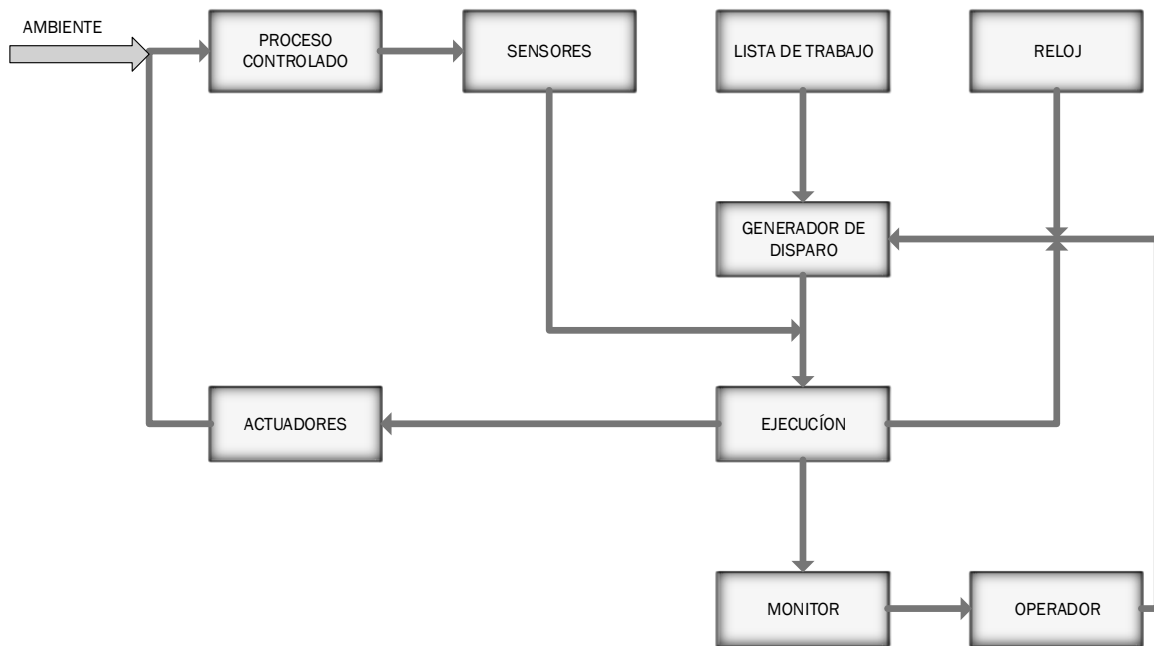
**Sistema en Tiempo Real Duro.**- El software tienen una serie de plazos estrictos y no cumplir un plazo se considera un fallo del sistema. Ejemplos de sistemas duros en tiempo real: sensor de aviones y sistemas de piloto automático, naves espaciales y vehículos de exploración planetaria.[7]

**Sistemas en Tiempo Real Blando.**- Tratan de llegar a los plazos, pero no fallan si el plazo se pierde. Sin embargo, pueden degradar la calidad del servicio en tal caso para mejorar la capacidad de respuesta. Ejemplos de sistemas de tiempo real suave: software de entrega de audio y video para el entretenimiento (el retraso no es deseable, pero no es catastrófico).[7]

**Sistema en Tiempo Real Firmes.**- Tratan la información entregada / cálculos realizados después de un plazo como no válido. Al igual que los sistemas en tiempo real blandos, que no fallan después de un plazo incumplido, y que pueden degradar la QoS si un plazo se pierde. Ejemplos de sistemas de tiempo real firmes: los sistemas de predicción financiera, líneas de montaje robotizadas.[7]

### 2.1.3 Estructura General de un Sistema en Tiempo Real

La estructura general de un sistema en tiempo real que controla un proceso cualquiera se muestra en la **Figura 2**.



**Figura 2. Diagrama de un Proceso en Tiempo Real[9]**

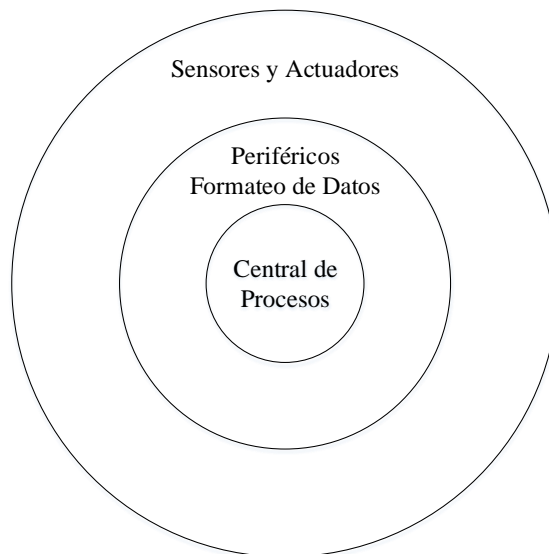
El estado del proceso bajo control y su entorno (presión, temperatura, velocidad, etc.) se adquiere mediante sensores, y se transmiten al controlador o ordenador a distinta velocidad dependiendo del parámetro concreto que se quiere medir (generalmente a velocidades inferiores a 1 Kbyte/s).

En el sistema hay una serie de tareas que debe realizar el ordenador.

El bloque generador de disparo representa el mecanismo que arranca unas determinadas tareas.

El resultado de la computación se traslada a los actuadores o bien al panel de control o pantallas de que disponga el operador. Las consignas a los actuadores suele hacerse también a una baja velocidad del orden de una consigna cada 25 ms.[10]

Todo control por ordenador presenta esta división, por un lado los sensores y actuadores trabajan a baja velocidad mientras que el ordenador debe trabajar lo suficientemente rápido para realizar correctamente los algoritmos de control. En términos de velocidad de trabajo del ordenador en tiempo real se puede definir tres capas.[10] La **Figura 3** muestra las prioridades del control por ordenador.



**Figura 3. Prioridad de Control por Ordenador [9]**

#### **2.1.4 Aplicaciones de Sistemas en Tiempo Real**

Son numerosas las aplicaciones donde se utilizan sistemas en tiempo real:

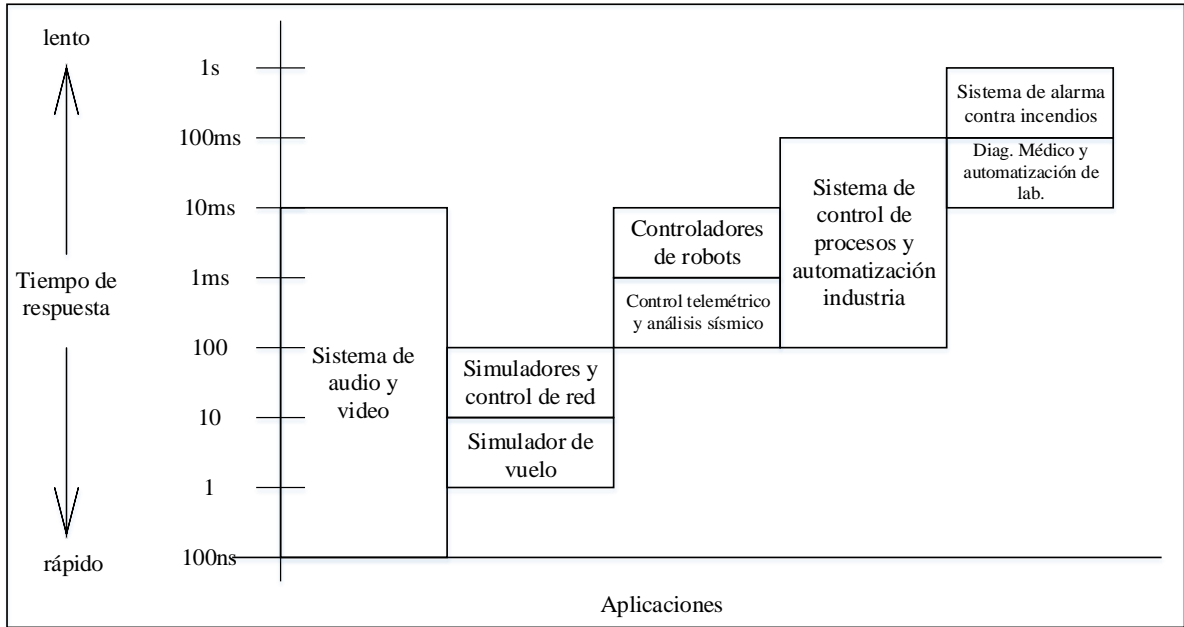
- Sistemas de defensa
- Sistemas de radar
- Control de procesos

- Aviación control de tráfico aéreo servidores multimedia
- Sistemas de comunicación
- Sistemas de satélites
- Sistemas de procesamientos de señales
- Sistemas de navegación autónoma
- Sistemas de control y adquisición de datos. Etc...

Un grupo importante de sistemas en tiempo real son los sistemas empotrados (Embedded systems). Son sistemas diseñados para una aplicación específica. Por ejemplo un teléfono móvil, el control del abs en un coche, etc. Sin embargo un ordenador personal no sería un sistema empotrado ya que no se ha concebido para que realice una aplicación concreta. Por tanto la afirmación de que todo sistema empotrado es un sistema en tiempo real es cierta, mientras que la afirmación contraria no lo es. [11]

La clasificación de un sistema en una de las categorías de tiempo real no reside en el orden de magnitud de los tiempos que se manejan sino en la importancia del cumplimiento de las restricciones temporales. En la **Figura 4** se destaca algunas aplicaciones y su tiempo de respuesta.

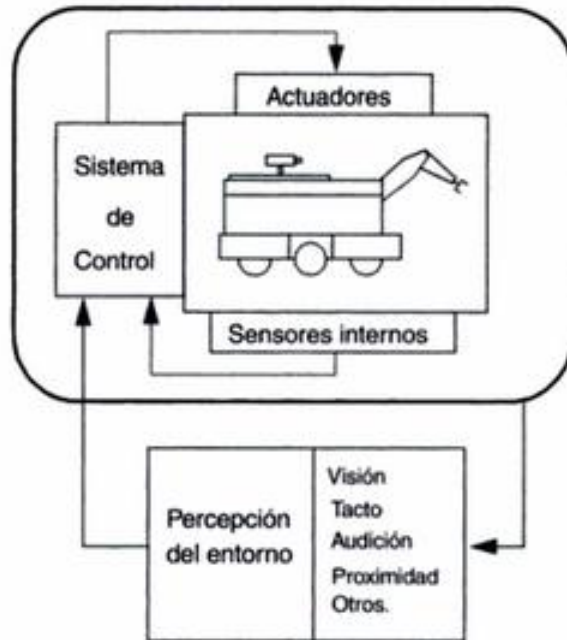




**Figura 4. Aplicaciones y su Tiempo de Respuesta[11]**

## 2.2 Tiempo Real en los Robots

Un robot es una máquina que realiza trabajos productivos y de imitación de movimientos y comportamientos de seres vivos. En la actualidad los robots son obras de ingeniería que están compuestos por sistemas mecánicos, actuadores, sensores y sistemas de control. Este último es importante ya que es el encargado de obtener la información del exterior por medio de los sensores, interpretar los datos y enviar la información a los actuadores.[10] En la **Figura 5** se detalla al robot sus elementos y la interacción con el entorno.



**Figura 5. Robot y su Interacción con el Entorno [12]**

En la industria, los robots cada vez son más indispensables en los procesos de producción y con una población en crecimiento la demanda de productos exige que estos sean más rápidos en la recolección, interpretación y envío de datos para que sean más eficientes, además, puedan realizar acciones en respuesta a eventualidades que se pueden suscitar en su entorno en lapsos de tiempo muy cortos por ellos es necesario la implementación de sistemas en tiempo real para los robots.[13]

Las ventajas de utilizar sistemas en tiempo real en los robots son: su mayor exactitud, seguridad y adaptabilidad a contingencias variadas de su entorno y en cada uno de los procesos que esté realizando ya que continuamente está adquiriendo información del medio externo.

## **2.3 Robot Operating System (ROS)**

La plataforma (ROS) es un framework de desarrollo de algoritmos de control en robótica además cuenta con paquetes que incluyen librerías de control de dispositivos para actuadores, sensores, motores. Es de código abierto, esta licencia permite libertad para uso comercial y para la investigación, también está soportado por Unix-Ubuntu actualmente y tiene un soporte experimental para Mac, Fedora, Windows, OpenSuse entre otros.

La principal ventaja de este tipo de comunicación es la creación de grandes bases de datos de manera gratuita ya que todos los ordenadores conectados en línea pueden descargar archivos de otros ordenadores también conectados.[14]

(ROS) cuenta con dos partes principales para su buen funcionamiento: el núcleo del sistema operativo y el ros-pkg. Este último un grupo de paquetes de licencia de código abierto, desarrollados por usuarios que implementan distintas funcionalidades como mapping, planning, etc.

### **2.3.1 Características Principales del Sistema Operativo de Robots (ROS)**

Las principales características de ROS son:

**Código Libre y Abierto.-** (ROS) es de código libre bajo términos de licencia BSD, contiene libertad de usos tanto comercial como de investigación esto quiere decir que está disponible al público en general.[15]

**“Peer to Peer”.-** Es un sistema distribuido en el cual los diferentes procesos se comunican entre sí utilizando una topología “peer to peer”. Con esta topología se utiliza un canal nuevo para la comunicación entre dos procesos distintos, evitando utilizar un servidor central para comunicar todos los procesos.[15]

**Multi-Lenguaje.-** Se puede trabajar en diferentes lenguajes de programación como lo son: C++, Python y Lisp. Además contiene bibliotecas en Java y Lua, en fase experimental.

