INSTITUTO TECNOLÓGICO SUPERIOR DE ATLIXCO

ORGANISMO PÚBLICO DESCENTRALIZADO DEL GOBIERNO DEL ESTADO DE PUEBLA

INGENIERÍA MECATRÓNICA

IMPLEMENTACIÓN DE LA PRIMERA VERSIÓN DE LA ESTIMACIÓN DE LA FUENTE SONORA, CON LA PLACA ARDUINO Y EL SENSOR SONIDO FC-04.

Reporte final del Proyecto de Residencia Profesional.

Presenta:
C. (José Ángel Balbuena Palma).

Asesores:

Dr. Eduardo Morales Manzanares

Dra. Mariana Natalia Ibarra Bonilla

Atlixco, Pue. Junio 2019.

INSTITUTO TECNOLÓGICO SUPERIOR DE ATLIXCO

ORGANISMO PÚBLICO DESCENTRALIZADO DEL GOBIERNO DEL ESTADO DE PUEBLA

INGENIERÍA MECATRÓNICA

IMPLEMENTACIÓN DE LA PRIMERA VERSIÓN DE LA ESTIMACIÓN DE LA FUENTE SONORA, CON LA PLACA ARDUINO Y EL SENSOR SONIDO FC-04.

Reporte de actividades realizadas en la empresa INSTITUTO NACIONAL DE ASTROFÍSICA, ÓPTICA Y ELECTRÓNICA durante el periodo comprendido del 05 de FEBRERO hasta 05 de JUNIO del 2019 para su acreditación.

Presenta:
C. (José Ángel Balbuena Palma).

Asesores:
Dr. Eduardo Morales Manzanares
Dra. Mariana Natalia Ibarra Bonilla

Atlixco, Pue. Junio 2019.

Agradecimientos

Doy gracias a Dios por la vida que brindo. A mi madre Patricia por ser un pilar en mi vida, a mis hermanos Jesús y Monserrat por siempre apoyarme. Al Dr. Eduardo Morales Manzanares por la oportunidad brindada para realizar mis prácticas profesionales. Al M.C. David Carrillo López por su apoyo en las pruebas y experimentos realizados en este proyecto y a la Dra. Mariana Natalia Ibarra Bonilla por sus asesorías brindadas en el desarrollo del proyecto.

Resumen.

El trabajo realizado trata de la implementación de un sistema para la estimación de la dirección de una fuente de sonido, por medio de la diferencia del tiempo de arribo, empleando una tarjeta de desarrollo Arduino Uno, la cual es muy utilizada en proyectos de electrónica y el sensor de sonido FC-04 un sensor económico y fácil de adquirir, también se realizó la integración de esta implementación al robot de servicio Markovito del laboratorio de robótica del Instituto Nacional de Astrofísica, Óptica y Electrónica, por medio de ROS.

El actual sistema implementado solo es capaz de determinar la dirección de una fuente de sonido en cuatro posibles direcciones frente, atrás, derecha e izquierda, por lo que el objetivo de este proyecto es ampliar la resolución en la estimación de la dirección de la fuente sonora. Para el posterior uso en otras tareas del robot, como girar hacia la fuente de sonido, ya que una correcta estimación de la dirección de una fuente de sonido es muy importante para una adecuada orientación del robot.

El desarrollo de este proyecto se realizó en cuatro fases que se describen a continuación:

1) Análisis el sistema ya implementado para la estimación de la dirección de la fuente sonora, para hacer un reconocimiento de los recursos disponemos para llevar a cabo el proyecto.
2) Investigación y selección de un nuevo método para establecer el origen de la fuente del sonido, quedando como ganador el método de la diferencia del tiempo de arribo (TDOA).
3) Adaptación e implementación del método de la diferencia del tiempo de arribo, generando un nodo de ROS.
4) Realización de pruebas y depuración de los errores presentados en la implementación.

Los resultados que se obtuvieron de la integración del nodo pararealizar la estimación de la dirección de la fuente sonora y su empleo en la tarea del robot Hear and Rotate se presentaron dos variantes. El giro era exacto en dirección de

la fuente de sonido, por lo que le error en la estimación de la dirección de la fuente de sonido era muy pequeño. El giro era inexacto en dirección a la fuente de sonido apareciendo un el error de estimación de hasta $\pi/2$ rad esto debido a problemas con los sensores de sonido específicamente el umbral que se establece para detectar un cierto nivel de sonido este problema es analizado con un mayor detalle en la metodología del proyecto. Para poder evitar este error se propuso como solución para trabajos futuros la implementación de circuitos de ajuste automático de ganancia en cada uno de los micrófonos y así disminuir el error generado por el umbral de sonido.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN.

El presente proyecto trata de la estimación de la dirección de una fuente sonora, con el uso de la placa de desarrollo Arduino Uno y el sensor de sonido Fc-04. Para integración en el robot de servicio Markovito de la coordinación de ciencias computacionales del Instituto Nacional de Astrofísica, Óptica y Electrónica.

El motivo de la realización de este trabajo es obtener una mayor resolución en la estimación de la dirección de la fuente sonora, que el sistema ya existente. Para después emplear esta estimación por otros nodos en el desarrollo de alguna tarea específica, como girar el robot hacia la fuente de sonido. Ya que una adecuada estimación del origen del sonido es crucial para una realizar una orientación más exacta hacia la fuente del sonido.

En el trabajo realizamos una propuesta metodología orientada a la investigación, implementación y experimentación, la cual se desarrolló de la siguiente forma:

- Primero se analizó el sistema ya implementado para la detección de la dirección de la fuente sonora, y obtener una idea de los recursos disponemos para llevar a cabo el proyecto.
- Segundo la investigación y selección de un nuevo método para establecer el origen de la fuente del sonido, seleccionando el método de la diferencia del tiempo de arribo (TDOA).
- Tercero adaptación e implementación del método de la diferencia del tiempo de arribo.
- Cuarto la realización de pruebas y depuración de los errores presentados en la implementación del método seleccionado.

Capítulo 1
Marco Contextual.

En este capítulo se dará a conocer una concisa descripción del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE). Así mismo, se mostrará una breve reseña de la Coordinación de Ciencias Computacionales del INAOE, específicamente de la línea de investigación de Robótica.

1.1 Historia del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE).

El Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) fue creado por decreto presidencial el 11 de noviembre de 1971 como un organismo descentralizado, de interés público, con personalidad jurídica y patrimonio propio, ubicado en Tonantzintla, Puebla, con los siguientes objetivos:

- Preparar investigadores, profesores especializados, expertos y técnicos en astrofísica, óptica y electrónica.
- Procurar la solución de problemas científicos y tecnológicos relacionados con las citadas disciplinas.
- Orientar sus actividades de investigación y docencia hacia la superación de las condiciones y resolución de los problemas del país.
- Con este decreto el INAOE tiene la facultad de impartir cursos y otorgar grados de maestría y doctorado en las diversas disciplinas que en él se desarrollan.

EL INAOE es heredero de una gran tradición científica que data de 1942, cuando Luis Enrique Erro fundó el Observatorio Astrofísico Nacional de Tonantzintla. En aquel entonces, Tonantzintla se escogió como el lugar idóneo para la instalación del Observatorio, el cual cumplía con las exigentes normas de calidad como noches despejadas y en cantidad por año, así como altura geográfica.

Erro fue sustituido en la dirección del Observatorio por el doctor Guillermo Haro, bajo cuya dirección se convirtió en uno de los centros más importantes de América Latina por la calidad del trabajo científico que en él se llevaba a cabo. El mismo Haro se dio

cuenta de la importancia para el país de la óptica y la electrónica, por lo que en 1971 decidió fundar el INAOE. (Ver Figura 1).

Figura 1: Logotipo del INAOE.

En 1972 se fundó el Departamento de Óptica, y dos años después inició sus actividades el Departamento de Electrónica. Desde su creación uno de los principales objetivos del INAOE ha sido la preparación de investigadores jóvenes, capaces de identificar y resolver problemas científicos y tecnológicos en astrofísica, óptica, electrónica y áreas afines. En 1972 se iniciaron los estudios de maestría en Óptica y en 1974 los de Electrónica. En 1984 se inició el programa de doctorado en Óptica, y en 1993 los programas de doctorado en Electrónica; así como la maestría y doctorado en Astrofísica. Finalmente, en agosto de 1998 se inició el programa de maestría y doctorado en Ciencias Computacionales. Siendo hoy en día uno de los centros de investigación más importantes a nivel mundial. (Ver Figura 2).

Figura 2: Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Tonantzintla, Puebla, México.

1.2 Ubicación del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE).

La ubicación del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) es Luis Enrique Erro 1, Sta. María Tonantzintla, 72840 San Andrés Cholula, Puebla, México con Tel. 01 222 266 3100. (Ver Figura 3).

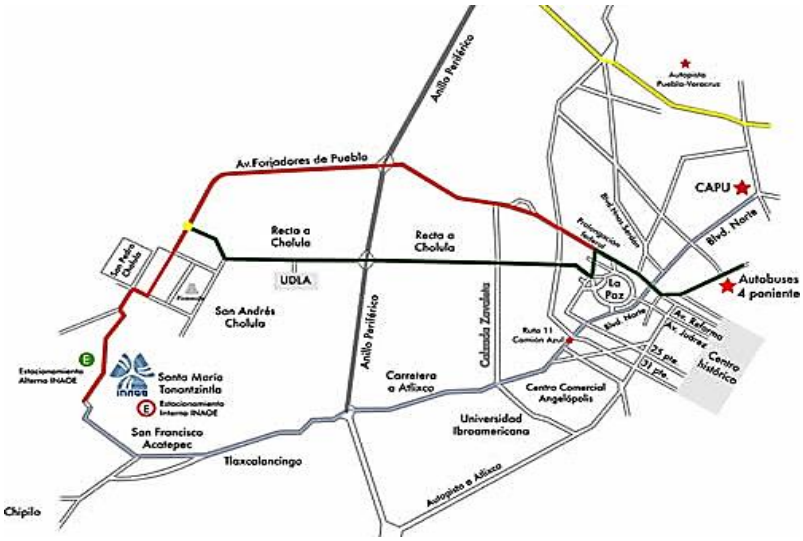

Figura 3: Ubicación del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Tonantzintla, Puebla, México.

1.3 Layout del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE).

1. Laboratorio Microelectrónica
2. Oficinas de estudiantes
3. Edificio del G.T.M.
4. Sala de Eventos
5. Sala Braulio Iriarte
6. Cámara Schmidt
7. Alberca
8. Laboratorio de óptica
9. Docencia
10. Oficinas
11. Auditorio, Centro de Información
12. Entrada
13. Bungalows #1-6
14. 2o. piso: Sala Guillermo Haro
15. Estacionamiento
16. Cancha de Fútbol
17. Cancha de Basquetbol
18. Cafeteria
19. Bungalows #7-30
20. Laboratorio de Superficies Asféricas
21. "La Cabaña"
A. Bungalow B2
B. UNAM bungalow

Figura 4: Layout del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Tonantzintla, Puebla, México.