**Multi-Herramientas.-** Contiene una gran cantidad de herramientas, que nos permite realizar diferentes tareas, desde navegar en los ficheros, también permite modificar los parámetros de configuración de un robot, proceso o driver, observar la topología de los procesos en ejecución y realizar la comunicación entre procesos.[15]

### 2.3.2 Conceptos Básicos del Sistema Operativo de Robots (ROS)

Para obtener un adecuado funcionamiento de (ROS) se debe tener conocimiento de los elementos fundamentales de esta plataforma como son: nodos, ROS Master, Mensajes y topics.

**Nodos.-** Son ejecutables que se comunican con otros procesos usando topics o servicios. El uso de nodos en (ROS) proporciona tolerancia a fallos y separa el código del sistema haciéndolo más simple. Un paquete puede contener varios nodos (cada nodo dispone de un nombre único), cada uno de ellos lleva a cabo una determinada acción.[15]

**ROS Master.-** Proporciona un registro de los nodos que se están ejecutando y permite la comunicación entre nodos. Sin el maestro los nodos no serían capaces de encontrar al resto de nodos, intercambiar mensajes o invocar servicios.[15]

**Mensajes.-** Los nodos se comunican entre sí mediante el paso de mensajes. Los tipos primitivos de mensajes (“integer”, “floating point”, “boolean”, etc.) están soportados y se pueden crear tipos personalizados. [15]

En la siguiente **Tabla 1** se presentan algunos ejemplos de tipos de mensajes que utiliza (ROS).

**Tabla 1. Tipos de Mensajes de ROS**

<b>Tipo Primitivo</b>	<b>Declaración tipo</b>	<b>C++</b>	<b>Python</b>
Uint16	<i>Entero 16 bits sin signo</i>	<i>uint16_t</i>	int
Float32	<i>Flotante de 32 bits</i>	<i>float</i>	float
Sting	<i>Cadena de caracteres</i>	<i>Std :: string</i>	String

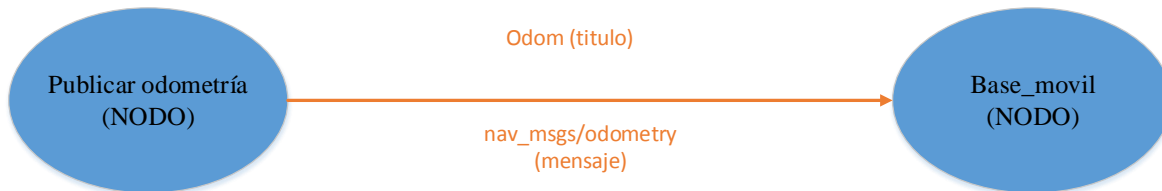
**Topics.-** Son canales de comunicación para transmitir datos. Los topics pueden ser transmitidos sin una comunicación directa entre nodos, significa que la producción y el consumo de datos esta desacoplada. Los nodos pueden comunicarse entre sí mediante topics, pudiendo actuar como publicadores (publisher) y como suscriptores (subscriber).[15]

**Publicador.** El nodo que actúa como publicador es el encargado de crear el topic por el que va a difundir ciertos mensajes. Estos mensajes podrán ser visibles por los nodos que estén suscritos a este topic.[15]

**Suscriptor.-** Este nodo se deberá suscribir a los topics correspondientes para poder acceder a los mensajes que hay publicados en ellos.

Un nodo puede publicar o suscribirse a un mensaje a través de un topic. En la Figura 6 se puede ver un ejemplo de comunicación entre dos nodos por medio de un topic. En este ejemplo, el nodo “publisher odometry” publica en el topic “Odom” un mensaje del tipo

“nav\_msgs/odometry” y el nodo “Base\_movil” accede a este mensaje suscribiéndose a este topic. En la **Figura 6** se detalla la comunicación entre tópicos. [15]



**Figura 6. Comunicación por Tópicos**

## 2.4 Tiempo real en ROS

A pesar de la importancia de la reactividad y la baja latencia en el control de robots, (ROS), en sí, no es un sistema operativo en tiempo real (RTOS), aunque es posible integrar (ROS) con código en tiempo real. Para ello debe cumplir con unos requisitos para su buen funcionamiento.[14]

Los requisitos de un sistema en tiempo real varían dependiendo del caso de uso. En su esencia, el requisito en tiempo real de un sistema tiene dos componentes:

1. Estado latente
  - Período de actualización (también conocida como fecha límite)
  - Previsibilidad
  
2. Modo de fallo
  - Cómo reaccionar ante un vencimiento del plazo

Cuando se hace referencia a los sistemas duro / blando / firmes en tiempo real generalmente se refiere al modo de fallo. Un sistema de tiempo real duro trata un plazo incumplido como un fallo del sistema. Un sistema de tiempo real blando trata de cumplir con los plazos, pero no falla cuando falta una fecha límite. Un sistema en tiempo real firme descarta cálculos realizados por el incumplimiento de plazos y puede degradar su requisito de rendimiento con el fin de adaptarse a un plazo incumplido.

Un modo de fallo de tiempo real se asocia a menudo con alta predictibilidad. Los sistemas de seguridad críticos a menudo requieren un sistema de tiempo real con alta predictibilidad.

## **2.5 Problemática y la Necesidad del Sensor Ultrasónico**

La plataforma robótica Kobuki cuenta con muchos sensores como lo son: sensores de impacto son tres que le permite en el instante de llegar a tener contacto con un objeto la plataforma robótica se detiene inmediatamente, sensores de desnivel son tres estos sensores están situados debajo del parachoques que son infrarrojos que sirven para detectar si puede seguir por la dirección prevista caso contrario tomará otra ruta, sensor de caída de rueda son dos son conmutadores de dos posiciones con muelle de retorno a su posición inicial y son accionados por medio de una palanca. Se ubican en la parte interna de las ruedas.

Sin embargo ninguno de estos sensores envía una señal de los objetos que lo rodea sin colisionar por esta razón se ha visto conveniente implementar un sensor que de un aviso temprano y así el operador tome decisiones de evasión de obstáculos evitando daños ya sea en la plataforma robótica como en su entorno sin la necesidad que el operador se encuentre cerca de la plataforma robótica.

Se seleccionó un sensor ultrasónico ya que proporciona un método sencillo de medición de distancia. Este sensor es perfecto para cualquier número de aplicaciones que requieren que se realice mediciones entre objetos en movimiento o estacionarios.

Consta de transmisor ultrasónico, receptor y circuitos de control, cuando se dispara, envía una serie de pulsos de ultrasonidos de 40KHz y recibe eco de un objeto. La distancia entre la unidad y el objeto se calcula mediante la medición del tiempo de desplazamiento del sonido y su salida como la anchura de un pulso TTL. [16]. En la **Figura 7** se muestra un sensor ultrasónico.



**Figura 7. Sensor Ultrasónico [16]**

Este sensor se puede acoplar a diferentes aplicaciones como por ejemplo: sistemas de seguridad, exposiciones animadas interactivas, sistemas de asistencia de parqueo y navegación robótica.[16]

## **2.6 Módulo de Tiempo Real con el Sistema Operativo de Robots (ROS)**

Arduino es una plataforma de electrónica de código abierto, para la creación de prototipos basada en software y hardware libre, flexible y fácil de usar.



El hardware consiste en una placa de circuito impreso con un microcontrolador, usualmente Atmel AVR, puertos digitales y analógicos de entrada y salida, pueden ser conectados a placas de expansión, que amplían las características de funcionamiento de la placa Arduino. Así mismo, posee un puerto USB desde donde se puede alimentar la placa y establecer comunicación con el ordenador. El microcontrolador de la placa se programa mediante un ordenador, usando una comunicación serial mediante un conversor de niveles RS-232 a TTL serial.[17]

El software consiste en un entorno de desarrollo IDE (Integrated Development Environment), basado en el entorno de processing y lenguaje de programación basado en Wiring, así como en el cargador de arranque (bootloader) que es ejecutado en la placa.

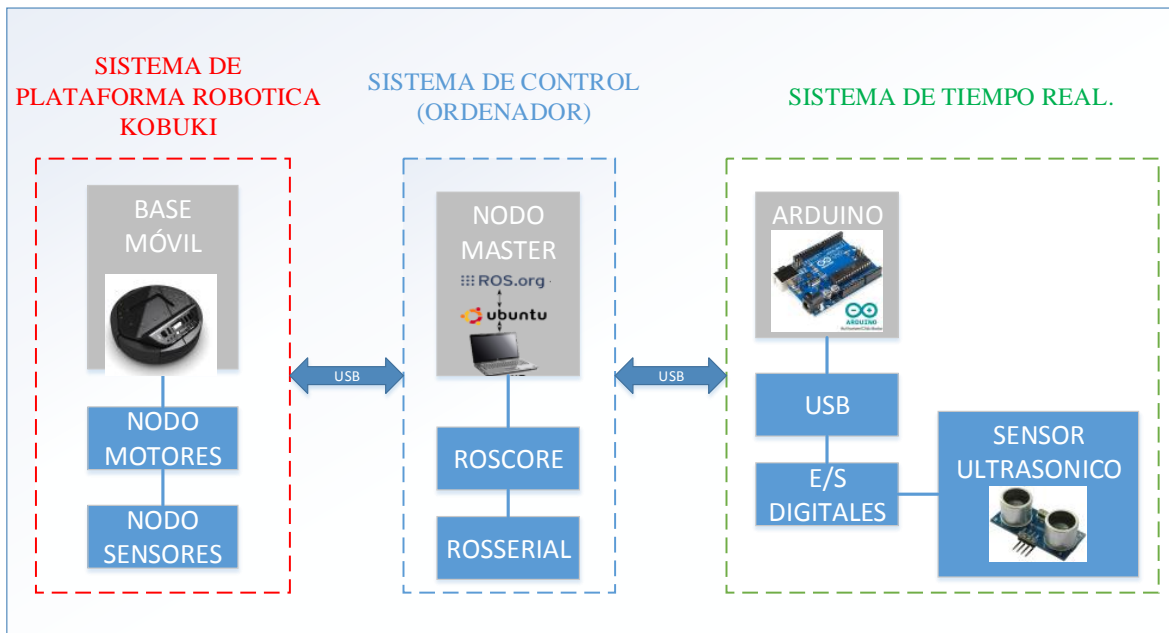
A través de los años Arduino ha sido el cerebro de miles de proyectos, a partir de objetos cotidianos a los instrumentos científicos complejos.



**Figura 8. Microcontrolador Arduino [18]**

Al unir esta útil herramienta con (ROS) que conjugadas con el hardware generan un automatismo de alto rango.

## 2.7 Diagrama General del Sistema



**Figura 9. Esquema General del Sistema [19]**

Como se puede evidenciar en la **Figura 9**, el sistema se divide en tres partes importantes los que se detallaran a continuación.

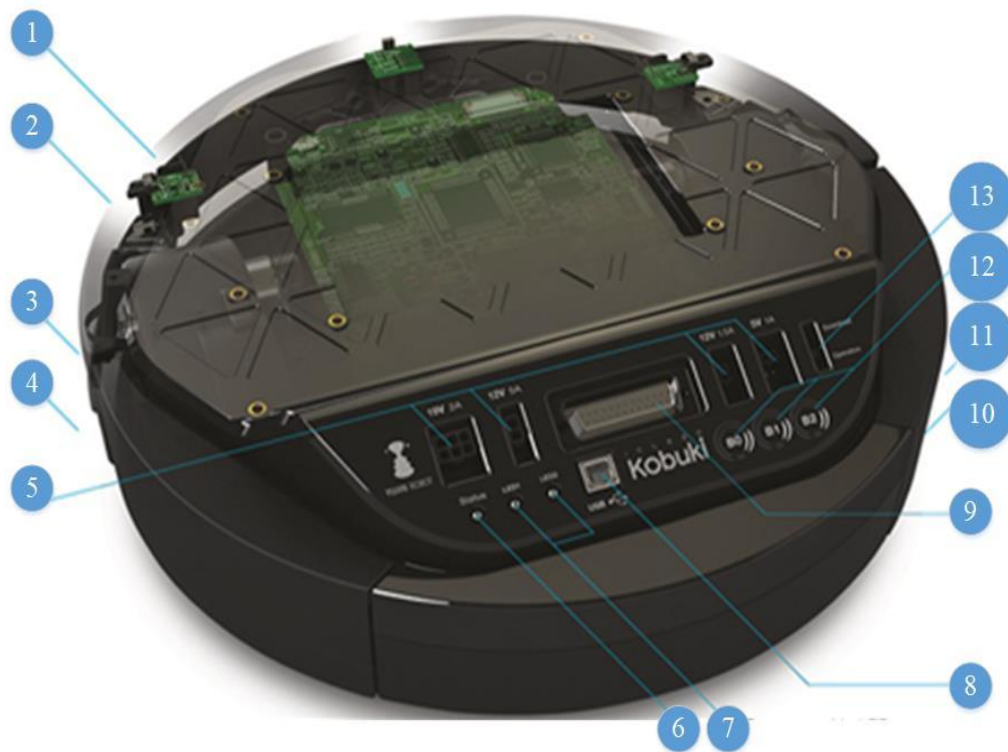
### 2.7.1 Sistema de Plataforma Robótica (Kobuki)

IClebo Kobuki es una base de investigación móvil de bajo costo diseñada para la educación y la investigación sobre el estado de la robótica de arte. Con la operación continua en mente, Kobuki proporciona fuentes de alimentación para un ordenador externo, así como sensores y actuadores adicionales. Su odometría altamente precisa, enmendada por nuestro giroscopio calibrado en fábrica, permite una navegación precisa.

El Kobuki es una base móvil. Tiene sensores, motores y fuentes de energía, sin embargo por sí mismo, no puede hacer nada.