1.4 Historia de la Coordinación de Ciencias Computacionales del INAOE.

Ciencias Computacionales es la coordinación en el INAOE de más reciente creación. A finales de 1997 se desarrollan y aprueban los programas de Maestría y Doctorado en Ciencias Computacionales, los cuales inician en agosto de 1998. En 1999, el

Consejo Consultivo Interno del INAOE decide crear el Programa en Ciencias Computacionales, donde se ubica a los investigadores en Computación. En el 2001 se decide que el Programa pase a ser Coordinación de Ciencias Computacionales ante el crecimiento en investigadores y estudiantes del programa. **[1].**

1.5 Misión de la Coordinación de Ciencias Computacionales del INAOE.

La CCC tiene como misión contribuir al avance de la ciencia en México en el área de ciencias computacionales y tecnológicas de información, a través de las siguientes metas:

1. Realizar investigación básica de vanguardia en las áreas de especialidad que la caracterizan.
2. Formar maestros y doctores en ciencias capaces de resolver problemas científicos y tecnológicos de alta relevancia en el campo de las ciencias computacionales en las áreas en las cuales se especializa y
3. Llevar a cabo investigación aplicada orientada a satisfacer necesidades planteadas por el sector productivo y de servicios del país. **[2].**

1.6 Líneas y Laboratorios de Investigación de la Coordinación de Ciencias Computacionales del INAOE.

- Aprendizaje Computacional y Reconocimiento de Patrones.
- Cómputo Reconfigurable y de Alto Rendimiento.
- Cómputo y Procesamiento Ubicuo.
- Procesamiento de Bioseñales y Computación Médica.
- Robótica.
- Tecnologías del Lenguaje.
- Visión por Computadora.

1.7 Objetivos de la Línea de Investigación de Robótica.

- Realizar investigación básica y aplicada en robots móviles autónomos e inteligentes.
- Coadyuvar a la transferencia de Tecnología para la solución de problemas reales.

- Formar Estudiantes de maestría y doctorado en el Área de Robótica.

1.8 Temas Principales de Estudio de la Línea de Investigación de Robótica.

- Robótica Colectiva.
- Robótica Probabilística.
- Aprendizaje en Robótica.
- Robots de Servicio. (Ver Figura 5).
- Aplicaciones Médicas.
- Diseños de Prototipos.

Figura 5: Robots de Servicio Markovito – Línea de Investigación de Robótica.

Capítulo 2

Propósito y organización.

En el presente capitulo representa el planteamiento del problema, los objetivos, la justificación y un cronograma de las actividades para la realización de este proyecto.

2.1 Planteamiento del problema.

En la actualidad la robótica está al servicio de las personas ya no solo en sector industrial, sino también en otros ámbitos de la vida humana como el sector salud, agrícola o doméstico. Un claro ejemplo de esto son los robots empleados para realizar tareas de logística, agrícolas y del hogar por mencionar algunas. Para poder realizar estas tareas y al mismo tiempo tener una adecuada interacción con las personas, el robot debe poder percibir el ambiente en el que se desenvuelven, de lo contrario no podría realizar con éxito las tareas asignadas, por lo que se deben desarrollar sistemas sensoriales que doten a los robots de diversas capacidades como visión, escucha, percepción de la profundidad, entre otras más.

El robot de servicio Markovito del área de ciencias de la computación del Instituto Nacional de Astrofísica, Óptica y Electrónica, cuenta con sistema de cuatro sensores de sonido (Fc-04) integrados a una placa de desarrollo Arduino UNO, los cuales otorgan al robot la capacidad de determinar la dirección de origen de fuente sonora que se dirige a él. El sistema implementado solo es capaz de determinar la dirección del sonido en 4 direcciones posibles. Al emplear este sistema en pruebas como hear and rotate, la cual consiste en hacer girar el robot hacia una fuente de sonido no es muy exacto en su posicionamiento respecto a la dirección de la fuente de sonido.

2.2 Objetivos.

2.2.1 Objetivo general.

Implementar un nuevo método para la estimación de la dirección de la fuente sonora, con el uso de la placa de desarrollo Arduino Uno y los sensores de sonido Fc-04. Para obtener una mayor resolución en la estimación de la fuente sonora que el sistema ya existente.

2.2.2 Objetivos específicos.

- Analizar el sistema ya implementado para la detección de la dirección fuente sonora en el robot.
- Investigar y seleccionar un método para la estimación de la fuente sonora.
- Implementación de un método para obtener un nodo que realice la estimación de la fuente sonora.
- Integrar el nodo desarrollado en el robot para su uso en la prueba hear and rotate.

2.3 Justificación.

La implementación actual de la estimación de la fuente sonora en el robot de servicio Markovito, solo es capaz de determinar la dirección de una fuente de sonido en cuatro posiciones atrás, enfrente, derecha e izquierda. Por lo que se busca implementar un nuevo método para determinar la dirección de la fuente de sonido con una mayor resolución su estimación, sin cambiar el hardware ya existente. Para que el robot tenga una mayor exactitud al girar hacia la ubicación de una fuente de sonido y posicionarse de manera frontal a la fuente emisora del sonido.

2.4 Cronograma de actividades.

Actividades	Tiempo de Duración																		
	Febrero				Marzo				Abril				Mayo				Junio		
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3
Definición de la problemática.																			
Planteamiento del estado del arte.																			
Justificación del porqué la implementación del proyecto.																			
Determinación de objetivos y actividades generales para la resolución de la problemática.																			
Establecimiento de las delimitaciones y alcances del proyecto.																			
Designación de la metodología a seguir.																			
Descripción de las actividades a realizar.																			
Estipulación del cronograma de actividades (Plan de trabajo).																			

Búsqueda de fuentes de información sobre la determinación de la fuente sonora.																			
Investigación sobre los recursos y funcionamiento de la plataforma Arduino UNO, así como el funcionamiento del sensor Fc-04.																			
Búsqueda documental: instalación y funciones básicas de ROS.																			
Análisis de tutoriales sobre el uso de la librería rosserial en libros y manuales, para el establecimiento de comunicación entre Arduino y la PC por medio de ROS.																			
Filtración de artículos, tesis, y reportes recopilados sobre la determinación del origen de la fuente sonora en robots de servicio, para su síntesis posterior.																			
Organizar, seccionar y sintetizar manuales y datasheets recopilados de la plataforma Arduino Uno más convenientes para el desarrollo del proyecto.																			
Seleccionar y resumir la información más relevante de las fuentes recopiladas acerca de ROS.																			
Seleccionar y sintetizar la información sobre el paquete rosserial y Arduino.																			
Realización de pruebas de la implementación del método seleccionado en el robot de servicio Markovito.																			
Revisión final del método implementado en el robot de servicio Markovito del área de robótica del INAOE.																			

Tabla 1: Cronograma de actividades.

Capítulo 3
Marco Teórico.

3.1 Proyectos Similares.

3.1.1 Robot Seguidor de Sonido, por Karl Enoksson y Bohan Zhou 2017.

Desarrollo de un robot con un arreglo de cuatro sensores de sonido unidireccionales, los cuales son montados en un vehículo con ruedas (ver Figura 6), la detección del sonido sirve para dar una dirección a la cual se debe dirigir. La aplicación buscada en este proyecto implementado es emplearlo para asistir personas discapacitadas, la localizando personas. El sistema empleado por este equipo para la detección del sonido es TDOA, el cual se basa en la diferencia de tiempo de llegada del sonido a los diferentes micrófonos ubicados en el vehículo. El sistema emplea la tarjeta de desarrollo Arduino, las preguntas de investigación realizadas en este proyecto eran determinar si la tarjeta de desarrollo Arduino era la adecuada para el procesamiento adecuado del sonido y la aplicación del algoritmo TDOA. **[3].**

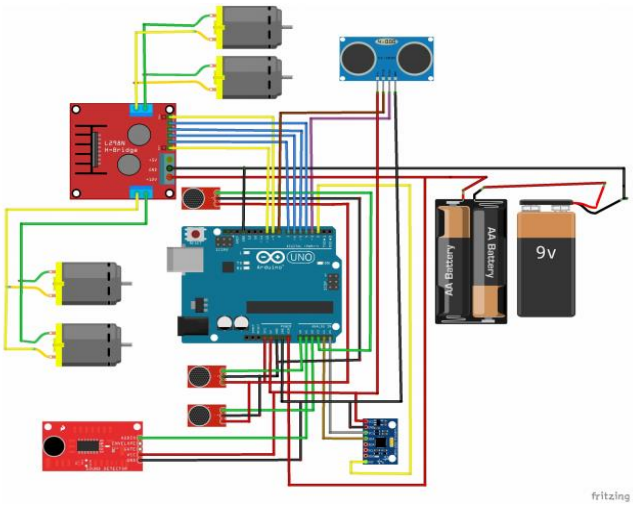

Figura 6: Sistema implementado en el robot seguidor de sonido.

3.1.2 Detección de la dirección de la fuente de sonido usando Robot Autónomo, por Aditya Argawal 2014.

Desarrollaron un sistema para la detección del sonido con frecuencias entre 300 Hz y 3400 Hz, utilizando el algoritmo TDOA basándose en las diferencias de los tiempos de

llegadas entre los micrófonos, el sistema cuenta con un arreglo de tres micrófonos (ver Figura 7). El sistema se fue instalado en el robot autónomo SODDRO (Sound Direction Detection Robot). Este robot, una vez determinada la dirección de la fuente del sonido automáticamente se dirige hacia este punto, la ventaja del desarrollo de este sistema presenta una gran ventaja de implementar un filtro por lo que no es afectado por ruidos externos. **[4].**

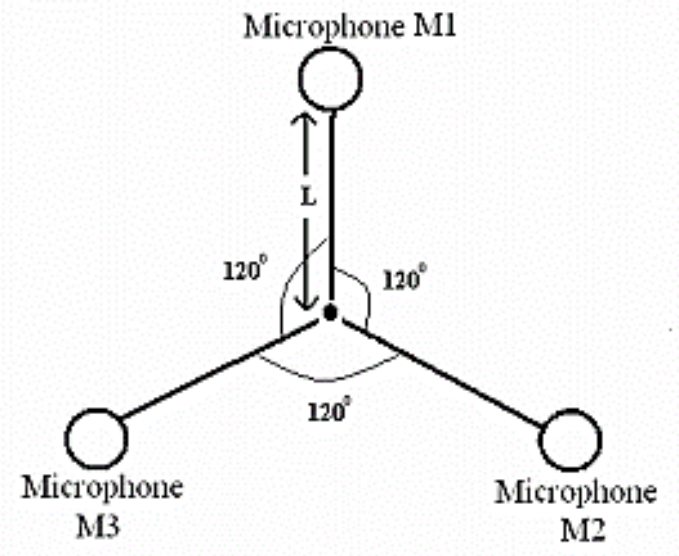

Figura 7: Arreglo de micrófonos para el robot SODDRO.

3.1.3 Localización del Sonido con Arduino, por Jonas Skarstein Waaler y Daniel Gusland 2015.

Desarrollo de un dispositivo electrónico que pueda localizar el sonido y apuntar a la fuente de sonido (ver Figura 8). La aplicación buscada en este proyecto implementado es registrar la diferencia de tiempo entre dos puntos en términos de sonido, utilizando una tarjeta Arduino y dos micrófonos. El sistema empleado por este equipo para la detección del sonido es TDOA, el cual se basa en la diferencia de tiempo de llegada del sonido a los diferentes micrófonos ubicados en el dispositivo electrónico. Como parte de los resultados obtenidos de este proyecto se logró que la unidad ubicara el sonido en dos niveles y apuntara a la fuente de sonido. **[5].**

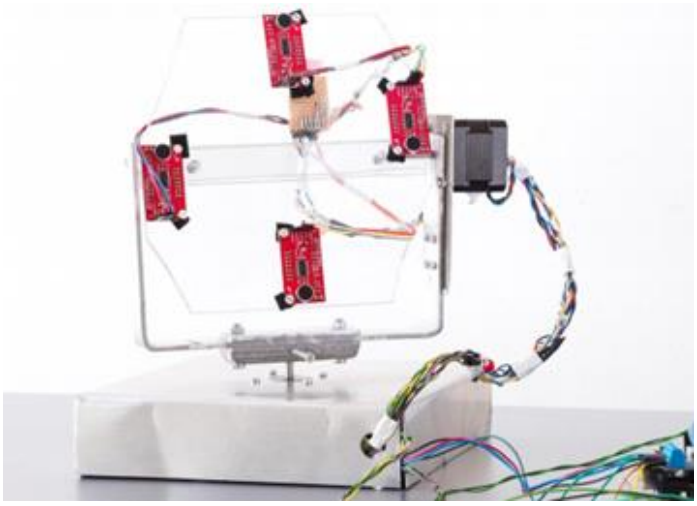

Figura 8: Sistema implementado en el dispositivo electrónico de Localización del Sonido con Arduino.

3.2 Características del sonido y métodos para determinar la dirección de una fuente de sonido.

3.2.1 El sonido.

El sonido son ondas longitudinales que se originadas por la vibración de un cuerpo y propagan en un medio como el aire, el agua y otros medios elásticos. Una onda de sonido puede visualizarse como una onda sinusoidal compuesta.

Las ondas de sonido presentan diversas características que se describen a continuación:

- La frecuencia es el número de vibraciones completas por segundo que realiza una fuente de sonido y que se trasmite en ondas.
- Amplitud relacionada con el volumen y la intensidad del sonido, es la distancia entre el punto más alejado de una onda y el punto de equilibrio.
- Longitud de onda: La distancia que recorre una onda en un periodo determinado de tiempo
- Potencia acústica (W): Cantidad de energía emitida por unidad de tiempo determinada. Se mide en watts y se relaciona con la amplitud de onda.
- Espectro de frecuencia: las características de una onda sinusoidal son trasladables a las ondas del sonido, pero en ambiente real es difícil encontrarse con un sonido con una sola frecuencia o una misma amplitud ya que la mayoría de los sonidos son superposiciones de diversos sonidos, por lo que se generan

sonidos complejos, esta distribución de energía acústica de las diversas ondas de sonido se conoce como espectro de frecuencia de sonido.

- La velocidad del sonido propagado en el aire a una temperatura ambiente de 20ºC es aproximadamente de 343 m/s.

3.2.2 Métodos para determinar la dirección de una fuente del sonido.

La localización de la fuente de sonido (SSL) aborda el problema de estimar la posición de una fuente solo a través de datos de audio. Esto generalmente implica varias etapas de procesamiento de datos.

El modelo de propagación del sonido se propone en función de: la posición de los micrófonos, ya que puede haber un objeto entre ellos; la aplicación robótica, ya que el usuario puede estar muy cerca o lejos de la matriz de micrófonos; y las características de la sala, ya que definen cómo se refleja el sonido desde el entorno. Además, el modelo de propagación generalmente dicta el tipo de características que se utilizarán. El modelo de propagación más popular utilizado es el modelo de campo libre / campo lejano, que asume lo siguiente:

- Campo libre: el sonido que se origina en cada fuente llega a cada micrófono a través de una única ruta directa. Esto significa que no hay objetos entre las fuentes y los micrófonos y que no hay objetos entre los micrófonos. Además, se supone que no hay reflexiones del entorno (es decir, no hay reverberación).
- Campo lejano: la relación entre la distancia entre micrófonos y la distancia de la fuente de sonido a la matriz de micrófonos es tal que la onda de sonido se puede considerar como plana.

El modelo de campo cercano asume que el usuario puede estar cerca de la matriz de micrófonos, lo que requiere considerar la onda de sonido como circular. Hay algunas aplicaciones robóticas que utilizan el modelo de campo cercano, sin embargo, no se usa tan comúnmente como el modelo de campo lejano. De hecho, existen enfoques que utilizan un modelo de campo lejano modificado con éxito en circunstancias de campo cercano o que modifican el diseño de la metodología para considerar el caso de campo cercano. Sin embargo, un modelo de campo lejano utilizado directamente

en circunstancias de campo cercano puede disminuir considerablemente el rendimiento de SSL.

Método por Diferencia de Intensidades.

La razón por la que podemos localizar la fuente de un sonido con precisión es que tenemos dos oídos. En cada oído, se percibirá una señal ligeramente diferente y, al analizar estas diferencias, el cerebro puede determinar dónde se originó el sonido. Las dos señales de localización más importantes son la diferencia de tiempo interaural, o ITD, y la diferencia de intensidad o IID. El IID surge del hecho de que, debido a la sombra de la onda de sonido de la cabeza (sombra de la cabeza), un sonido proveniente de una fuente ubicada en un lado de la cabeza tendrá una mayor intensidad, o será más alto, en el oído, más cerca de la fuente de sonido. (El IID más pequeño que la mayoría de las personas puede detectar de manera confiable es de aproximadamente 1 dB.) Por lo tanto, se puede crear la ilusión de una fuente de sonido que emana de un lado de la cabeza simplemente ajustando el nivel relativo de los sonidos que se transmiten a dos altavoces o auriculares separados. Esta es la base del control de paneo comúnmente usado. (Ver Figura 9).

Figura 9: Método por diferencia de intensidades.

Método por Tiempo de Diferencia de Arribo (TDOA).

La diferencia de tiempo arribo (TDOA) cuando a humanos o animales respecta, es la diferencia de tiempo de llegada de un sonido entre dos orejas y oídos. Es importante en el proceso de localización de sonidos, ya que proporciona la dirección de la fuente

sonora con respecto a la cabeza. Si una señal llega a la cabeza de un lado A, la señal debe viajar un mayor recorrido para lograr llegar al lado B. Este aumento de recorrido de la onda resulta en una diferencia de tiempo en la llegada del sonido a los oídos, lo cual es detectado por el sistema nervioso para ser analizado.

La teoría Dúplex propuesta por el Barón Rayleigh en 1907, provee una explicación a la habilidad de los humanos de localizar la fuente de sonido por las diferencias de tiempo interaural y diferencias en niveles interaurales (DNI).

La teoría Dúplex establece que las DTIs son usadas para localizar fuentes de sonido de bajas frecuencias, mientras que las DNI son usadas para la ubicación de fuentes de sonido de altas frecuencias. No obstante, los rangos de frecuencia que el sistema auditivo utiliza se enciman, y ya que los sonidos más naturales contienen altas y bajas frecuencias en la mayoría de los casos, el sistema auditivo combina información tanto como de las DTIs y DNIs para juzgar la ubicación de la fuente sonora. A consecuencia de este sistema dúplex, es posible generar estímulos llamados intercambio de tiempo-intensidad en dispositivos como audífonos.

Cuando una señal es producida en el plano horizontal, al ángulo en relación a la cabeza se le conoce como acimut. A 0 grados (0°), la fuente de sonido se encuentra enfrente de la persona, a 90° se encuentra a la derecha y a 180° se encontrará detrás de ella. Esta diferencia de tiempo es usada por nuestro cerebro para calcular la procedencia de la fuente emisora. (Ver Figura 10).

Esta parte de la Teoría Dúplex se basa en las diferencias de fase creadas por la llegada de un mismo sonido a dos receptores distintos en dos momentos diferentes. Las ITD varían desde 0 segundos para fuentes sonoras con un azimut de 0° hasta cerca de 0.69 milisegundos para fuentes sonoras con un ángulo de 90°.

El TDOA para un par dado de micrófonos y la fuente se define como la diferencia de tiempo entre las señales recibidas por los dos micrófonos.

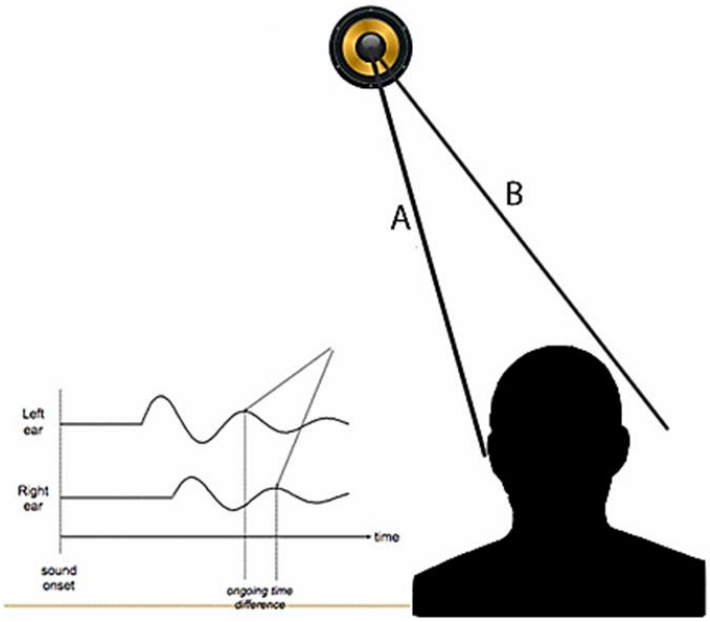

Figura 10: Método por diferencia de tiempo de arribo.

La frecuencia de la señal incidente y la separación máxima permitida entre cada par de micrófonos en la matriz está restringida a | π |. Cualquier diferencia de fase fuera del rango de −π y π está envuelta alrededor de este rango. Esto coloca una restricción importante en la geometría de la matriz cuando se realiza una estimación de TDOA. Considere una señal incidente en el par de micrófonos como se muestra en la Figura 11, en un ángulo θº. Deje que la señal de banda ancha tenga una frecuencia máxima de $f_{máx}$. En $f_{máx}$, si restringimos la diferencia de fase entre las señales de un par de micrófonos es menor o igual a π, entonces requerimos:

$$2\pi f_{máx}\, \tau \leq \pi$$

Y

$$\tau = \frac{dsen(\theta)}{v}$$

Donde

$\tau =$ señal de retardo de tiempo entre los dos micrófonos,

$d =$ distancia entre el par de micrófonos,

$\theta =$ ángulo de incidente,

$v =$ velocidad del sonido.

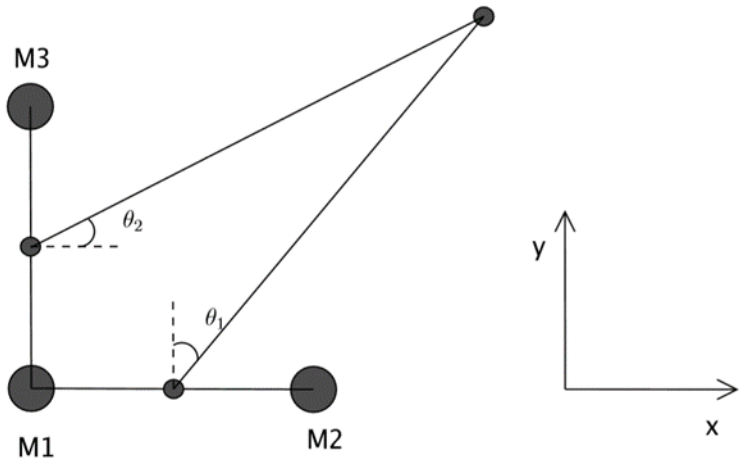

Figura 11: Micrófono de tres elementos y dos dimensiones.