Para ser funcional, se requiere construir una plataforma encima de la hoja de comandos. En el lado del hardware, esto suele implicar la adición de un notebook o una placa incorporada para ser el núcleo computacional para su sistema y por lo general algunos sensores extra para que sea realmente funcional. En el lado del software, esto implica construir cualquier software propio o integrar software de otros grupos con sus propias fuentes de desarrollo.

En la figura se muestra de la plataforma robótica Kobuki y en la tabla se detalla los elementos principales de ésta.



**Figura 10. Plataforma Robótica Kobuki [19]**

En la **Tabla 2** representa los elementos que conforman la Plataforma Robótica Kobuki.

**Tabla 2. Elementos de la Plataforma Robótica Kobuki [20]**

<b>N°</b>	<b>ELEMENTOS</b>
<b>1</b>	Sensor de Acoplamiento (Izquierdo/Central/Derecho)
<b>2</b>	Sensor de Impacto (Izquierdo/Central/Derecho)
<b>3</b>	Sensor de Acantilado (Izquierdo/Central/Derecho)
<b>4</b>	Sensores de Caída de Rueda
<b>5</b>	Conectores de Alimentación
<b>6</b>	Estado de la Batería (led)
<b>7</b>	Leds Programables.
<b>8</b>	Conexión de Datos al Ordenador por USB
<b>9</b>	Puerto de Extensión Serial  Rx/Tx  3.3V/5V  4x Entradas Digitales  4x Salidas Digitales  4x Entradas Analógicas
<b>10</b>	Toma de Recarga
<b>11</b>	Interruptor Encendido / Apagado
<b>12</b>	Botones Programables
<b>13</b>	Interruptor de descarga de Firmware

### 2.7.1.1 Nodo Actuador (motor)

Los encargados de transmitir las ordenes de programación en movimiento son los motores DC de escobillas su número de serie en el mercado es RP385-ST-2060, están ubicados a los costados del robot que transmiten el movimiento a través de engranes. Su velocidad máxima es de 0.65 m/s tanto hacia delante como en retroceso.

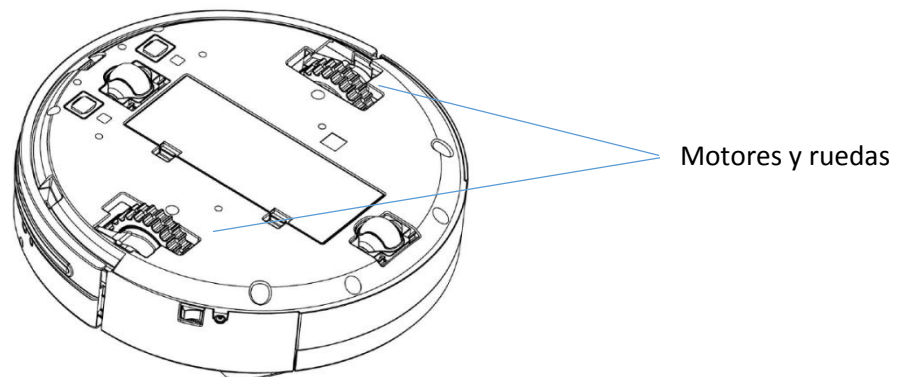
En la **Tabla 3**. Se detalla las especificaciones del nodo actuador.[21]

**Tabla 3. Características Principales del Actuador [22]**

ELEMENTO	CARACTERÍSTICAS
<b>Alimentación del motor</b>	Corriente directa
<b>Fabricante del motor</b>	Motor estándar
<b>Nombre de la pieza</b>	RP385-ST-2060
<b>Tensión nominal</b>	12 V
<b>Carga nominal</b>	5 mN · m
<b>Sin corriente de carga</b>	210 Ma
<b>Velocidad sin carga</b>	9960 rpm ± 15%
<b>Corriente de carga</b>	750 mA
<b>nominal</b>	
<b>Velocidad de carga</b>	8800 rpm ± 15%
<b>nominal</b>	
<b>Resistencia de armadura</b>	1.5506 Ω a 25 ° C
<b>Inductancia del inducido</b>	1,51 mH

<b>Constante de par (Kt)</b>	10.913 mN · m / A
<b>Constante de velocidad (Kv)</b>	830 rpm / V
<b>Corriente de bloqueo</b>	6.1 A
<b>Par de bloqueo</b>	33 mN · m

El método de control de los actuadores (motores) es a base de impulsado por la fuente de tensión (puente H) y es controlado por la modulación de ancho de pulso (PWM). En la **Figura 11** se detalla la posición de los actuadores en la plataforma robótica Kobuki.



**Figura 11. Ubicación de los Motores y Ruedas en el Robot Móvil [21]**

### 2.7.1.2 Nodo Sensores.

Kobuki posee sensores que son muy útiles en la navegación, teleoperación y otras diversas aplicaciones a continuación se detallan:

### 2.7.1.2.1 Sensores de Impacto

La plataforma robótica tiene tres sensores de impacto, estos tienen la función de detectar los diferentes obstáculos en su desplazamiento por un contacto directo con el objeto.

Los sensores de choque son conmutadores de dos posiciones con muelle de retorno a su posición inicial y son accionados por medio de una palanca. Se ubican en la parte interna del robot y la detección es producida por el golpe de los parachoques. En la **Figura 12** se muestra.



**Figura 12. Sensor de Impacto [19]**

### 2.7.1.2.2 Codificador de Cuadratura

La Plataforma Robótica Kobuki posee dos sensores de rotación su principal funcionamiento es permite el desplazamiento angular, distancia recorrida, ángulo de giro y sentido de giro. Estos sensores se encuentran en cada una de las ruedas de diámetro de 70 mm. Además los sensores son muy útiles en la odometría del robot.



**Figura 13. Codificador de Cuadratura[19]**

### **2.7.2 Sistema de Control (Ordenador)**

Este sistema está compuesto del software como de hardware. El software está compuesto por un ordenador de marca Acer que cumple con los requerimientos necesarios para el funcionamiento óptimo del robot.

El hardware que se empleara para realizar el proyecto, es el sistema operativo Ubuntu 14.04 LTE. Para el buen funcionamiento y desarrollo del proyecto la elección del sistema operativo es fundamental. Teniendo en conocimiento que (ROS) también puede ser instalado en Windows, las aplicaciones desarrolladas (Paquetes) deben cumplir ciertas limitaciones computacionales. Además, cabe mencionar (ROS) solo es totalmente funcional en la plataforma Linux fundamentalmente para la distribución Ubuntu; con otras distribuciones no garantizan el correcto funcionamiento de (ROS).

Para concluir, se ha escogido Linux, concretamente la distribución Ubuntu 14.04 LTS, cuyo origen se basa en Debian, además se eligió esta versión por ser LTS sus siglas están en inglés (Long Term Support), esto significa que es una versión de Ubuntu que tendrá soporte y será actualizada más tiempo que una versión normal.



Del mismo modo, las versiones LTS suelen ser versiones más estables y probadas. Además, las versiones LTS de Ubuntu tienen soporte durante 5 años y las versiones normales tienen un soporte de 9 meses, luego de ese tiempo tendrá que actualizarse en un periodo relativamente corto de tiempo.

En el interior del sistema operativo robótico (ROS) se está ejecutando dos nodos sumamente importantes para la comunicación entre el ordenador y el sistema en tiempo real estos son: roscore y roserial.[23]

#### **2.7.2.1 Roscore**

Es un conjunto de nodos y programas que es condición previa para la utilización de un sistema basado en (ROS). Se debe ejecutar un roscore con la finalidad de que (ROS) se pueda comunicar con todos los nodos. Se lo inicia con el comando roscore.

#### **2.7.2.2 Rosserial**

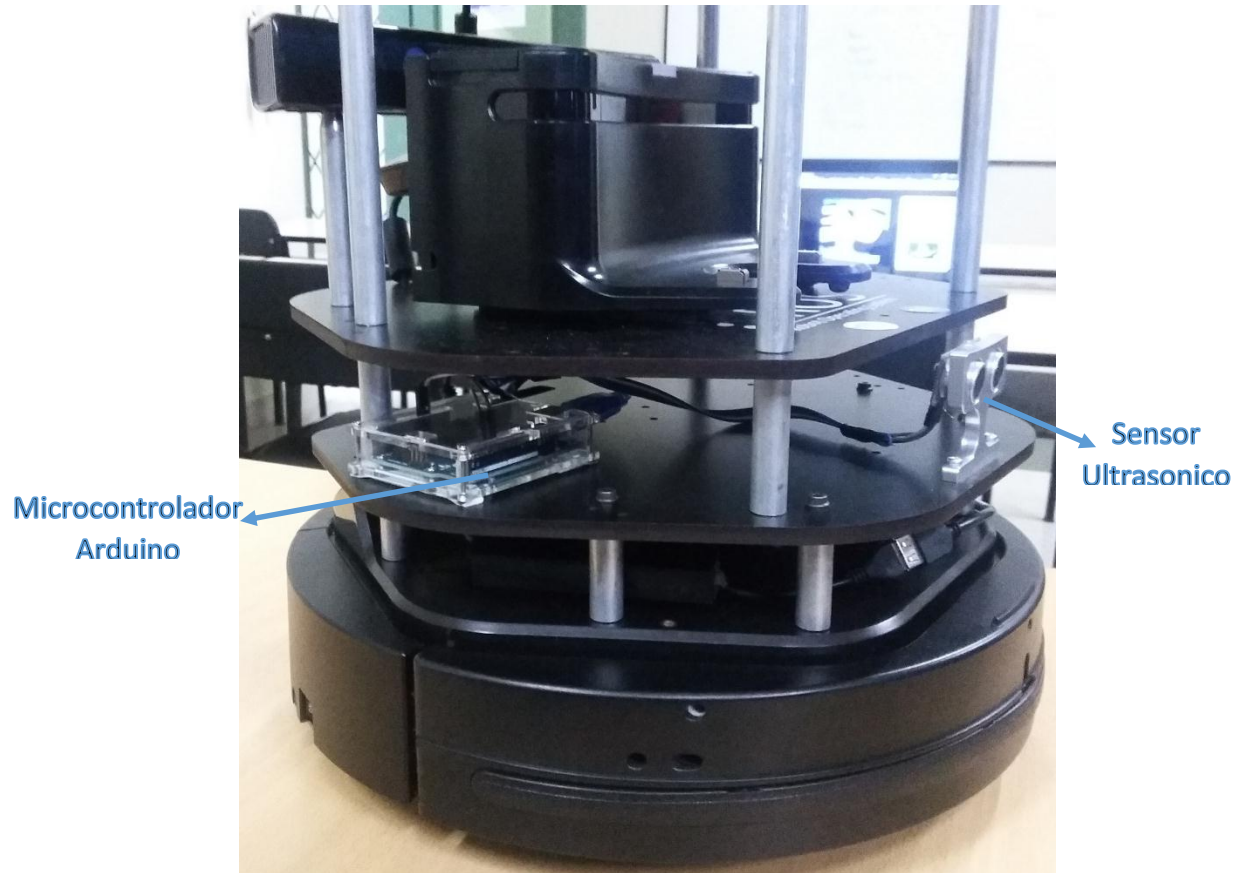
ROS Serial es una versión punto a punto de las comunicaciones (ROS) sobre serie, principalmente para la integración de microcontroladores de bajo coste (Arduino) en (ROS). ROS serie consta de bibliotecas para uso con Arduino, y nodos para el lado PC / Tablet (actualmente en Python y Java).[24]

#### **2.7.3 Sistema de Tiempo Real**

Para la parte del sistema en tiempo real se empleó el microcontrolador Arduino y el sensor ultrasónico que se conectan por medio de cables, y los dos elementos se comunican al ordenador por medio de UBS.

Frecuencia de muestreo del sensor ultrasónico es de 1 muestras por segundo.

En la **Figura 14** se muestra la ubicación del sistema de tiempo real implementado sobre la plataforma robótica con sus componentes.



**Figura 14. Sistema de Tiempo Real Implementado [19]**

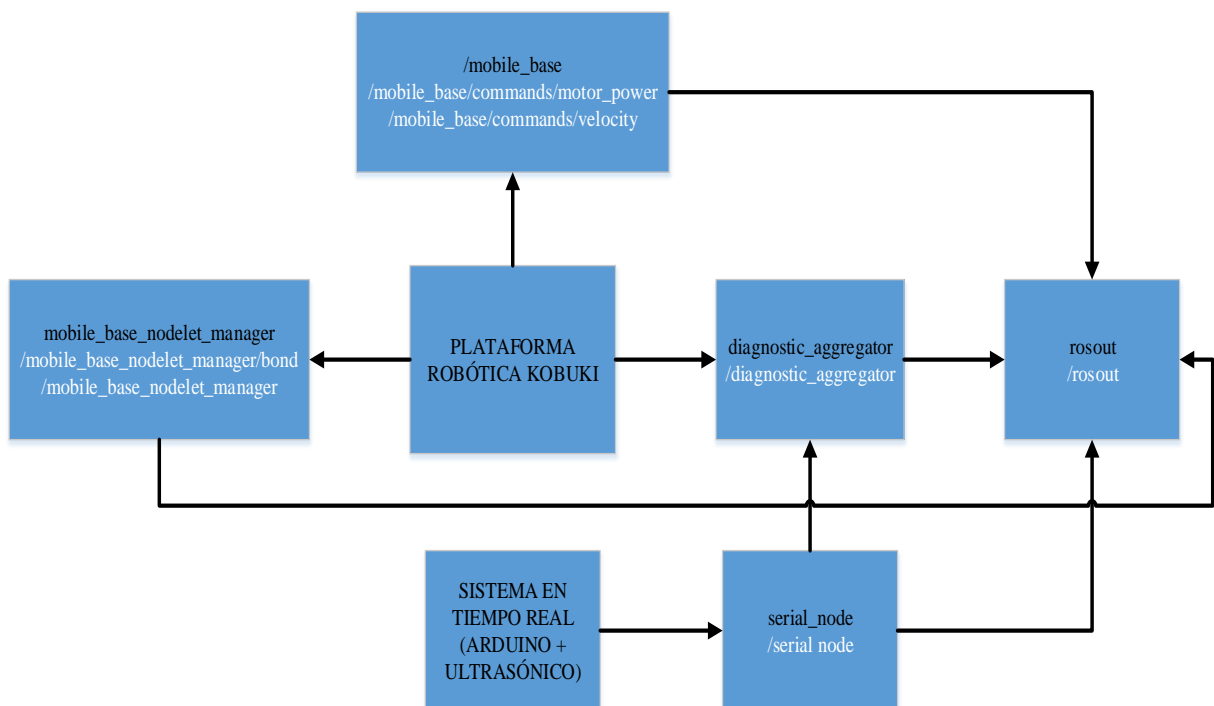
## CAPÍTULO III

### 3. Caracterización del Módulo de Tiempo Real

En este capítulo se detallan los bloques de los que está constituido el robot Kobuki, incluyendo el nodo de tiempo real.