3.3 Sensor de Sonido y voz FC- 04.

El sensor de sonido FC-04 es un módulo que emplea un micrófono que convierte una señal acústica en una señal eléctrica. La señal eléctrica proveniente del micrófono pasa por un divisor de voltaje y es convertida a un valor cercano a los 2V. Usando el amplificador operacional LM393 la señal es amplificada y comparada contra un voltaje de referencia. La referencia es ajustada por medio de un potenciómetro y de esta manera se establece un umbral en la detección del sonido en la Figura 12 podemos observar un diagrama a bloques de este proceso. Al final tendremos a la salida 0V cuando hay detección de sonido y 5V cuando no hay detección de sonido.

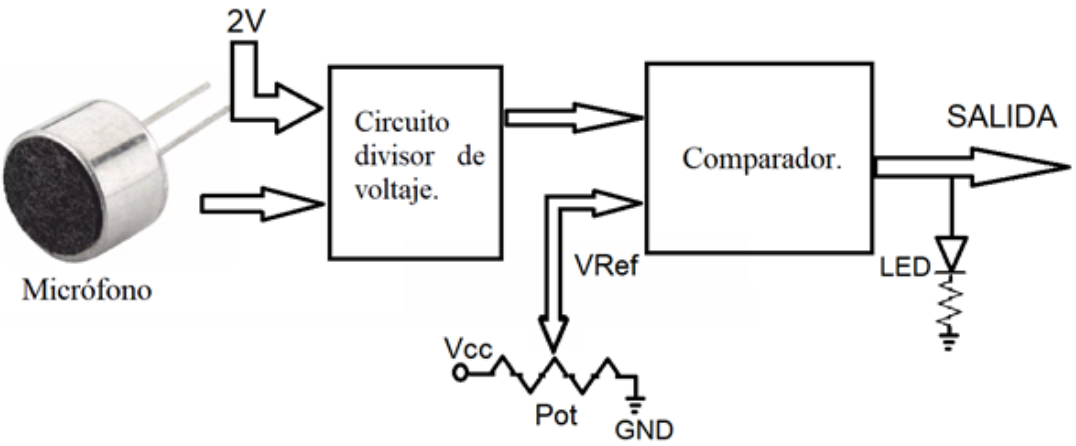

Figura 12: Diagrama a bloques del sensor FC-04.

3.4 Arduino.

Es una plataforma de desarrollo, basada en una placa electrónica de hardware libre, la cual incorpora un microcontrolador reprogramable. La mayoría de las placas de desarrollo Arduino pertenecen a la familia de microcontroladores AVR, por lo que comparten características de software, arquitectura librerías y documentación. Su lenguaje de programación está basado en C++. Arduino nació en el año de 2005 en el Instituto de Diseño Interactivo de Ivrea (Italia), Arduino apareció con la necesidad de contar con un dispositivo de bajo coste, para uso escolar. Sus creadores decidieron liberarlo y abrirlo al público para la evolución del proyecto.

3.4.1 Arduino Uno.

Es una placa electrónica con un microcontrolador de 8 bits el Atmega328p (Ver Figura 13). Este tiene 14 entradas/salidas digitales, 6 entradas analógicas y un reloj de cristal de16 MHz, una conexión USB, alimentación por Jack, un circuito para programación serial ICSP, un botón para reset.

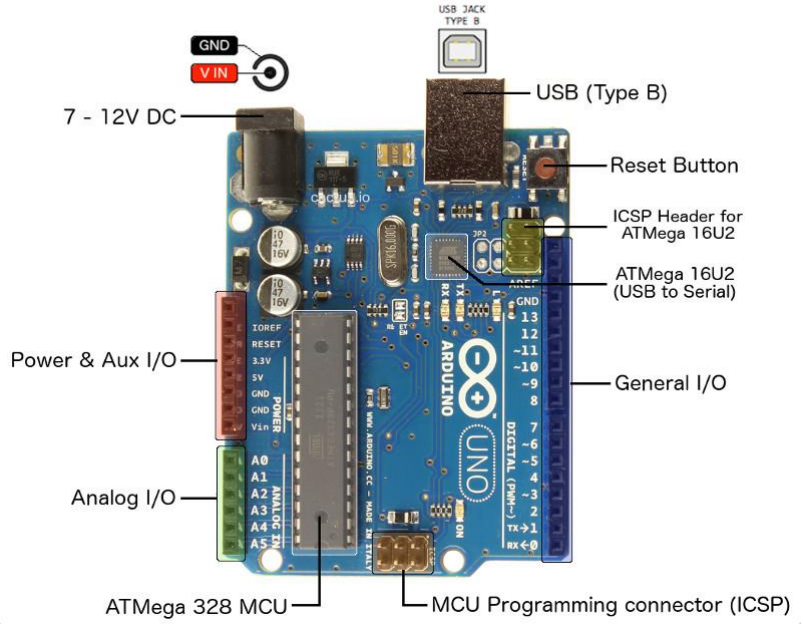

Figura 13: Arduino Uno.

3.4.2 Atmega328p.

Es un microcontrolador de alto desempeño y arquitectura RISC de 8 bits, de bajo consumo energético, 32x8 registros de propósito general, 32k bytes de memoria flash programable, 1kbyte de memoria EEPROM, 2k bytes de memoria interna. Dos timers de 8 bit con prescaler separado y un timer de 16 bits con prescaler separado. Dos pines de interrupción externa y 20 pines de interrupciones por cambio de pin. Comunicación serial (USART), entre otras características más.

3.4.3 Interrupciones.

Las interrupciones son mecanismos del microcontrolador para responder a eventos, suspendiendo el programa principal, para la ejecución de una subrutina de servicio (ISR Interrupt Service Routines) una vez terminada la ejecución de la subrutina regresa al programa principal como se muestra en la Figura 14.

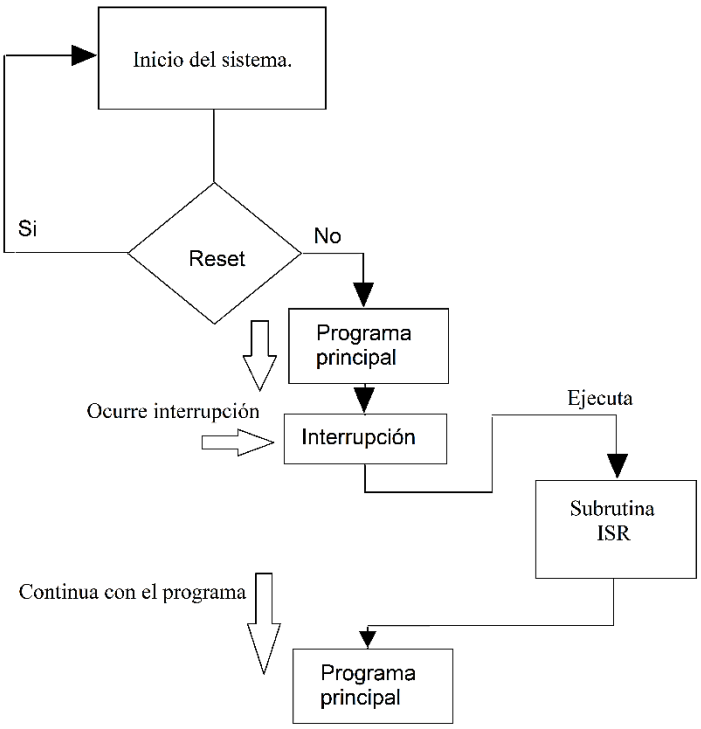

Figura 14: Ejecución de una Interrupción.

Las interrupciones pueden ser generadas por dispositivos periféricos, conectados a pines del microcontrolador que envían información a este, esto puede ser de manera

asíncrona. También es posible generadas por medio de una señal digital conectada a un pin específico.

Los microcontroladores AVR incluyen un controlador de interrupciones para controlar la solicitud de eventos o dispositivos periféricos, lo cual evita consumir tiempo al microcontrolador para censar o monitorear la información de un evento ocurrido.

Cuando se genera una interrupción la unidad de interrupciones indica que un evento requiere solicitud de interrupción, el microcontrolador consulta los registros del controlador de interrupción para determinar qué tipo de interrupción se generó y guarda el estatus del contador del programa. El número de interrupción se determina por medio de la tabla de vectores de interrupción como se muestra en la Tabla 2, con las direcciones de las subrutinas ISR de servicio.

Vector	Dirección en memoria (Registro)	Fuente	Definición de interrupción.
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset.
2	0x0002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt

8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete

23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

Tabla 2: Vector de Interrupción Atmega328p.

Cuando se genera una interrupción, el microcontrolador termina la ejecución de la instrucción en curso guarda el estado de los registros y banderas así mismo la dirección de memoria del contador del programa. Va a la dirección donde se almacena la rutina de servicio de interrupción ISR y ejecuta dicha subrutina; al terminar de ejecutarla el microcontrolador restaura el estado que se había guardado y retorna al programa principal a partir de la siguiente instrucción donde se quedó el contador del programa.

Las subrutinas ISR son funciones que se ejecutan al ocurrir el evento de interrupción, estas funciones no tienen parámetros de entrada y en casi ningún caso retornan valor. Una característica importante de la ISR es que el código necesario para llevar acabo la tarea específica del proceso a la que atienden debe ser lo más corto posible, para evitar alteraciones en el programa principal.

3.4.4 Tipos de Interrupciones.

El microcontrolador atmega328p tiene diferentes tipos de interrupciones:

- **Externas:** Tiene asociado las interrupciones INT0 e INT1 mapeadas en los pines 2 y 3. Estas interrupciones son muy rápidas y eficientes porque pueden ser disparadas por transiciones de subida y bajada, así también por niveles de voltaje 0.
- **Interrupciones por cambio de pin:** Este tipo de interrupciones tiene asociada 20 señales de disparo en los puertos D, C y B. Una desventaja que presenta que son un poco menos eficientes que las interrupciones externas.

3.4.5 Registros de interrupciones por cambio de pin.

PCICR (Registro de Control de Interrupciones por cambio de pin). (Ver Tabla 3).

PCICR								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(0x68)	-	-	-	-	-	PCIE2	PCIE1	PCIE0
Lectura /Escritura	L	L	L	L	L	L/E	L/E	L/E
Valor inicial	0	0	0	0	0	0	0	0

Tabla 3: Registros de interrupciones por cambio de pin.

Bit 2- PCIE2 Pin Change Interrupt Enable 2.
Cundo este bit es puesto a uno las interrupciones por cambio de pin 2 son habilitas del PCINT16 al PCINT23, en todos estos pines se habilitan las interrupciones por cambio de pin, estos pines corresponden al registro PCMSK2.

Bit 1- PCIE1 Pin Change Interrupt Enable 1.
Cundo este bit es puesto a uno las interrupciones por cambio de pin 1 son habilitas del PCINT14 al PCINT8, en todos estos pines se habilitan las interrupciones por cambio de pin, estos pines corresponden al registro PCMSK1.

Bit 0- PCIE0 Pin Change Interrupt Enable 0.
Cundo este bit es puesto a uno las interrupciones por cambio de pin 0 son habilitas del PCINT7 al
PCINT0, en todos estos pines se habilitan las interrupciones por cambio de pin, estos pines corresponden al registro PCMSK0.

PCMK2- Registro de la máscara del pin change interrupt 2. (Ver Tabla 4).

Bit	7	6	5	4	3	2	1	0
(0x6D)	PCINT 23	PCINT 22	PCINT 21	PCINT 20	PCINT 19	PCINT 18	PCINT 17	PCINT 16
Lectura/Escritura	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Tabla 4: PCMK2- Registro de la máscara del pin change interrupt 2.

Con la activación del bit PCIE2 en el registro PCICR, la interrupción de cambio de pin se habilita para todos los pines PCINT23 al PCINT16, aplicando una máscara al registro PCMK2 podemos activar cada una de las interrupciones por cambio de pin de manera individual a cada pin o varias a la vez y de la misma manera desactivarlas.

PCMK1- Registro de la máscara del pin change interrupt 1. (Ver Tabla 5).

Bit	7	6	5	4	3	2	1	0
(0x6D)	-	PCINT 14	PCINT 13	PCINT 12	PCINT 11	PCINT 10	PCINT9	PCINT8
Lectura/Escritura	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Tabla 5: PCMK1- Registro de la máscara del pin change interrupt 1.

Con la activación del bit PCIE1 en el registro PCICR, la interrupción de cambio de pin se habilita para todos los pines PCINT14 al PCINT8, aplicando una máscara al registro PCMK1 podemos activar cada una de las interrupciones por cambio de pin de manera individual a cada pin o varias a la vez y de la misma manera desactivarlas.

PCMK0- Registro de la máscara del pin change interrupt 0. (Ver Tabla 6).

Bit	7	6	5	4	3	2	1	0
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
Lectura/Escritura	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Tabla 6: PCMK0- Registro de la máscara del pin change interrupt 0.

Con la activación del bit PCIE0 en el registro PCICR, la interrupción de cambio de pin se habilita para todos los pines PCINT7 al PCINT0, aplicando una máscara al registro PCMK0 podemos activar cada una de las interrupciones por cambio de pin de manera individual a cada pin o varias a la vez y de la misma manera desactivarlas.

3.4.6 Timers.

Una importante característica de los sistemas embebidos basados en un microcontrolador son la capacidad de trabajar con tiempos con una gran precisión, los periféricos integrados al microcontrolador son los timers. Nosotros los usamos todos los días, un ejemplo simple es el reloj que cronometra el tiempo transcurrido en segundos, minutos y horas en el transcurso del día. Los timer AVR tiene una función similar, midiendo un intervalo determinado de tiempo.

Un timer es simplemente un registro, el microcontrolador Atmega328p tiene timers de 8 y 16 bits de resolución, el valor del registro aumenta automáticamente su valor con una señal de reloj, una vez que llega a su desborde este se reinicia su cuenta Figura 15.

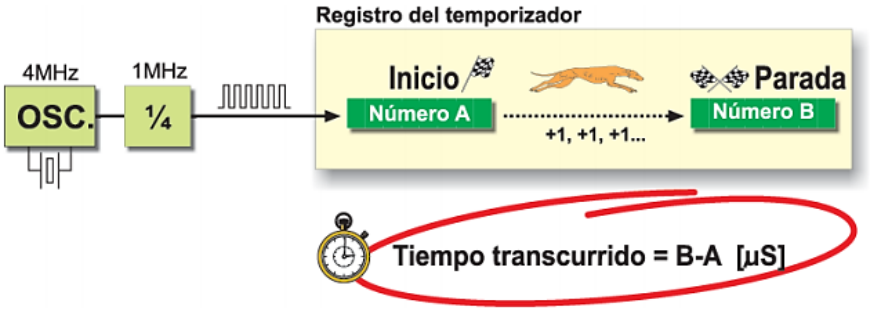

Figura 15: Diagrama de un Timer.

La frecuencia de operación del reloj que posee el microcontrolador Atmega328p es de 16MHZ, esta frecuencia es muy alta por lo que es necesario recurrir a los prescalers, este es un divisor de frecuencia programable, lo cual nos sirve para reducir esta frecuencia muy alta a una deseada Figura 16.

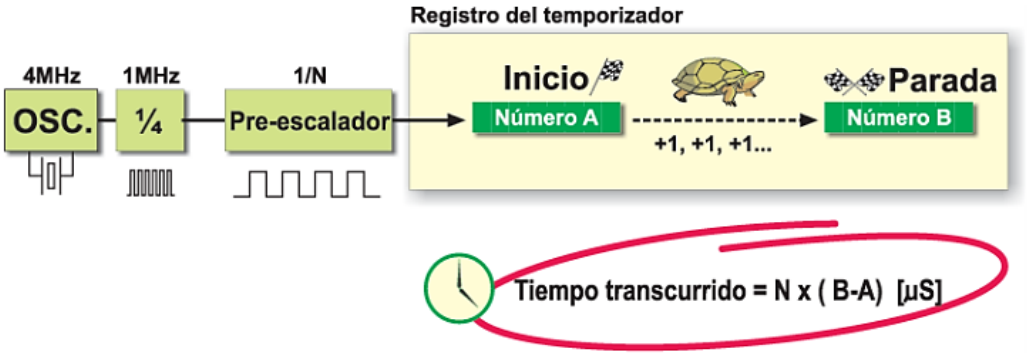

Figura 16: Diagrama de un timer con prescaler.

3.4.7 Timer 2 (Atmega328p).

El timer1 es un temporizador contador con una resolución de 8 bits, por lo que puede tomar valores de 0 a 255. El timer 2 pude ser utilizado tanto en su configuración normal, comparador y modulación por ancho de pulso el diagrama esquemático de este timer se puede ver en la Figura 17.

Este timer cuenta con dos registros para la configuración, en sus diferentes modos de operación los cuales se muestran a continuación en la Tabla 7 y 8.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(0x80)	COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20
Lectura/Escritura	L/E	L/E	L/E	L/E	L	L	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Tabla 7: Tabla Registro TCCR2A.

Bit	7	6	5	4	3	2	1	0
(0x81)	FOC2A	FOC2B	-	_-	WGM22	CS22	CS21	CS20
Lectura/Escritura	L/E	L/E	L	L	L	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Tabla 8: Tabla Registro TCCR2B.

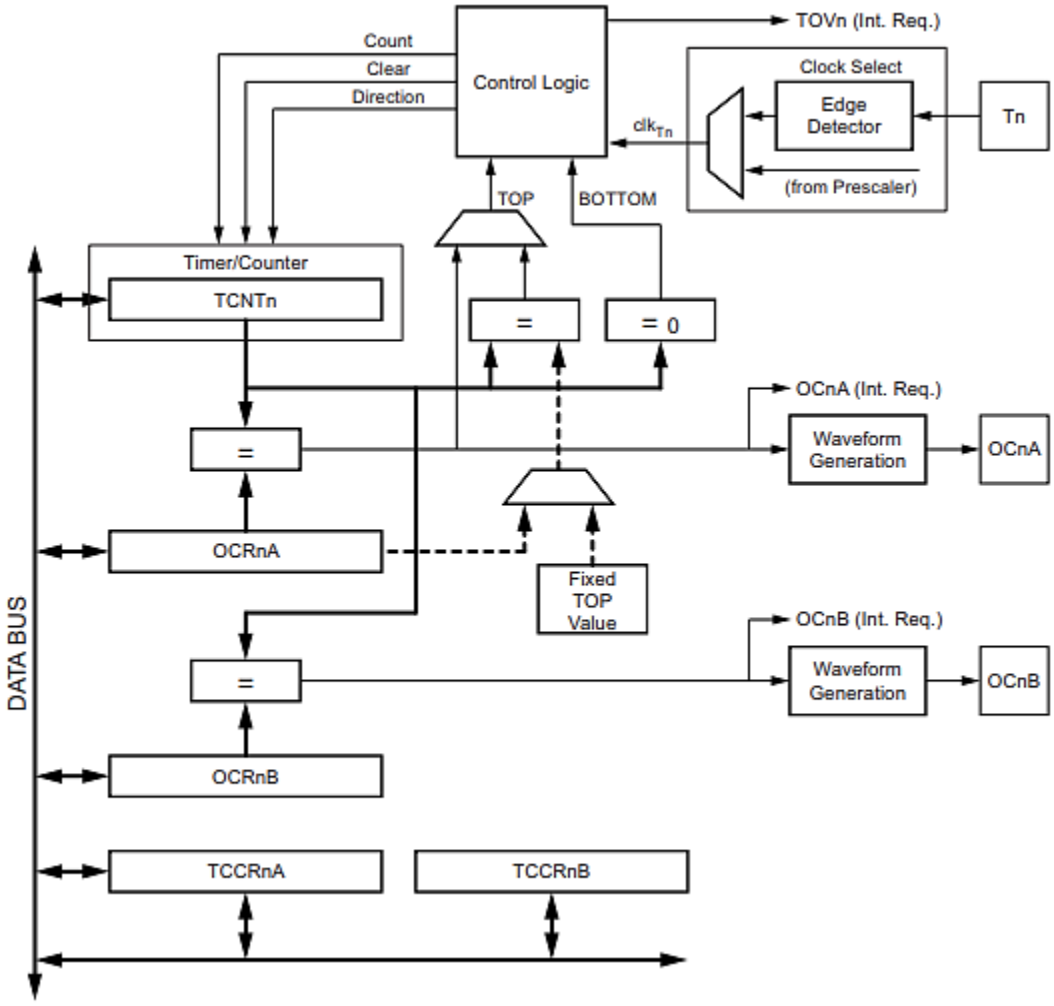

Figura 17: Diagrama del Timer 2.

3.4.8 Interrupción por Timer 2.

Cuando la señal de reloj activa el control lógico del timer este incrementa el registro TCNT2 en una unidad, cuando este registro alcanza su valor máximo (TOP) este se establece a cero, para lograr esto el timer debe estar en configuración normal, que se utiliza para la generación de una interrupción cuando ocurre un desborde en el registro

TCN2, para activar la interrupción, si esta es deseada se debe establecer un set en el bit T0IE2 del registro TIMSK2.

Para establecer el timer 2 en su con figuración normal se deben configurar los bits COM2A1 y COM2A0 del registro TCCR2A como se muestra en la Tabla 9.

MODE	COM2A1	COM2A0	Descripción
0	0	0	Operación en modo Normal, OC2A desconectado.

Tabla 9: Arreglo de bits para configuración Normal.

Si empleamos la señal de reloj con su frecuencia por defecto, tendremos un incremento en el registro TCNT2 cada 62.9ns, por lo que el valor máximo a contabilizar es de 16.1 μs, por lo que es necesario prescalar esta frecuencia para obtener diferentes tiempos, este timer tiene 8 configuraciones para diferentes valores de prescaler, que se pueden observar en la Tabla 10.

CS22	CS21	CS20	Descripción
0	0	0	No fuente de reloj (timer/counter detenido)
0	0	1	$clk_{I/O}$/1 (No prescalado)
0	1	0	$clk_{I/O}$/8 (prescalado)
0	1	1	$clk_{I/O}$/32 (prescalado)
1	0	0	$clk_{I/O}$/64 (prescalado)
1	0	1	$clk_{I/O}$/128 (presacalado)

1	1	0	$clk_{I/O}$/256 (presacalado)
1	1	1	$clk_{I/O}$/1024 (presacalado)

Tabla 10: Establecer prescaler en el registro TCCR2B.