#### 3.1 Diagrama de Bloques del Robot Kobuki ejecutando el Sistema en Tiempo Real

En la **Figura 15** se visualiza todos los nodos que se están ejecutando en la plataforma robótica Kobuki en tiempo real.



**Figura 15. Diagrama de Bloques de los Sistemas en Tiempo Real [19]**

### **3.1.1 Plataforma Robótica Kobuki-Nodos**

Se detalla los nodos que se encuentran comunicados al ejecutar el tiempo real.

#### **3.1.1.1 Mobile\_Base\_Nodelet\_Manager (Administrador de nodo de base móvil)**

Este nodo está diseñado para proporcionar una forma de ejecutar múltiples algoritmos en una sola máquina, en un solo proceso, sin incurrir en costos de copia al pasar mensajes interproceso. Para ello, los nodos permiten la carga dinámica de las clases en el mismo nodo, sin embargo, proporcionan espacios de nombres separados simples, de manera que el nodelet actúa como un nodo separado, a pesar de estar en el mismo proceso.[11]

La función principal es los flujos de datos de alto rendimiento pueden estar compuestos de muchos nodos y luego cargarse en el mismo proceso para evitar la copia y el tráfico de red.

#### **3.1.1.2 Mobile\_Base (base móvil)**

En este nodo se encuentran los elementos que necesarios para operar la plataforma robótica Kobuki como son los motores, velocidad, sensores.

#### **3.1.1.3 Diagnostic Aggregator**

El sistema de diagnóstico está diseñado para recopilar información de controladores de hardware y hardware de robot para usuarios y operadores para análisis, solución de problemas y registro. La pila de diagnósticos contiene herramientas para recopilar, publicar, analizar y visualizar datos de diagnóstico.

El diagnóstico\_agregador contiene un nodo ROS, agregador\_nodo, que escucha los mensajes diagnostic\_msgs / DiagnosticArray en el tema / diagnostics, procesa y categoriza los datos y

vuelve a publicar en / diagnostics\_agg. El agregador\_nodo carga los complementos "Analyzer" para realizar el proceso de diagnóstico y la categorización. La configuración de cada agregador de diagnóstico es específica para cada robot y puede ser determinada por los usuarios o desarrolladores.[26]

#### **3.1.1.4 Rosout**

- ✓ Rosout es el nombre del mecanismo de informes de registros de consola en (ROS). Se puede pensar que comprende varios componentes:
- ✓ El nodo `rosout` para suscribir, registrar y volver a publicar los mensajes.
- ✓ El tema / rosout.
- ✓ El tema / rosout\_agg para suscribirse a un alimentador agregado.
- ✓ Rosgraph\_msgs / Log tipo de mensaje, que define los campos estándar, así como los niveles de verbosidad.
- ✓ API de cliente para facilitar el uso fácil del mecanismo de generación de informes
- ✓ Herramientas de la GUI, como rqt\_console , para ver los mensajes de registro de la consola.[27]

### **3.1.2. Sistema en Tiempo Real**

#### **3.1.2.1 Nodo Serial**

Este paquete contiene extensiones específicas de Arduino necesarias para ejecutar rosserial\_client en el Arduino. Se pretende demostrar lo cómodo que es integrar hardware personalizado y sensores económicos en el proyecto (ROS) utilizando un Arduino.

Gracias a este nodo se realiza la comunicación entre el ordenador y la placa de tiempo real, que permite visualizar los datos obtenidos del sensor.[24]

### 3.1.2.2 Instalación del Software

En primer lugar es necesario instalar roserial para Arduino. Para descargar el paquete se realiza los siguientes pasos:

Colocar desde un terminal, dentro de nuestro directorio de trabajo de (ROS) donde se va a descargar a descargar el paquete. En caso catkin\_workspace/src.

Descargar el paquete ejecutando:

```
git clone https://github.com/ros-drivers/roserial.git
```

Compilar el paquete ejecutando:

```
catkin_make install.
```

Recompilar las fuentes en (ROS):

```
source install/setup.bash
```

Finalmente compilar la librería de roserial:

```
roslaunch roserial_arduino make_libraries.py
```

### 3.1.2.3 Instalación de la Librería ROS\_lib en el Entorno de Arduino

El procedimiento es sencillo, basta con copiar el directorio `ros_lib`, que se encuentra en dentro de nuestro programa Arduino IDE.[28]

Lo primero es localizar donde se encuentra instalado el paquete `roserial_arduino` que se acaba de instalar, ejecutando el siguiente comando desde un terminal:

```
rospack find roserial_arduino
```

Si el paquete ha sido instalado correctamente el comando nos devolverá su ruta. Una vez localizada su ubicación se desplaza al directorio `src`, se lo hace mediante el siguiente comando:

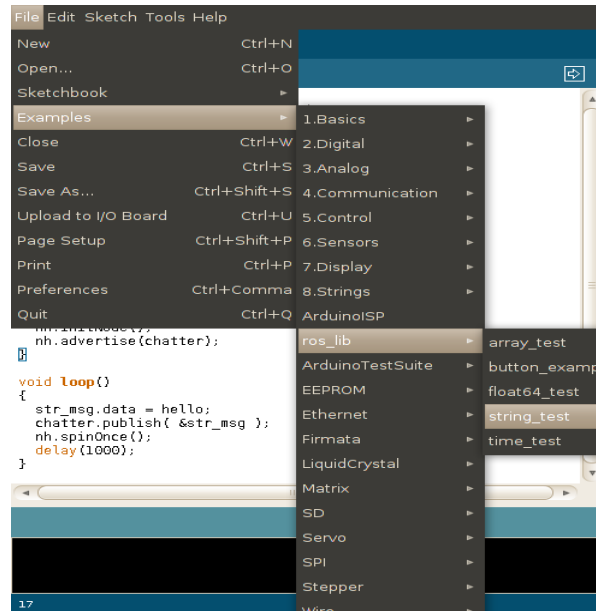
```
roscd roserial_arduino/src
```

Dentro de este directorio se encuentra el directorio `ros_lib`. Se lo copia al directorio librerías de nuestro Arduino IDE, en mi caso el comando a ejecutar sería:

```
cp-r ros_lib ~/arduino-1.5.6-r2/libraries/ros_lib
```

Una vez copiado se inicia el IDE de Arduino. En la opción del menú se puede encontrar una nueva opción llamada `ros_lib` con ejemplo de uso.

Después de reiniciar su IDE, se obtiene una imagen como se muestra en la **Figura 16** de la nueva librería instalada `ros_lib` en la lista de ejemplos:



**Figura 16. Instalación de la Librería ros-lib [28]**

### 3.1.2.4 Ejemplo de su Uso

A continuación, un ejemplo de cómo utilizar esta librería con Arduino.

Se abre el IDE y dentro de los ejemplos que nos propone esta librería se abre HelloWorld.

Si no estuviera el código del sketch es el siguiente:

```

/*
 * roserial Publisher Example
 * Prints "hello world!"
 */
#include <ros.h>
#include <std_msgs/String.h>

ros::NodeHandle nh;

std_msgs::String str_msg;

ros::Publisher chatter("chatter", &str_msg);
  
```



```

char hello[13] = "hello world!";

void setup()
{
  nh.initNode();
  nh.advertise(chatter);
}

void loop()
{
  str_msg.data = hello;
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(1000);
}

```

Esta sketch simplemente crea un nodo (ROS) llamado chatter, que publicará la cadena **HELLO WORD**.

Se inicia (ROS) mediante el comando roscore. Lo siguiente será conectar el Arduino al ordenador y cargar el programa.

Y aquí viene el truco. Para permitir la comunicación entre el Arduino y (ROS) a través de un puerto serie es necesario ejecutar el paquete roserial, que se instaló previamente. Con el cable USB conectado al Arduino se comprueba el puerto de conexión, en mi caso es /dev/ttyACM0.

Sabiendo el puerto se ejecuta el siguiente comando:

```
roslaunch rosserial_python serial_node.py /dev/ttyACM0
```

Si todo ha ido correctamente se visualiza el siguiente mensaje:

```
[INFO] [WallTime: 1410022427.334118] ROS Serial Python Node
[INFO] [WallTime: 1410022427.339428] Connecting to /dev/ttyACM0 at
57600 baud
[INFO] [WallTime: 1410022430.473442] Note: publish buffer size is
280 bytes
[INFO] [WallTime: 1410022430.473836] Setup publisher on chatter
[std_msgs/String]
```

En este momento ya está funcionando nuestro nodo (ROS) en el Arduino. Para comprobarlo, se ejecuta el siguiente comando:

```
rostopic /list
```

En la pantalla se debe visualizar la siguiente respuesta:

```
/chatter
/diagnostics
/rosout
/rosout_agg
```

Si todo está funcionando correctamente se verá que el nodo /chatter en la lista de nodos de (ROS).

## CAPÍTULO IV

### 4. IMPLEMENTACIÓN

En este capítulo se detalla paso a paso como fue ensamblado el sistema de tiempo real tanto su software como su hardware. Además el acoplamiento de dicho sistema con la plataforma robótica Kobuki.

#### 4.1 Programa del Microcontrolador (Arduino)

Gracias a este hardware, de bajo coste, se puede realizar fácilmente grandes proyectos de robótica. Sin embargo, como todo microcontrolador su capacidad es limitada, sobre todo en sus primeras versiones. Parece claro que en grandes proyectos de robótica es necesario conectar el Arduino con una CPU central más potente. Esta CPU sería la encargada de las tareas más complejas, dejando al Arduino como elemento de comunicación con los sensores y actuadores.

Así que una evolución natural sería utilizar (ROS) en la CPU central y que este se conectase con el Arduino para obtener y enviar información a los sensores y actuadores. Esta comunicación se puede realizar a través del paquete `rosserial_arduino`. Mediante este paquete se puede usar (ROS) con el IDE de Arduino. `Rosserial` proporciona un protocolo de comunicaciones en (ROS) que permite trabajar con Arduino. Permite al Arduino crear un nodo, éste pueda publicar o suscribir mensajes (ROS), transformaciones TF y obtener parámetros del sistema (ROS).

### 4.1.1 Programa de Medición de Distancia y Publicación de Datos

```
#include <ros.h>

#include <ros/time.h>

#include <sensor_msgs/Range.h>

ros::NodeHandle nh;

sensor_msgs::Range range_msg;

ros::Publisher pub_range( "/ultrasound", &range_msg);

const int adc_pin = 0;

char frameid[] = "/ultrasound"

int echoPin = 5;

int trigPin = 4;

float getRange_Ultrasound(){

float duration, cm;

digitalWrite(trigPin, LOW);

    delayMicroseconds(2);

    digitalWrite(trigPin, HIGH);

    delayMicroseconds(10);

    digitalWrite(trigPin, LOW);

    duration = pulseIn(echoPin, HIGH);

    float m = (duration / 58.2)/100;

    return m ;    // (0.0124023437 /4) ; //cvt to meters

}
```

```

void setup() {
    nh.initNode();
    nh.advertise(pub_range);

    range_msg.radiation_type = sensor_msgs::Range::ULTRASOUND;
    range_msg.header.frame_id = frameid;
    range_msg.field_of_view = 0.1; // fake
    range_msg.min_range = 0.05;
    range_msg.max_range = 3.0;

    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
}

long range_time;
void loop() {
    if ( millis() >= range_time ){
        range_msg.range = getRange_Ultrasound();
        range_msg.header.stamp = nh.now();
        pub_range.publish(&range_msg);
        range_time = millis() + 50;
    }

    nh.spinOnce();
}

```

### 4.1.2 Código Explicado

Se detallan todas las partes de programa que se está ejecutando en el nodo de tiempo real.

```
#include <ros.h>

#include <ros/time.h>

#include <sensor_msgs/Range.h>
```

El primer paso es verificar las librerías las que se importaran, sin este paso tan importante no se podría ejecutar la parte de (ROS) como es la publicación y suscripción al nodo de tiempo real. Debe incluir el archivo de encabezado ros.h y los archivos de encabezado para los mensajes que va a utilizar.

```
ros::NodeHandle nh;
```

Se necesita instalar el identificador de nodo, lo que permite a nuestro programa crear editores y suscriptores. El identificador de nodo también se encarga de las comunicaciones del puerto serie.

```
sensor_msgs::Range range_msg;

ros::Publisher pub_range( "/ultrasound", &range_msg);
```

Aquí se inicia un editor con un nombre de tema de "ultrasound". El segundo parámetro de Publisher es una referencia a la instancia de mensaje que se va a utilizar para la publicación.

```
const int adc_pin = 0;

char frameid[] = "/ultrasound";

int echoPin = 5;
```

```
int trigPin = 4;
```