3.5 ROS.

3.5.1 ¿Qué es ROS?

Es un open-source, meta sistema operativo para tu robot. Esto te proporciona los servicios que tu esperarías de un sistema operativo, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades de uso común, paso de mensajes entre procesos, y gestión de paquetes. Esto también provee herramientas y librerías para obtención, construcción, escritura y ejecución de código en múltiples plataformas y equipos.

3.5.2 ¿Entonces ROS es un sistema operativo?

Ros no es un sistema operativo como Windows, Linux y Android, este es un meta sistema operativo. Entonces qué es un meta sistema operativo, es un sistema que ejecuta procesos tales como programación, carga, monitoreo y manejo de errores, haciendo uso de una capa de virtualización y recursos computacionales distribuidos.

Entonces podemos decir que ROS corre en los sistemas operativos convencionales tales como Android Linux y Windows. Por ejemplo, para emplear ROS en una distribución de Linux tal como Ubuntu, primero deberemos instalar ROS como un programa, una vez completada la instalación, las características convencionales de este sistema operativo como sistema de gestión archivos, las interfaces de usuarios y utilidades para la programación (compiladores, ejecución de hilos, etc.) podrán ser utilizadas.

ROS no solo soporta comunicación para un solo sistema operativo, sino también con múltiples sistemas operativos, hardware y programas (ver Figura 18). Por lo que ROS es el adecuado para el desarrollo de robots donde se combinen diversos hardware.

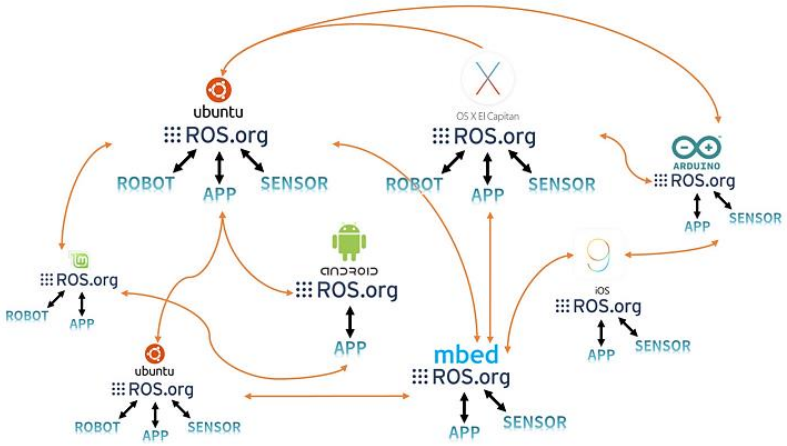

Figura 18: Multi-Comunicación de ROS.

3.5.3 Los objetivos de ROS.

La meta de ROS es construir el entorno de desarrollo que permita el desarrollo software a nivel mundial, ros se enfoca en reutilización y desarrollo de código de la robótica. Las características de ROS son:

- Procesos Distribuidos: Esto es programar en mínimas unidades de procesos ejecutables (Nodos), y cada proceso correr independientemente e intercambia datos de forma sistemática.
- Gestión de Paquetes: Múltiples procesos teniendo el mismo propósito son manejados como un paquete así que es fácil de usar y desarrollar, estos paquetes se pueden compartir, modificar y distribuir.
- Repositorio Publico: Cada paquete es echo publico si el desarrollador así lo desea en repositorios como GitHub.
- API: Cuando se desarrolla un programa que usa ROS, este se diseña para llamar una API e insertar fácilmente en el código que se está implementando.
- Soporte en varios lenguajes de programación: El programa de ROS provee una librería cliente que soporta varios lenguajes, como C++, Python, Lisp, Java, etc.

3.5.4 Componentes de ROS.

Ros consiste de una librería cliente que soporta varios lenguajes de programación, interfaces y hardware, comunicación para la transición y recepción de datos, es un framework para el desarrollo de aplicaciones robóticas, también ofrece herramientas con las cuales puedes controlar un robot en un espacio virtual y herramientas de desarrollo de software (ver Figura 19).

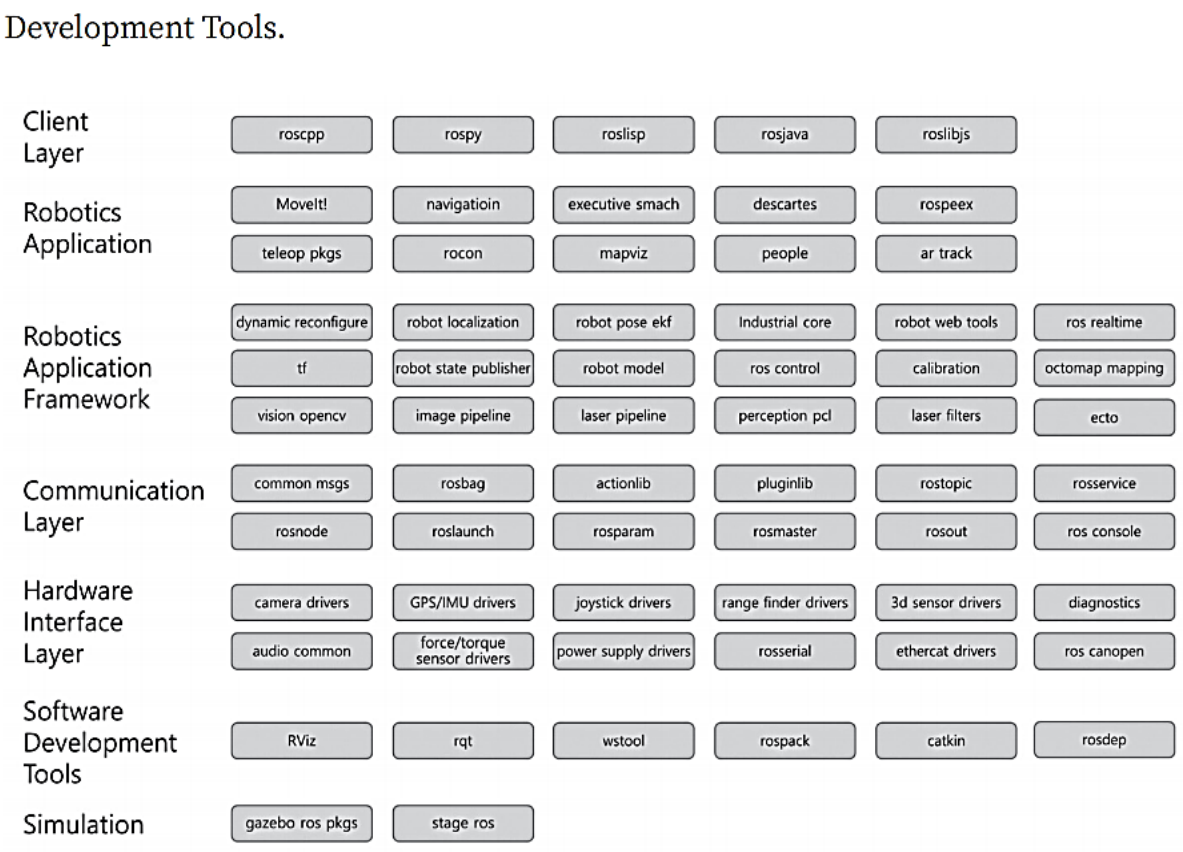

Figura 19: Componentes de ROS.

3.5.5 Ecosistema de Ros.

Un ecosistema es la estructura que une a las empresas de desarrollo de hardware, compañías que desarrollan sistemas operativos, desarrolladores de aplicaciones y usuarios. Cuando ROS apareció fue lo suficientemente atractiva como para construir un ecosistema, aunque este efecto aun es pobre, el número de usuarios, empresas relacionadas a la robótica, centros de investigación, herramientas y librerías, relacionados a ROS han ido aumentando, por lo que en un futuro ROS se convertirá en un ecosistema funcional. (Ver Figura 20).

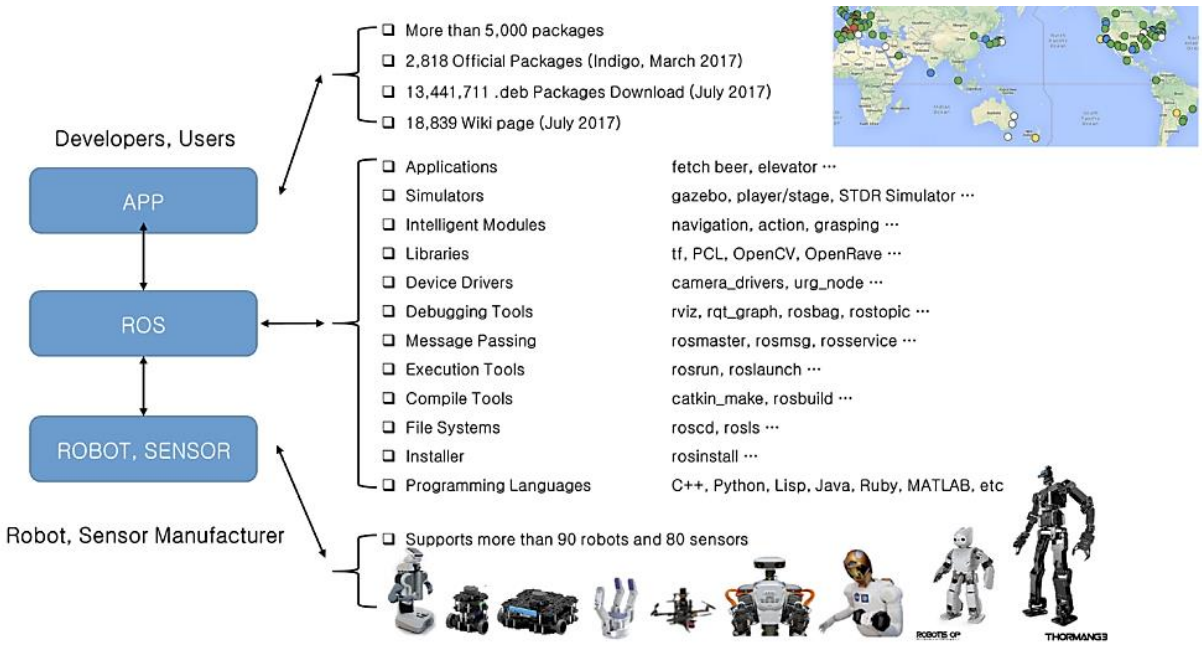

Figura 20: Ecosistema de ROS.

3.5.6 Versiones de ROS.

En noviembre de 2007 Willow Garage una empresa reconocida en el campo de los robots personales y de servicio, la misma que desarrollo la librería Poind Cloud (PCL) usada para dispositivos 3D como el Kinect y la librería para el procesamiento de imágenes de Opencv comenzó a desarrollar ROS y el 22 de enero de 2010 ROS 1.0 salió al mundo. Desde entonces las distribuciones de ROS disponibles son:

1) ROS Box Turtle.
2) ROS C Turtle.
3) ROS Diamondback
4) ROS Electric Emys.
5) ROS Fuerte Turtle.
6) ROS Grooby Galapagos.
7) ROS Hidro Medusa.
8) ROS Indigo Igloo.
9) ROS Jade Turtle.
10) ROS Kinectic Frame.
11) ROS Lunar Loggerhead.
12) ROS Melodic Morenia.