Se define los números de los pines en los cuales van a ir conectados en el microcontrolador Arduino.

```
float getRange_Ultrasound() {  
float duration, cm;
```

Se define las variables que se utilizaran en el programa.

```
digitalWrite(trigPin, LOW);  
delayMicroseconds(2);
```

El siguiente paso es limpiar el pin trigPin .

```
digitalWrite(trigPin, HIGH);  
delayMicroseconds(10);  
digitalWrite(trigPin, LOW);
```

Establece el trigPin en estado ALTO durante 10 micro segundos después de transcurrir el tiempo regresa al estado BAJO.

```
duration = pulseIn(echoPin, HIGH);
```

Lee el echoPin, devuelve el tiempo de recorrido de la onda sonora en microsegundos.

```
float m = (duration / 58.2)/100;
```

```
return m ; // (0.0124023437 /4) ; //
```

Esta sección de programa se encarga de calcular la distancia.

```
void setup() {  
    nh.initNode();  
    nh.advertise(pub_range);
```

En la función de configuración de Arduino, deberá inicializar el manejador del nodo (ROS), anunciar cualquier tema que se publique y suscribirse a cualquier tema que desee escuchar.

```
range_msg.radiation_type = sensor_msgs::Range::ULTRASOUND;  
range_msg.header.frame_id = frameid;  
range_msg.field_of_view = 0.1; // fake  
range_msg.min_range = 0.05;  
range_msg.max_range = 3.0;  
pinMode(trigPin, OUTPUT);  
pinMode(echoPin, INPUT);
```

En esta parte del programa se ingresa los rangos de medición del sensor. Para este caso el valor mínimo de rango será 0.05 que sería 5 centímetros y un valor máximo de 3, que sería 3 metros.

```
long range_time;  
void loop() {  
    if ( millis() >= range_time ){
```



```
range_msg.range = getRange_Ultrasound();  
range_msg.header.stamp = nh.now();  
pub_range.publish(&range_msg);  
range_time = millis() + 4000;
```

Finalmente, en la función de bucle, el nodo publica las mediciones que obtiene del sensor y llama a `ros::spinOnce()` donde se manejan todas las devoluciones de llamada de comunicación ROS además se está publicando los datos una vez por segundo.

### 4.1.3 Diagrama de flujo del programa ultrasonido

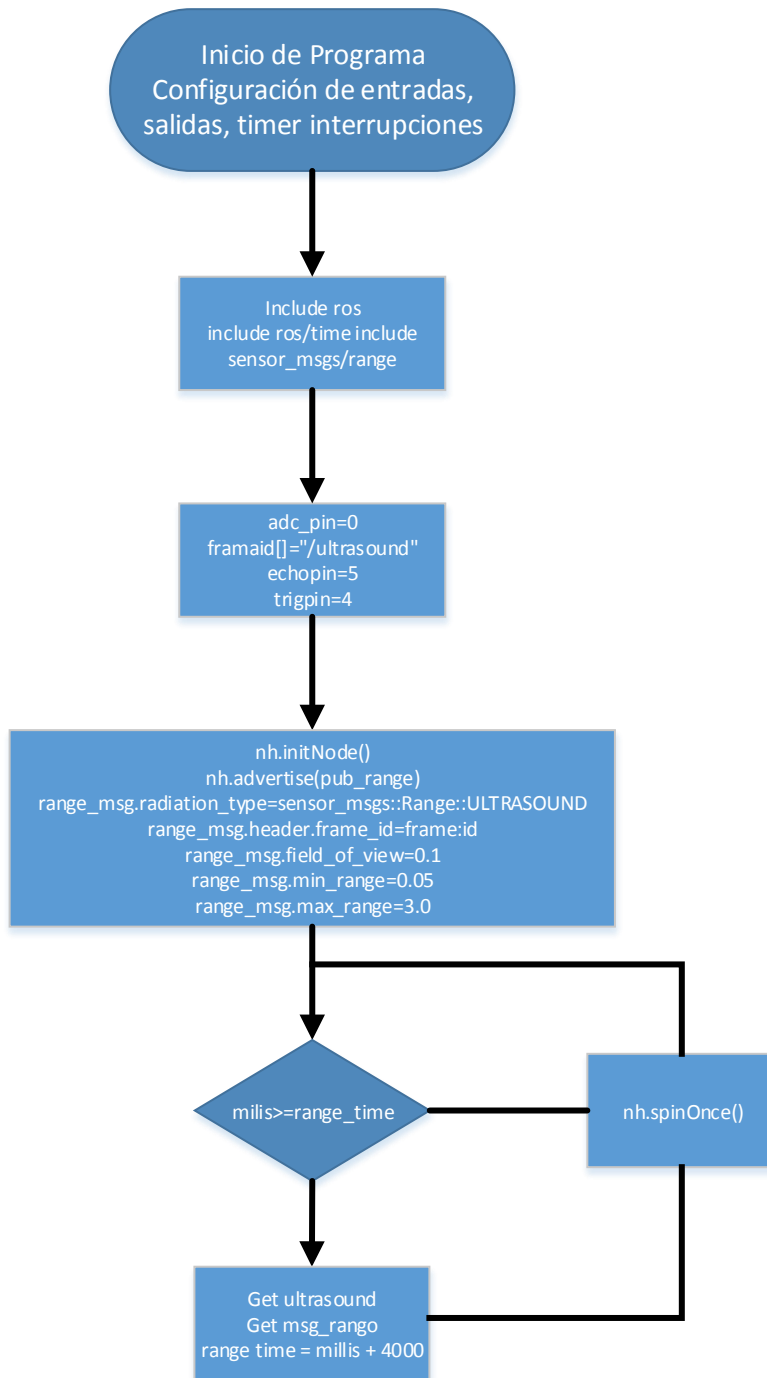
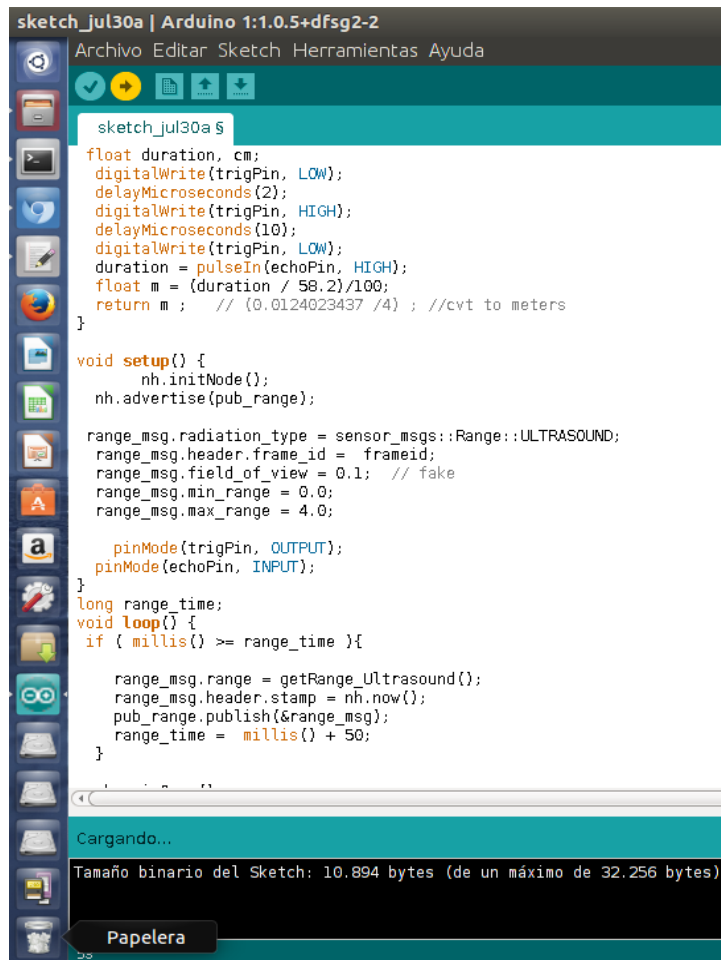


Figura 17. Diagrama de Flujo del Programa Ultrasonico [19]

## 4.2 Pasos para la Implementación y Verificación

A continuación se detallan los pasos que se deben seguir para el programa que se implementó de tiempo real

1.- Se carga el programa realizado en el software Arduino a la placa Arduino uno como se muestra en la **Figura 18**.



```
sketch_jul30a | Arduino 1:1.0.5+dfsg2-2
Archivo Editar Sketch Herramientas Ayuda
sketch_jul30a $
float duration, cm;
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH);
float m = (duration / 58.2)/100;
return m ; // (0.0124023437 /4) ; //cvt to meters
}

void setup() {
  nh.initNode();
  nh.advertise(pub_range);

  range_msg.radiation_type = sensor_msgs::Range::ULTRASOUND;
  range_msg.header.frame_id = frameid;
  range_msg.field_of_view = 0.1; // fake
  range_msg.min_range = 0.0;
  range_msg.max_range = 4.0;

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

long range_time;
void loop() {
  if ( millis() >= range_time ){

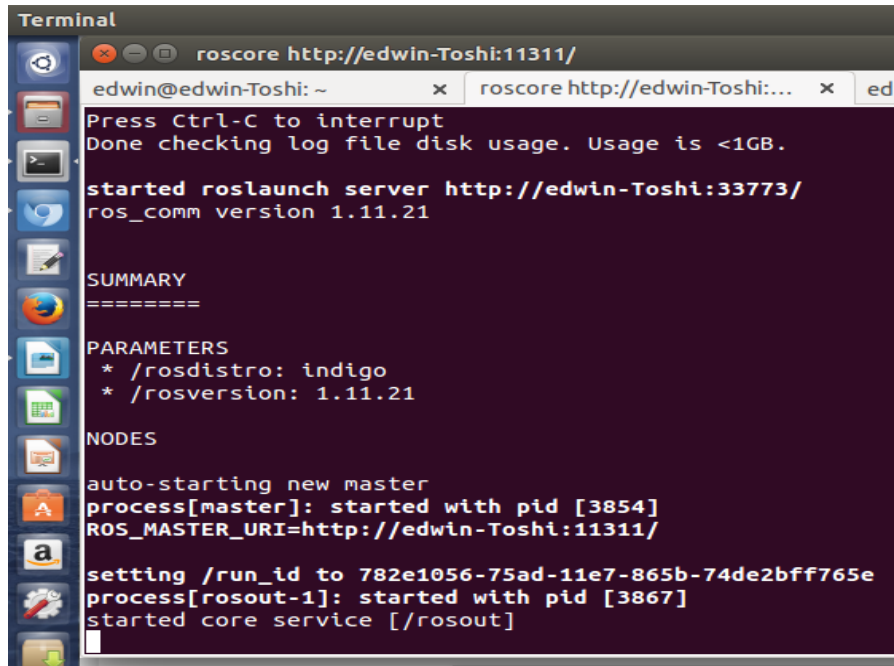
    range_msg.range = getRange_Ultrasound();
    range_msg.header.stamp = nh.now();
    pub_range.publish(&range_msg);
    range_time = millis() + 50;
  }
}

Cargando...
Tamaño binario del Sketch: 10.894 bytes (de un máximo de 32.256 bytes)
Papelera
```

**Figura 18. Visualización de Carga de Programa hacia el Arduino Uno [19]**

2.- Se abre un nuevo terminal en Ubuntu y se ejecuta roscore. Éste es una colección de nodos y programas que son pre-requisitos de un sistema basado en (ROS). Esto se lo hace con el fin de que (ROS) nodos se comuniquen. Se inicia con el comando roscore.

```
roscore
```

A terminal window titled "Terminal" showing the execution of the roscore command. The terminal output includes: "Press Ctrl-C to interrupt", "Done checking log file disk usage. Usage is <1GB.", "started roslaunch server http://edwin-Toshi:33773/", "ros\_comm version 1.11.21", "SUMMARY", "PARAMETERS", "\* /rostdistro: indigo", "\* /rosversion: 1.11.21", "NODES", "auto-starting new master", "process[master]: started with pid [3854]", "ROS\_MASTER\_URI=http://edwin-Toshi:11311/", "setting /run\_id to 782e1056-75ad-11e7-865b-74de2bfff765e", "process[rosout-1]: started with pid [3867]", "started core service [/rosout]".

```
Terminal
roscore http://edwin-Toshi:11311/
edwin@edwin-Toshi: ~
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
started roslaunch server http://edwin-Toshi:33773/
ros_comm version 1.11.21
SUMMARY
=====
PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.21
NODES
auto-starting new master
process[master]: started with pid [3854]
ROS_MASTER_URI=http://edwin-Toshi:11311/
setting /run_id to 782e1056-75ad-11e7-865b-74de2bfff765e
process[rosout-1]: started with pid [3867]
started core service [/rosout]
```

**Figura 19. Ejecución de Roscore [19]**

3.- Se ingresa el código en un nuevo terminal

```
roslaunch kobuki_node minimal.launch --screen
```

Con este código nos permite la comunicación entre la plataforma robótica Kobuki y el ordenador. Si la comunicación se realizó con éxito en el terminal nos mostrara los mensajes como en la **Figura 20**.

```
/opt/ros/indigo/share/kobuki_node/launch/minimal.launch http://localhost:11311
* /rosversion: 1.11.21
NODES
 /
  diagnostic_aggregator (diagnostic_aggregator/aggregator_node)
  mobile_base (nodelet/nodelet)
  mobile_base_nodelet_manager (nodelet/nodelet)
ROS_MASTER_URI=http://localhost:11311
core service [/rosout] found
process[mobile_base_nodelet_manager-1]: started with pid [8048]
process[mobile_base-2]: started with pid [8049]
process[diagnostic_aggregator-3]: started with pid [8050]
[ INFO] [1501481604.505223244]: Loading nodelet /mobile_base of type kobuki_node
/KobukiNodelet to manager mobile_base_nodelet_manager with the following remappings:
[ INFO] [1501481604.505302444]: /mobile_base/joint_states -> /joint_states
[ INFO] [1501481604.505337155]: /mobile_base/odom -> /odom
[ INFO] [1501481604.510307899]: waitForService: Service [/mobile_base_nodelet_manager/load_nodelet] has not been advertised, waiting...
[ INFO] [1501481604.563132773]: Initializing nodelet with 8 worker threads.
[ INFO] [1501481604.574116921]: waitForService: Service [/mobile_base_nodelet_manager/load_nodelet] is now available.
[ WARN] [1501481604.698692190]: Kobuki : no robot description given on the parameter server
[ INFO] [1501481604.698873358]: Kobuki : configured for connection on device_port /dev/kobuki [/mobile_base].
[ INFO] [1501481604.698917219]: Kobuki : driver running in normal (non-simulation) mode [/mobile_base].
[ INFO] [1501481604.700570781]: Kobuki : Velocity commands timeout: 0.600000000 seconds [/mobile_base].
[ INFO] [1501481604.701856140]: Kobuki : using odom_frame [odom][/mobile_base].
[ INFO] [1501481604.702586400]: Kobuki : using base_frame [base_footprint][mobile_base].
[ INFO] [1501481604.703315473]: Kobuki : publishing transforms [/mobile_base].
[ INFO] [1501481604.703869105]: Kobuki : using imu data for heading [/mobile_base].
[ WARN] [1501481604.956713339]: Kobuki : no data stream, is kobuki turned on?
[ INFO] [1501481604.957018127]: Kobuki : initialised.
```

Figura 20. Comunicación entre el Ordenador y Kobuki [19]

4.- A continuación se abre otro terminal y se ejecuta el código

```
roslaunch kobuki_keyop keyop.launch
```

Permite la teleoperación de la plataforma robótica Kobuki. El terminal indica cuales son los comandos de movimiento de éste, como lo indica la **Figura 21**.

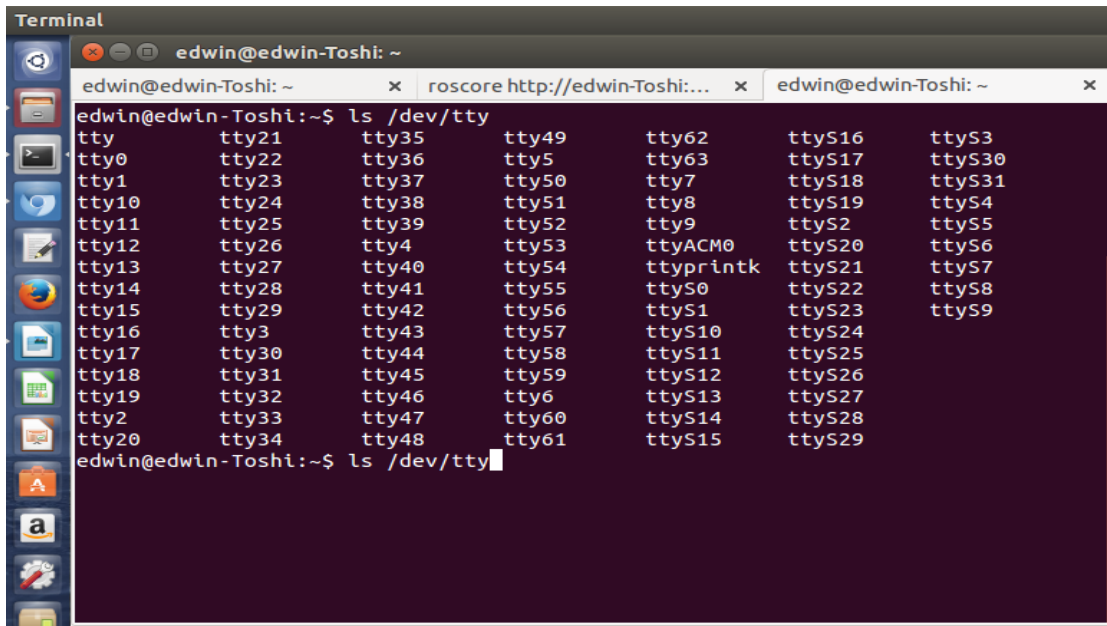
```
/opt/ros/indigo/share/kobuki_keyop/launch/keyop.launch http://localhost:11311
/opt/ros/indigo/share/kobuki_node/launch/minimal.launch http://localhost:11311 x /opt/ros/indigo
Done checking log file disk usage. Usage is <1GB.
started roslaunch server http://edwin-Toshi:43518/
SUMMARY
-----
PARAMETERS
* /keyop/angular_vel_max: 6.6
* /keyop/angular_vel_step: 0.33
* /keyop/linear_vel_max: 1.5
* /keyop/linear_vel_step: 0.05
* /keyop/wait_for_connection_: True
* /roscdistro: indigo
* /rosversion: 1.11.21
NODES
 /
  keyop (kobuki_keyop/keyop)
ROS_MASTER_URI=http://localhost:11311
core service [/roscout] found
process[keyop-1]: started with pid [8367]
[ INFO] [1501481642.958859214]: KeyOpCore : using linear vel step [0.05].
[ INFO] [1501481642.958973963]: KeyOpCore : using linear vel max [1.5].
[ INFO] [1501481642.959035493]: KeyOpCore : using angular vel step [0.33].
[ INFO] [1501481642.959095696]: KeyOpCore : using angular vel max [6.6].
[ WARN] [1501481642.966426936]: KeyOp: could not connect, trying again after 500ms...
[ INFO] [1501481643.466717790]: KeyOp: connected.
Reading from keyboard
-----
Forward/back arrows : linear velocity incr/decr.
Right/left arrows : angular velocity incr/decr.
Spacebar : reset linear/angular velocities.
d : disable motors.
e : enable motors.
q : quit.
```

Figura 21. Teleoperación del Robot Kobuki [19]

5.- Utilizando el comando `ls /dev/tty` y pulsando dos veces la tecla `tab` nos imprime todos los puertos que se están utilizando. Se puede verificar si está conectado el Arduino con ordenador.

Como se muestra en la **Figura 22**.

```
ls /dev/tty
```

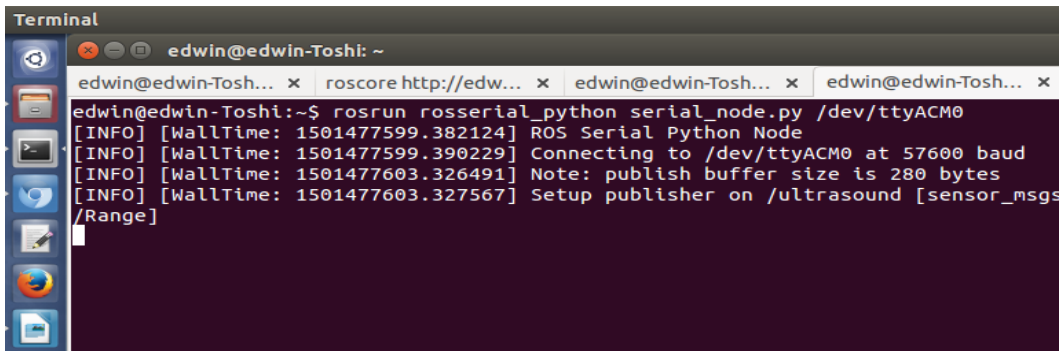


**Figura 22. Verificación de puertos activos [19]**

6.- A continuación se ejecuta el comando:

```
rosrun roserial_python serial_node.py / dev / ttyUSB0
```

Este comando ejecuta la aplicación cliente roserial que reenvía sus mensajes desde el Arduino al resto de (ROS). Asegúrese de utilizar el puerto serie correcto para nuestro caso será /ttyACM0. Como se muestra la **Figura 23**.

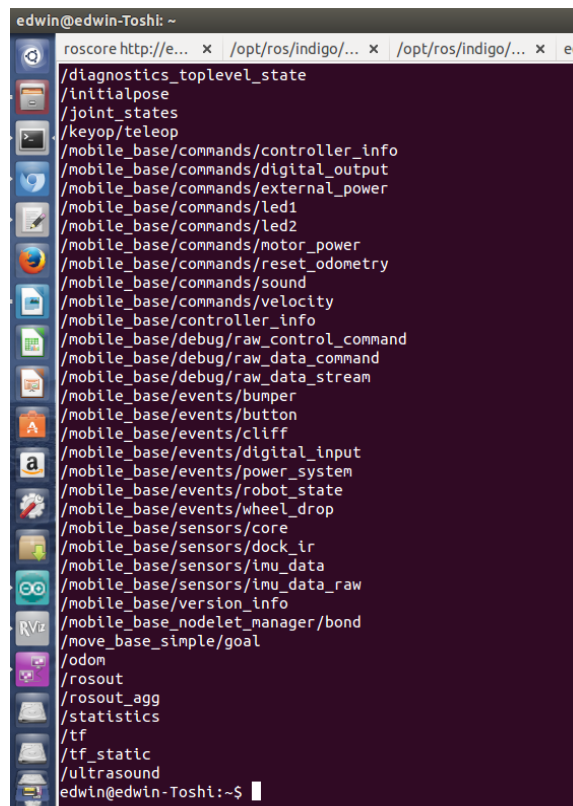


**Figura 23. Activación del Puerto Serial [19]**

7.- A continuación se ejecuta el comando:

```
rostopic list
```

Este comando nos permite imprimir una lista de los temas a los que se puede suscribir como se muestra en la **Figura 24**.



**Figura 24. Lista de Temas a los que se Puede Suscribir [19]**

8.- Ejecutando el siguiente código

```
rostopic echo /ultrasound
```

Imprime la información del tema ultrasónico que para el caso es el sensor Ultrasonico hacia el ordenador por el puerto serial, que se visualiza en el terminal. Se muestra en la **Figura 25** los datos del sensor.



```
edwin@edwin-Toshi: ~
roscore http://e... x /opt/ros/indigo/... x
range: 1.81168377399
---
header:
  seq: 38041
  stamp:
    secs: 1501481452
    nsecs: 504912925
  frame_id: /ultrasound
  radiation_type: 0
  field_of_view: 0.10000000149
  min_range: 0.8
  max_range: 4.8
  range: 1.8003436327
---
header:
  seq: 38042
  stamp:
    secs: 1501481452
    nsecs: 564912925
  frame_id: /ultrasound
  radiation_type: 0
  field_of_view: 0.10000000149
  min_range: 0.8
  max_range: 4.8
  range: 1.81683850288
---
header:
  seq: 38043
  stamp:
    secs: 1501481452
    nsecs: 623912925
  frame_id: /ultrasound
  radiation_type: 0
  field_of_view: 0.10000000149
  min_range: 0.8
  max_range: 4.8
  range: 1.81185567379
```

Figura 25. Datos Obtenidos del Sensor Ultrasónico [19]

9.- Para visualizar de una manera gráfica se lo puede hacer ejecutando el siguiente código.

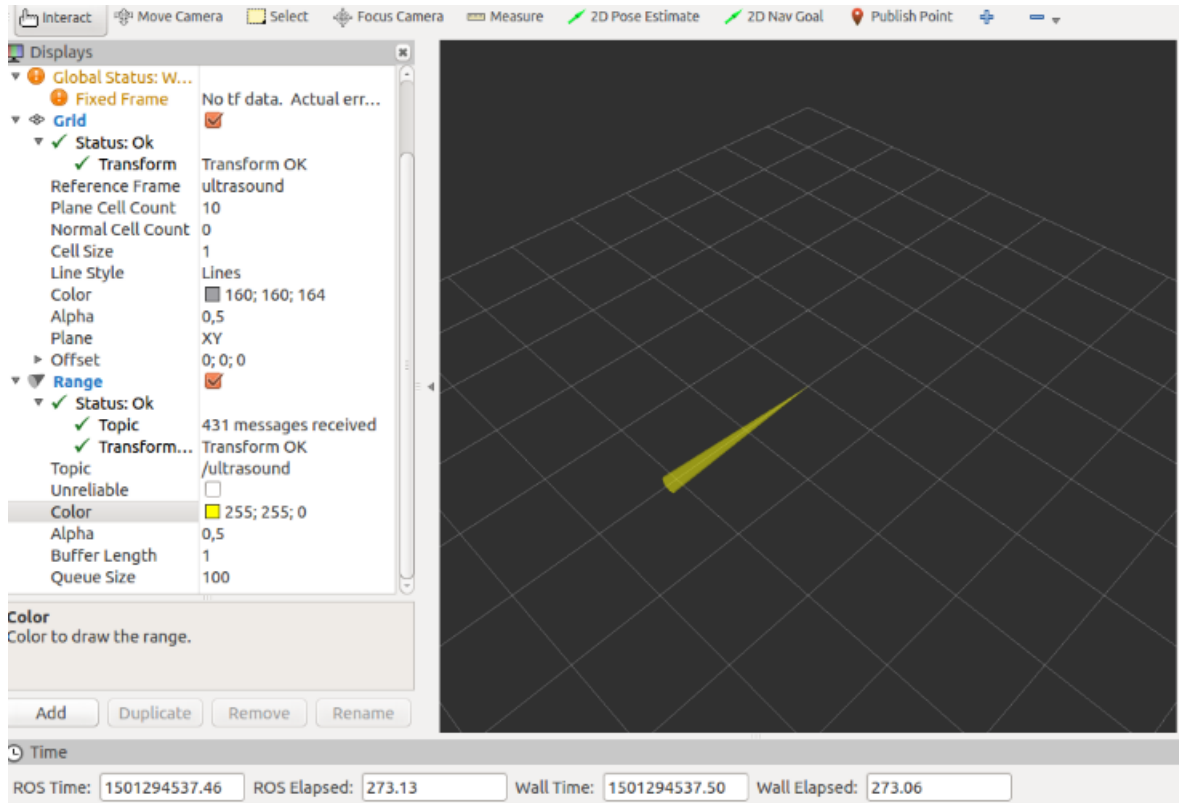
Se lo muestra en la **Figura 26**.

```
roslaunch rviz rviz
```

```
Terminal
edwin@edwin-Toshi: ~
edwin@edwi... x roscore http://e... x edwin@edwi... x edwin@edwi... x edwin@edwi... x
edwin@edwin-Toshi:~$ roslaunch rviz rviz
[ INFO] [1501477696.574135319]: rviz version 1.11.16
[ INFO] [1501477696.574231281]: compiled against Qt version 4.8.6
[ INFO] [1501477696.574263897]: compiled against OGRE version 1.8.1 (Byatis)
[ INFO] [1501477697.268547856]: Stereo is NOT SUPPORTED
[ INFO] [1501477697.268773443]: OpenGL version: 3 (GLSL 1.3).
[ WARN] [150147765.097726862]: MessageFilter [target=map ]: Dropped 100,00% of
messages so far. Please turn the [ros.rviz.message_notifier] rosconsole logger t
o DEBUG for more information.
```

Figura 26. Preparado para ejecutar Rviz [19]

Rviz es un visualizador 3D para mostrar datos de sensores e información de estado desde ROS. Usando rviz, puede visualizar la configuración actual del Kobuki en un modelo virtual del robot. También puede mostrar representaciones en vivo de los valores de los sensores sobre los temas ROS, incluidos los datos de la cámara, las mediciones de distancia infrarroja, los datos de sonar y más. La **Figura 27** está demostrando gráficamente la distancia.