3.5.7 Conceptos de ROS.

Maestro (Master).

Es un servidor de nombres para las conexiones entre nodos y paso de mensajes. El comando roscore es usado para correr el master, este registra el nombre de cada nodo y se puede obtener información cuando se necesita de cualquier nodo. La conexión entre nodos y comunicación de mensajes como son los tópicos servicios son imposibles sin el rosmaster, puesto que master se comunica con esclavos usando el protocolo XMLRCP basado en HTTP, el cual es soportado por varios lenguajes y es muy ligero, no mantiene conexión, por lo que el nodo esclavo solo se conecta cuando este necesita publicar su propia información o consultar la de alguien más.

Nodo.

Un nodo es la unidad de proceso de ejecución más pequeña de ROS, se recomienda crear un nodo para cada propósito, para la fácil reutilización. Los nodos registran información tal como nombre, tipo de mensaje, dirección URL, número del puerto del nodo. El registro del nodo actúa como un publicador, suscriptor, servidor de servicio y cliente de servicio basado en el registro de la información. Los nodos pueden cambiar los mensajes utilizando tópicos y servicios.

Paquete (Package).

Un paquete es la unidad básica de ROS, Las aplicaciones son desarrolladas en paquetes básicos y estos paquetes contienen un archivo para cargar otros nodos o paquete. Los paquetes también contienen todos los archivos necesarios para correr el paquete, esto incluye las librerías de dependencias de ROS para la ejecución de varios procesos, conjunto de datos y configuración de archivos.

Mensajes (Message).

Un nodo envía o recibe datos entre nodos por medio del mensaje. Mensajes son variables como enteros, flotantes, booleanos, strings, entre otros. Un mensaje anidado es una estructura que contiene otros mensajes o arrays de mensajes que pueden ser usados en el mensaje. Los tópicos utilizan entrega de mensajes unidireccionales. A

continuación, se muestra en la Tabla 11 los diferentes tipos de datos para los mensajes de ROS.

Bool	unsigned 8-bit int	int8_t
int8	signed 8-bit int	int8_t
uint8	unsigned 8-bits int	uint8_t
int16	signed 16-bits int	int16_t
uint16	unsigned 16-bits int	uint16_t
int32	signed 32-bits int	int32_t
uint32	unsigned 32-bits int	uint32_t
int64	signed 64-bits int	int64_t
uint64	unsigned 64-bits int	uint64_t
float32	32-bit IEEE float	Float
float64	64-bit IEEE float	Doublé
String	ascii string	std::string
Time	sec/nsecs unsigned 32-bits ints	ros::time
Duration	secs/nsecs signed 32-bit ints	ros::Duration

Tabla 11: Tipo de datos básicos de ROS.

Tópico (Topic).

Los tópicos son como los temas de una conversación. El nodo publicador primero registra este tópico con el maestro y entonces comienza la publicación de mensajes en un tópico. Los nodos subscritores los que buscan recibir la información solicitada al tópico del nodo publicador correspondientes al nombre del tópico al que se registraron en el master. En base a esta información, el nodo suscriptor se conecta directamente al nodo publicador para intercambiar mensajes como un tópico. En la comunicación por medio de tópico, el nodo subscriptor ye nodo publicador manejan el mismo tipo de dato en el mensaje. Pueden existir múltiples suscriptores para un mismo tópico como se muestra en la Figura 21.

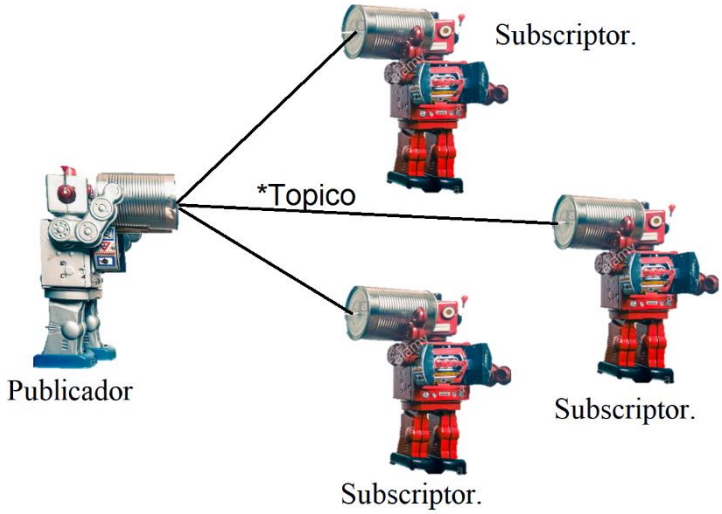

Figura 21: Tópico con múltiples subscriptores.

Publicar y Publicador (Publish and Publisher).

El publicar es transmitir un mensaje a un tópico correspondiente. EL nodo publicador registra su información y tópico con el maestro, también envía un mensaje a todos los nodos que estén interesados en el mismo tópico. El publicador es declarado en el nodo y puede ser declarado múltiples veces en el mismo nodo.

Suscriptor y Suscribirse.

El termino suscribirse significa la acción de recibir un mensaje correspondiente de un tópico. El nodo subscriptor registra su información y tópico con el maestro, y recibe información del nodo que publica el tópico desde el maestro. Basado en la información recibida del nodo publicador, el nodo subscriptor solicita la conexión directa con el nodo publicador y recibe mensajes del nodo publicador conectado. El suscritor es declarado en el nodo y puede ser declarado múltiples veces en un mismo nodo.

La comunicación por medio de tópico es de tipo asíncrono, está basado en publicar y subscribirse, esto es muy usado para la transferencia de ciertos tipos de datos. Debido a que el tópico transmite continuamente y recibe un flujo de mensajes una vez conectado se emplea para sensores que requieren transmitir datos en un flujo continuo.

Flujo de comunicación de mensajes.

El Maestro maneja la información de los nodos, y cada nodo se conecta y comunica con otros nodos como sea necesario. A continuación, se muestra la secuencia para ejecutar nodos y transmitir un tópico entre ellos.

Correr el Maestro.

El nodo Maestro es que maneja la información en un mensaje la comunicación entre nodos es el primer elemento que se debe ejecutar con el comando *roscore.* El maestro registra el nombre de nodos, tópicos, acciones, tipo de mensajes, las direcciones URL y puertos para las conexiones entre nodos y la retransmisión de la información a petición de otros nodos (ver Figura 22).

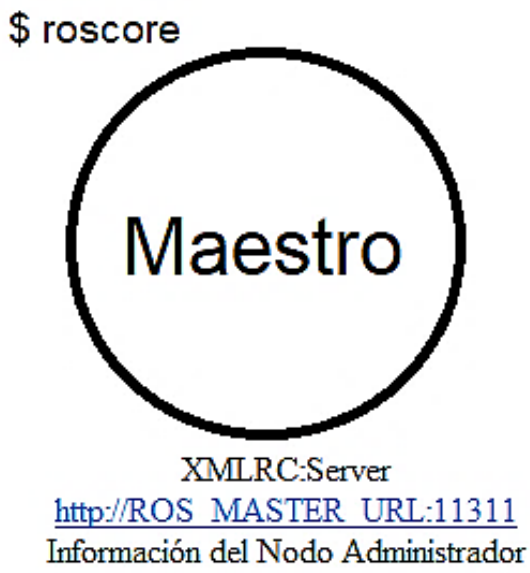

Figura 22: Corriendo el nodo Maestro.

Corriendo el nodo subscriptor.

Los nodos subscriptores son cargados tanto con el comando rosrun o el comando roslaunch. El nodo subscriptor registra su nombre, el nombre del tópico, el tipo de mensaje, la dirección URL y el puerto con el que el maestro está corriendo (ver Figura 23).

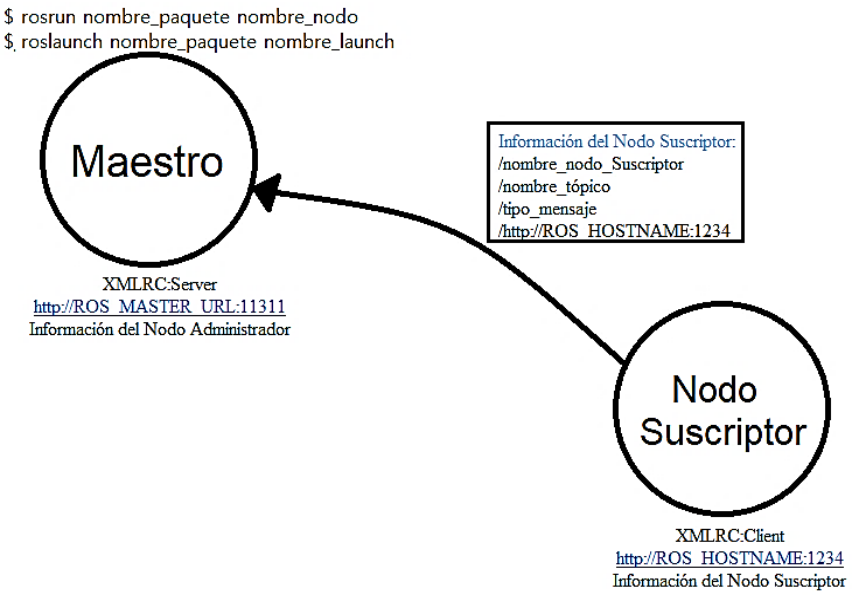

Figura 23: Corriendo el nodo Suscriptor.

Corriendo el nodo Publicador.

El nodo publicador se puede ejecutar como el suscriptor, con los comandos rosrun y roslaunch. El nodo publicador registra el nombre del nodo, el tópico del nodo, el tipo de mensaje, la dirección URL y el puerto. (Ver Figura 24).

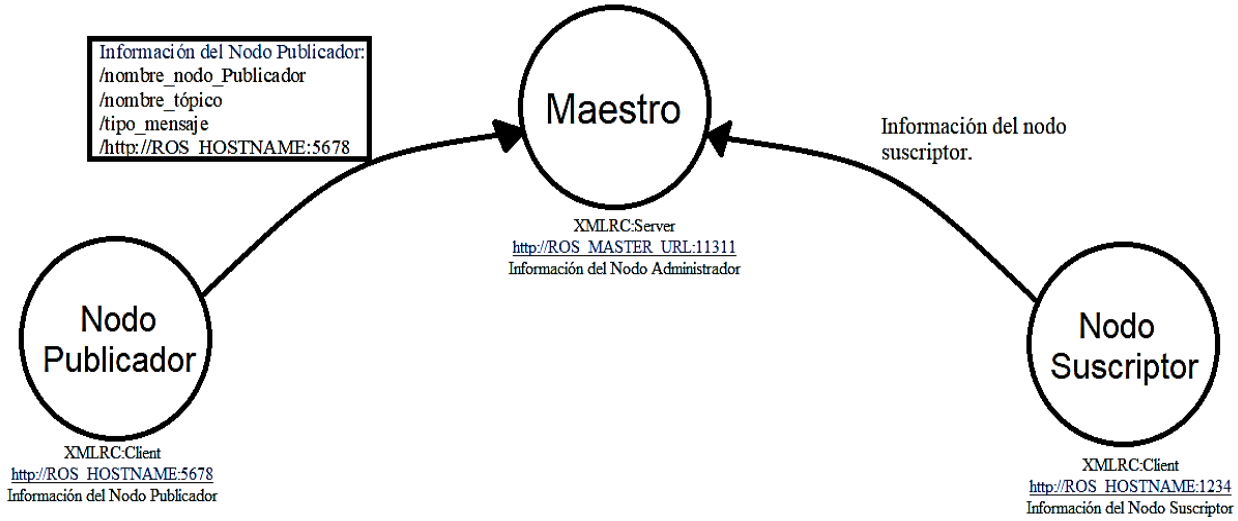

Figura 24: Corriendo el nodo Publicador.