**Figura 27. Recepción de datos desde el sensor ultrasónico en tiempo real [19]**

## CAPÍTULO V

### 5. Análisis y Resultados

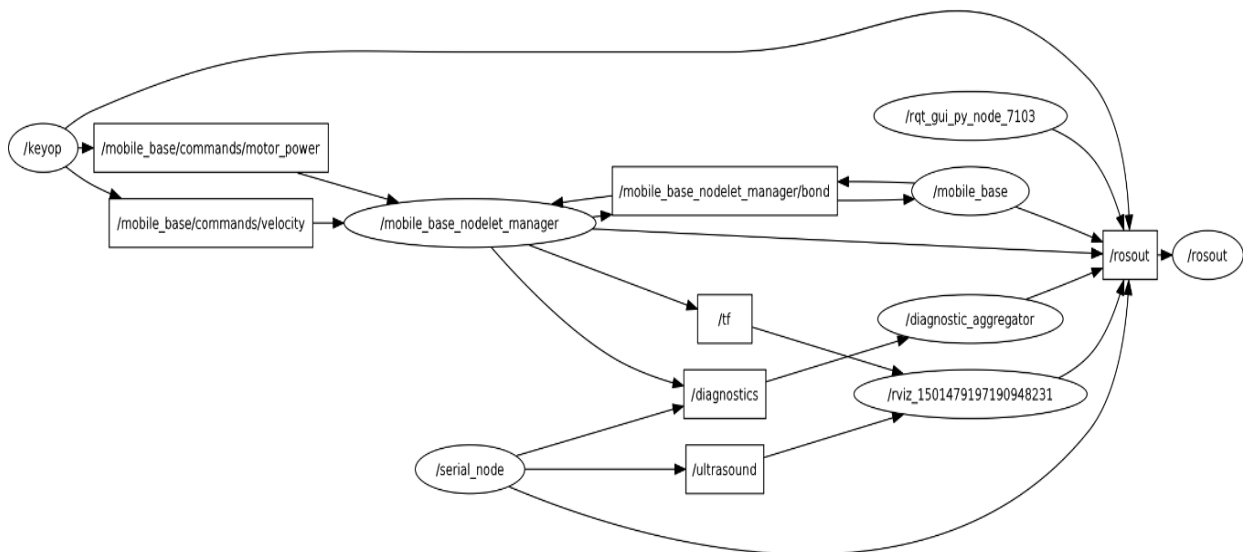
En el siguiente capítulo se detallan las pruebas realizadas para la verificación del buen funcionamiento del proyecto además el análisis y resultados respectivo de cada prueba.

#### 5.1 Prueba de conexión de los nodos en ejecución

En este capítulo se especifica los resultados obtenidos, Para este último capítulo se empleará una herramienta de ROS que facilite el reconocimiento de nodos y tópicos que se están ejecutando en ese momento. Se lo hace mediante el código

```
rqt_graph
```

Esta aplicación se ejecuta directamente, sin necesidad de llamar a rosrund. Visualiza los nodos que se están ejecutando en este momento y el paso de tópicos entre ellos. Sirve Para visualizar qué está pasando con nuestro programa como se puede evidenciar en la **Figura 28**.



**Figura 28. Diagrama de nodos en ejecución [19]**

El resultado de emplear el comando `rqt_graph` es una gráfica con todos los nodos en funcionamiento. Se puede apreciar el nodo que fue creado para la comunicación del sensor ultrasónico con el microcontrolador Arduino y el ordenador está en completa funcionalidad.

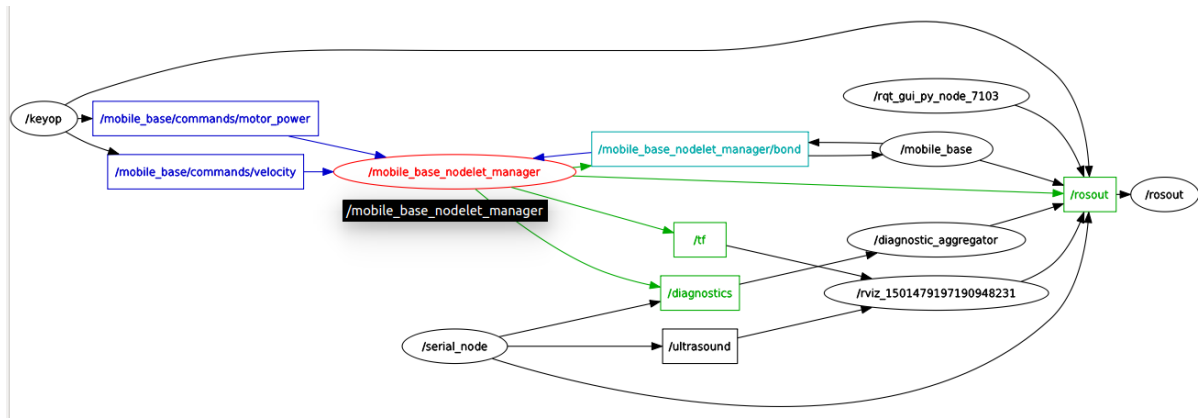
### **5.1.1 Pruebas de conexión del nodo Kobuki**

Éste se encuentra unido a los comandos de energía del motor, velocidad y al administrador, gestor de la base móvil que son de información de entrada.

Los nodos de salida al que está unido son Tf, éste es un paquete que permite al usuario realizar un seguimiento de múltiples tramas de coordenadas a lo largo del tiempo. Tf mantiene la relación entre los fotogramas de coordenadas en una estructura de árbol almacenada en el tiempo, y permite al usuario transformar puntos, vectores, etc. entre dos marcos de coordenadas en cualquier punto deseado en el tiempo.

Además está unido al sistema de diagnóstico; está diseñado para recopilar información de controladores de hardware y hardware de robot para usuarios y operadores para análisis, solución de problemas y registro. La pila de diagnósticos contiene herramientas para recopilar, publicar, analizar y visualizar datos de diagnóstico.

La cadena de herramientas de diagnóstico se construye alrededor del tema `/diagnostics`. En este tema, los controladores y dispositivos de hardware publican el mensaje `diagnostic_msgs/DiagnosticStatus` con los nombres de los dispositivos, el estado y los puntos de datos específicos.



**Figura 29. Interacción del Nodo Kobuki con los Demás Nodos [19]**

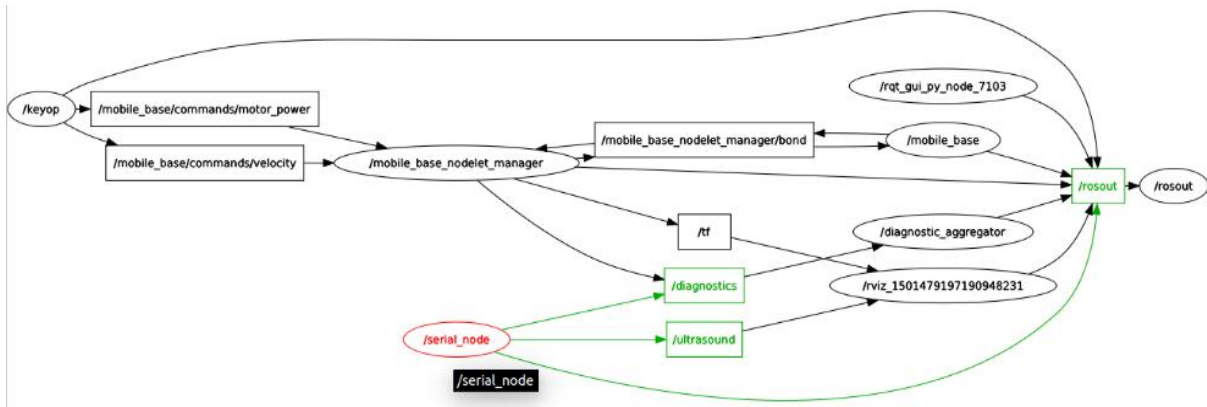
Al dividir los nodos de la plataforma robótica móvil se puede evidenciar la comunicación que hay entre sí. Como se pudo evidenciar en la **Figura 29**.

### 5.1.2 Pruebas de Conexión del Nodo Rosserial

El nodo se encarga de la comunicación del tiempo real con el ordenador en este caso de microcontrolador Arduino y el ordenador para ello primero debe pasar por el sistema diagnóstico y el de ultrasonido.

El sistema diagnóstico como ya se lo ha mencionado antes está diseñado para recopilar información de los controladores del hardware de robot que será utilizado por usuarios y operadores para análisis, solución de problemas y registro.

El sistema ultrasónico es donde se encuentra el programa corriendo del microcontrolador Arduino a su vez este se conecta con el sistema del Rviz para visualizar la medición.



**Figura 30. Comunicación del Nodo Serial con los Demás Nodos [19]**

El nodo serial está trabajando como se lo tenía previsto obteniendo los datos del sensor ultrasónico y enviando los datos a los suscriptores en tiempo real.

## 5.2 Pruebas de Respuesta del Sensor Ultrasónico Hacia un Objeto (pared)

Se realizó pruebas de confiabilidad de los datos adquiridos desde el sensor ultrasónico con diferentes tiempos de muestreo y a diferentes distancias.

### 5.2.1 Tiempo de Muestreo en $t=1s$ y Distancia Física de 0.50m



**Figura 31. Medición de Datos a una Distancia de 0.50m [19]**

En la Tabla 4 se registra los datos obtenidos del sensor ultrasónico en tiempo real.

**Tabla4. Datos de Sensor Ultrasónico t=1s y d=0.50m [29]**

<b>Tiempo de muestro (t=1s)</b>	<b>Distancia obtenida del sensor ultrasónico (d=m)</b>
1	<b>0.501</b>
2	<b>0.502</b>
3	<b>0.498</b>
4	<b>0.503</b>
5	<b>0.503</b>
6	<b>0.502</b>
7	<b>0.503</b>
8	<b>0.503</b>
9	<b>0.503</b>
10	<b>0.502</b>

**Promedio= 0.502m**

**Margen de error= promedio – distancia física.**

**Margen de error = 0.502 - 0.50**

**Margen de error = 0.002m**

Existe fiabilidad de los datos adquiridos ya que el margen de error es de 0.002m este no representa un riesgo para el proyecto y está dentro del rango aceptable que va desde 0m a 0.005m

### 5.2.2 Tiempo de Muestreo en $t=1s$ y Distancia Física de 0.70m



Figura 32. Medición de Datos a una Distancia de 0.70m[19]

Tabla5. Datos de Sensor Ultrasónico  $t=1s$  y  $d=0.70m$  [29]

Tiempo de muestro ( $t=1s$ )	Distancia obtenida del sensor ultrasónico ( $d=m$ )
1	0.702
2	0.701
3	0.706
4	0.702
5	0.702
6	0.703
7	0.705
8	0.703
9	0.702
10	0.702



**Promedio= 0.7028m**

**Margen de error= promedio (m) – distancia física (m)**

**Margen de error = 0.7028 - 0.70**

**Margen de error = 0.0028m**

Existe fiabilidad de los datos adquiridos ya que el margen de error es de 0.0028m este no representa un riesgo para el proyecto y está dentro del rango aceptable que va desde 0m a 0.005m

### **5.2.3 Tiempo de Muestreo en t=2s y Distancia Física de 0.70m**

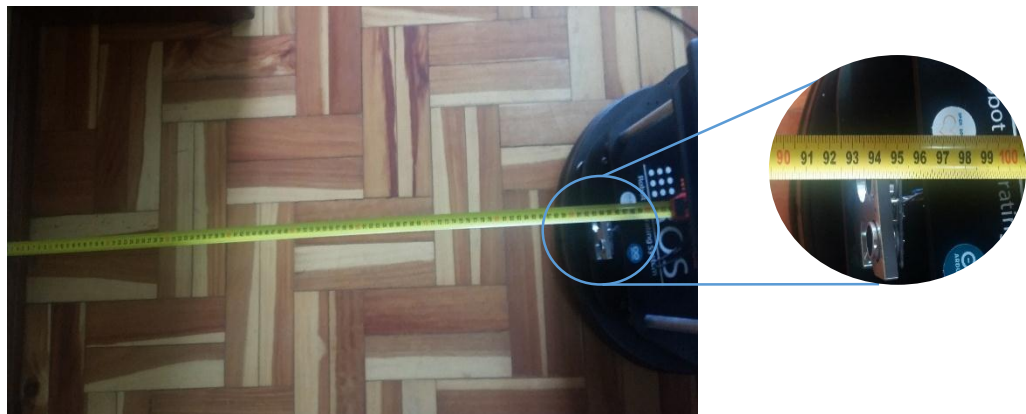
Para esta parte de la prueba se varió el tiempo de muestreo y se mantuvo la distancia para comparar con los resultados entre esta prueba y la anterior y así llegar a elegir el mejor tiempo de muestreo.

**Tabla 6. Datos de Sensor Ultrasónico t=2s y d=0.70m [29]**

<b>Tiempo de muestro (t=2s)</b>	<b>Distancia obtenida del sensor ultrasónico (d=m)</b>
1	<b>0.705</b>
2	<b>0.706</b>
3	<b>0.705</b>
4	<b>0.703</b>
5	<b>0.705</b>
6	<b>0.706</b>
7	<b>0.702</b>

8	0.702
9	0.702
10	0.703

**5.2.4 Tiempo de Muestreo en t=1s y Distancia Física de 0.94m**



**Figura 33. Medición de Datos a una Distancia de 0.94m [19]**

**Tabla 7. Datos de Sensor Ultrasónico t=1s y d=0.94m [29]**

<b>Tiempo de muestro (t=1s)</b>	<b>Distancia obtenida del sensor ultrasónico (d=m)</b>
1	0.944
2	0.945
3	0.944
4	0.941
5	0.940
6	0.941
7	0.944

8	<b>0.941</b>
9	<b>0.940</b>
10	<b>0.944</b>

**Promedio**= 0.9424m

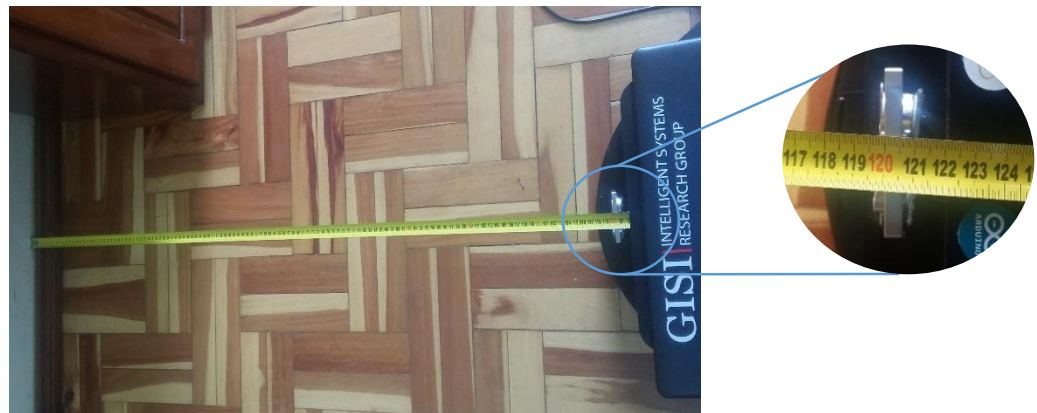
**Margen de error**= promedio (m) – distancia física (m)

**Margen de error** = 0.9424 - 0.70

**Margen de error** = 0.0024m

Existe fiabilidad de los datos adquiridos ya que el margen de error es de 0.0024m este no representa un riesgo para el proyecto y está dentro del rango aceptable que va desde 0m a 0.005m

### 5.2.5 Tiempo de Muestreo en t=1s y Distancia Física de 1.20m



**Figura 34. Medición de Datos a una Distancia de 1.20m [19]**

**Tabla 8. Datos de Sensor Ultrasónico t=1s y d=1.20m [29]**

<b>Tiempo de muestro (t=1s)</b>	<b>Distancia obtenida del sensor ultrasónico (d=m)</b>
1	<b>1.185</b>
2	<b>1.169</b>
3	<b>1.192</b>
4	<b>1.169</b>
5	<b>1.183</b>
6	<b>1.209</b>
7	<b>1.187</b>
8	<b>1.196</b>
9	<b>1.178</b>
10	<b>1.165</b>

**Promedio= 1.1833m**

**Margen de error= promedio (m) – distancia física (m)**

**Margen de error = 1.1833- 1.20**

**Margen de error = -0,0167m**

Se puede notar que entre mayor sea la distancia a medir los valores del sensor tiene mayor margen de error entre los datos adquiridos.

Mientras el margen de errores se encuentre entre los valores de 0 a 0.005 los valores son confiables. Además el tiempo de muestreo más óptimo es de 1 segundo ya que la obtención de datos es más estables.

Para que el tele-operador pueda reaccionar y evitar una colisión con los objetos necesita entre 0.50m a 1m de distancia entre la plataforma robótica y el obstáculo.

## **CONCLUSIONES Y RECOMENDACIONES**

En este capítulo se presenta las condiciones y recomendaciones en base a las pruebas realizadas y a los resultados obtenidos en el desarrollo de este proyecto de titulación.

### **CONCLUSIONES**

Las pruebas realizadas demostraron que está funcionando según lo previsto. Es posible enviar y recibir datos entre una plataforma embebida de tiempo real, en tiempo real y la plataforma robótica Kobuki. Resolviendo así el problema planteado en el anteproyecto.

El Sistema Operativos de Robots (ROS) al ser un framework utilizando para el desarrollo de aplicaciones presta muchas ventajas, la principal es la abstracción de hardware que permite que los algoritmos implementados en este proyecto pueden ser ejecutados no solo con Kobuki sino también con otros sistemas robóticos de diferentes fabricantes.

La plataforma robótica móvil Kobuki tiene la ventaja de ser un robot omnidireccional que permite realizar movimientos de desplazamiento y de rotación facilitando la implementación de algoritmos para la teleoperación.

La comunicación en múltiples maquinas permite exportar la información a una instancia remota al igual que interactuar con el servidor local, pudiendo hacer monitoreo, control o simplemente tareas de almacenaje. La principal ventaja es poder sumar capacidad de cómputo a la experiencia que se esté llevando a cabo.

La configuración del sistema se basó en tres segmentos fundamentales que son: sistema de plataforma robótica Kobuki, Sistema de control, Sistema de tiempo real. Los cuales trabajando a la par se pudo resolver el problema planteado.

La implementación del software con el hardware se la realizó rápidamente gracias a la facilidades que otorgo la plataforma robótica Kobuki.

## **RECOMENDACIONES**

Se recomienda andes de efectuar cualquier manipulación a la plataforma robótica móvil Kobuki se debe leer con detenimiento el manual de operación con el fin de dar un buen funcionamiento de la plataforma y evitar daños que puedan ser irreversibles.

Antes de ejecutar cualquier aplicación es recomendable que la batería del robot Kobuki se encuentre totalmente carga con la finalidad de evitar la pérdida de conectividad entre la PC o el módulo de tiempo real y así poder garantizar que las operaciones del usuario se realizaran sin inconvenientes.

Se debe estimular al estudiante al manejo de la plataforma robótica móvil para la investigación y el aprendizaje gracias a sus diversas funcionalidades que tiene un alto grado de aplicabilidad en el campo de la robótica.

Al iniciar la carga del programa en el microcontrolador Arduino se debe cerciorar en escoger la placa Arduino con la que se está trabajando en el software de la aplicación, de la misma manera se debe verificar el puerto en el que se está conectando ya que de no hacerlo emitirá un error de carga.

Se debe estar pendiente de las actualizaciones y adiciones que se presentan en el Sistema Operativo de Robots (ROS), ya que se encuentra en continuo crecimiento y mejoramiento.

## **ANEXOS**



## Ensamblaje del Sistema de Tiempo Real a la Plataforma Robótica Kobuki



## Componentes Principales del Robot Kobuki



Placa Principal



Actuador (motor-sensor-rueda)



Rueda Estabilizadora



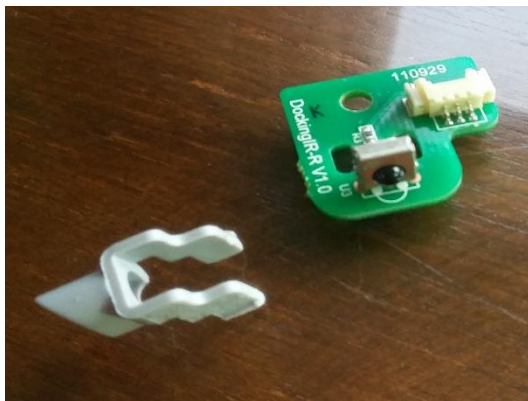
Panel de Conexión y Control



**Sensor de Impacto**



**Sensor IR Inferior**



**Sensor IR Frontal**



**Carcasa Superior**



**Parchoques**



**Carcasa Inferior**

## BIBLIOGRAFÍA

- [1] A. O. Baturone, Robótica: manipuladores y robots móviles. Marcombo, 2005.
- [2] A. Koubaa, Robot Operating System (ROS): The Complete Reference. Springer, 2017.
- [3] «ROS.org | Acerca de ROS». .
- [4] Ò. V. Oliver, Palabra de robot: Inteligencia artificial y comunicación. Universitat de València, 2006.
- [5] J. A. S. Sánchez, Avances en robótica y visión por ordenador. Univ de Castilla La Mancha, 2002.
- [6] Jackie Kay, «Introducción a los sistemas en tiempo real». [En línea]. Disponible en: [http://design.ros2.org/articles/realtime\\_background.html](http://design.ros2.org/articles/realtime_background.html). [Accedido: 05-jul-2017].
- [7] L. M. J. García y R. P. Manchón, Sistemas Informáticos en Tiempo Real: Teoría y Aplicaciones. Universidad Miguel Hernández, 2017.
- [8] W. A. Halang y K. M. Sacha, Real-time Systems: Implementation of Industrial Computerised Process Automation. World Scientific, 1992.
- [9] Manuel Ortiz, «SISTEMAS INFORMÁTICOS EN TIEMPO REAL».
- [10] «Introduction to Real-time Systems». [En línea]. Disponible en: [http://design.ros2.org/articles/realtime\\_background.html](http://design.ros2.org/articles/realtime_background.html). [Accedido: 17-sep-2017].
- [11] «Conceptos básicos en los Sistemas de Tiempo Real».
- [12] G. Zabala, Robotica. USERSHOP, 2007.
- [13] M. H. Ordoñez, M. B. O. Moctezuma, C. A. C. Arriaga, y J. C. R. Portillo, Robótica: Análisis, modelado, control e implementación. OmniaScience, 2015.
- [14] «ROS/Introduction - ROS Wiki». [En línea]. Disponible en: <http://wiki.ros.org/ROS/Introduction>. [Accedido: 16-sep-2017].

- [15] «Conceptos | Erle Robotics Gitbook». [En línea]. Disponible en: <https://erlerobotics.gitbooks.io/erlerobot/es/ros/ROS-concepts.html>. [Accedido: 17-sep-2017].
- [16] «Sensor de Distancia de Ultrasonido HC-SR04», Electronilab. .
- [17] «Arduino - Introduction». [En línea]. Disponible en: <https://www.arduino.cc/en/Guide/Introduction>. [Accedido: 17-sep-2017].
- [18] F. Doutel, «Guía del Arduinomaníaco: todo lo que necesitas saber sobre Arduino», Xataka, 18-ago-2015. [En línea]. Disponible en: <https://www.xataka.com/especiales/guia-del-arduinomaniaco-todo-lo-que-necesitas-saber-sobre-arduino>. [Accedido: 17-sep-2017].
- [19] Edwin Lascano, «FIGURAS REALIZADAS POR EL AUTOR DE LA TESIS.», Universidad Técnica del Norte, Ibarra.
- [20] Edwin Lascano, «Tablas diseñadas por el autor de la tesis».
- [21] «Online User Guide – KOBUKI». .
- [22] Edwin Lascano, «Tabla diseñada por el autor de la tesis1».
- [23] «The leading operating system for PCs, IoT devices, servers and the cloud | Ubuntu». [En línea]. Disponible en: <https://www.ubuntu.com/>. [Accedido: 16-sep-2017].
- [24] «roserial». [En línea]. Disponible en: <http://library.isr.ist.utl.pt/docs/roswiki/roserial.html>. [Accedido: 13-sep-2017].
- [25] «Nodelet - ROS Wiki». [En línea]. Disponible en: <http://wiki.ros.org/nodelet>. [Accedido: 02-ago-2017].
- [26] «diagnostic\_aggregator - ROS Wiki». [En línea]. Disponible en: [http://wiki.ros.org/diagnostic\\_aggregator](http://wiki.ros.org/diagnostic_aggregator). [Accedido: 02-ago-2017].
- [27] «rosout - ROS Wiki». [En línea]. Disponible en: <http://wiki.ros.org/rosout>. [Accedido: 17-sep-2017].

[28] «rosserial\_arduino/Tutorials/Arduino IDE Setup - ROS Wiki». [En línea]. Disponible en: [http://wiki.ros.org/rosserial\\_arduino/Tutorials/Arduino%20IDE%20Setup](http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup). [Accedido: 17-sep-2017].

[29] Edwin Lascano, «Tabla relizada por el autor de tesis datos adquiridos de las pruebas.», Universidad Técnica del Norte, Ibarra.