Proporcionando la información del nodo publicador.

El maestro distribuye la información como nombre del publicador, el nombre del tópico, el tipo de mensaje, la dirección URL y el número del puerto a todos los nodos suscriptores que quieren conectarse a este nodo publicador Figura 25.

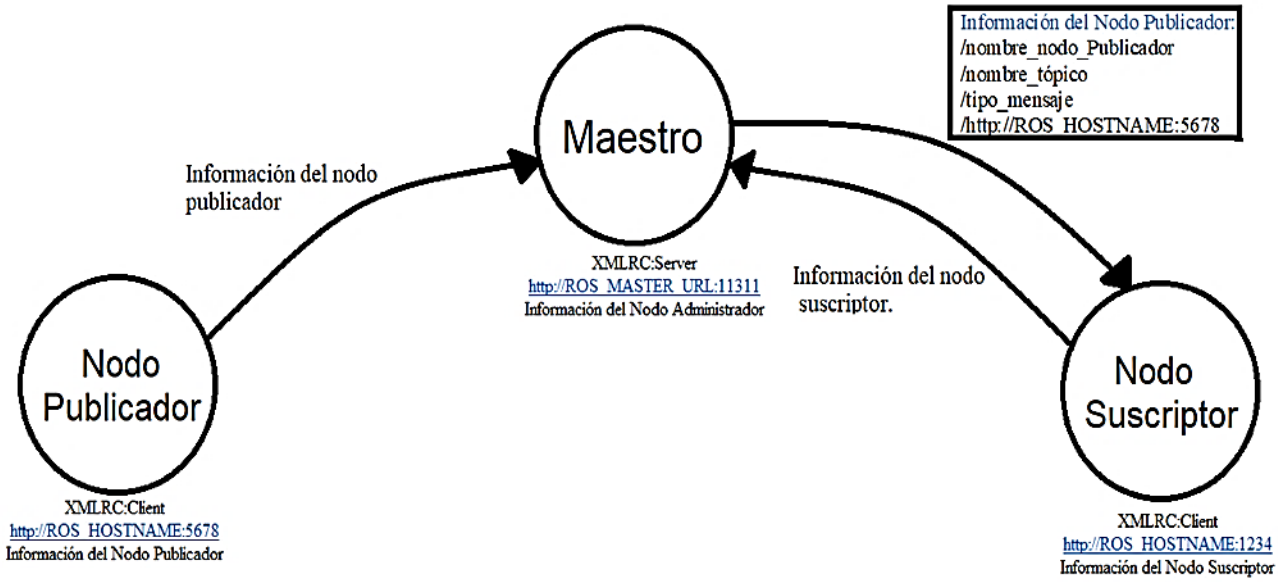

Figura 25: Proporcionado información acerca del nodo publicador.

Respuesta de conexión desde el nodo suscriptor.

El nodo suscriptor solicita una conexión directa con el nodo publicador en base a la información recibida del maestro. Durante el proceso de solicitud, el nodo suscriptor transmite información al nodo publicador como el nombre del nodo suscriptor, el nombre del tópico y el tipo de mensaje Figura 26.

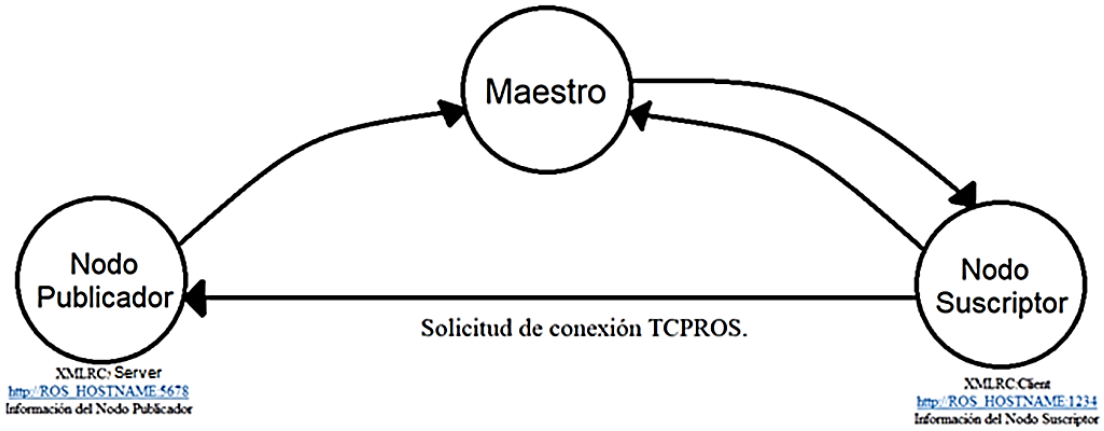

Figura 26: Solicitud de conexión desde el nodo suscriptor.

Respuesta de conexión desde el Nodo Publicador.

El nodo publicador envía la dirección URL y el número de puerto de su servidor en respuesta a la solicitud de conexión del nodo suscriptor (ver Figura 27).

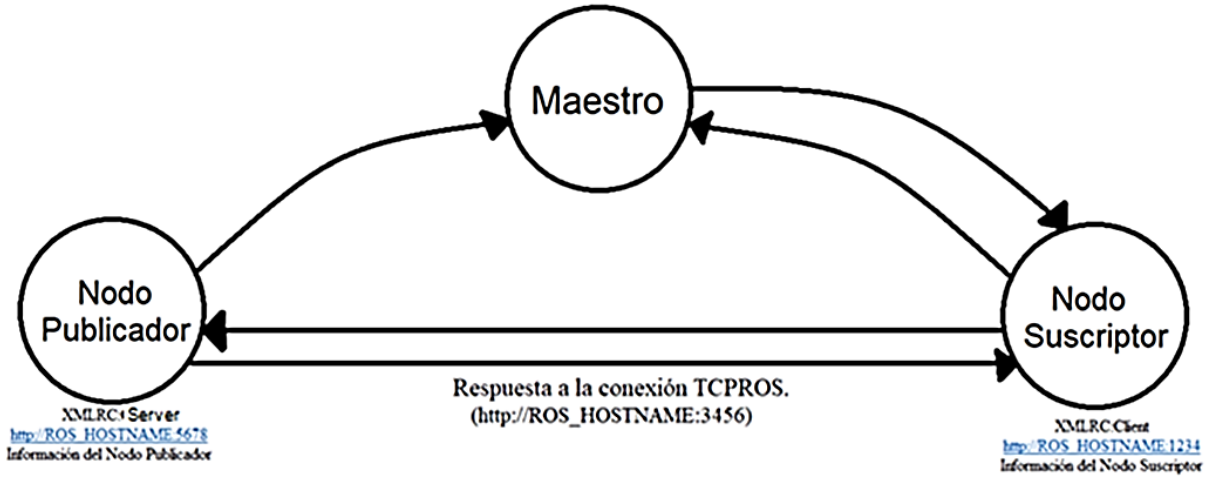

Figura 27: Respuesta a la conexión desde el nodo publicador.

Conexión TCPROS.

El nodo suscriptor crea un cliente para el nodo publicador usando el TCPROS, y conecta al nodo publicador. En este punto la comunicación entre nodos usa un protocolo llamado TCPROS basado en TCP/IP. (Ver Figura 28).

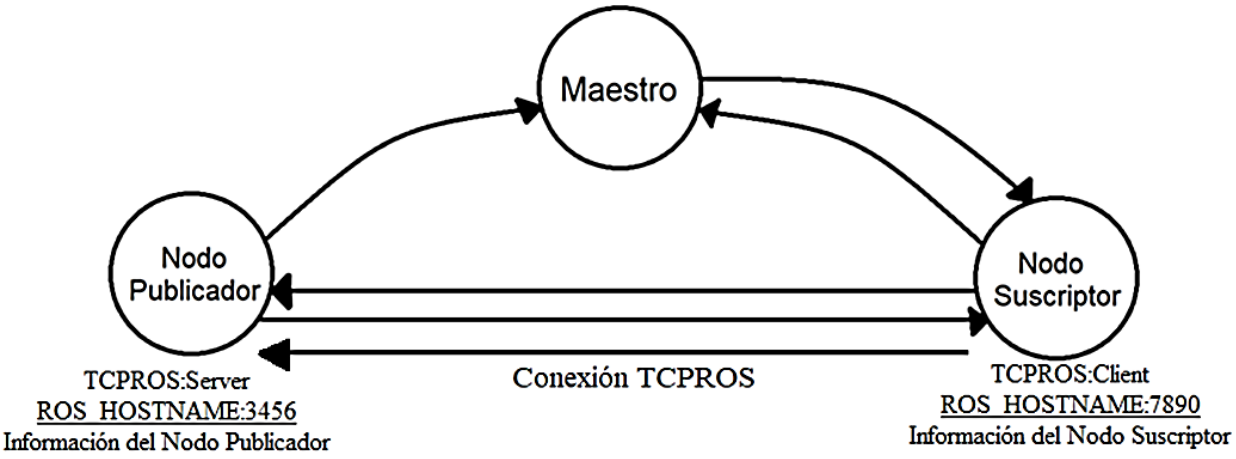

Figura 28: Conexión TCPROS.

Transmisión de mensajes.

El nodo publicador transmite un mensaje predefinido al nodo suscriptor. La comunicación entre nodos usa el protocolo TCPROS. (Ver Figura 29).

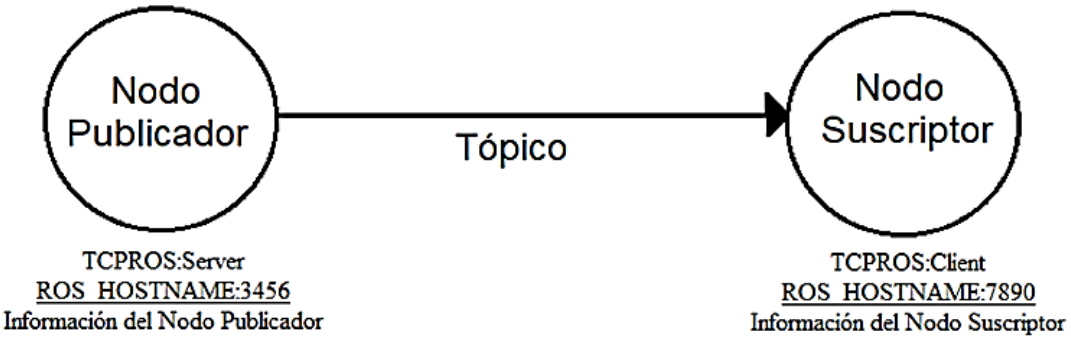

Figura 29: Tópico transmisión de mensajes.

A continuación, se muestra un ejemplo de cómo es el flujo de acciones para el paso de un mensaje desde el nodo publicador a un nodo suscriptor y todo dirigido por nodo. (Ver Figura 30).

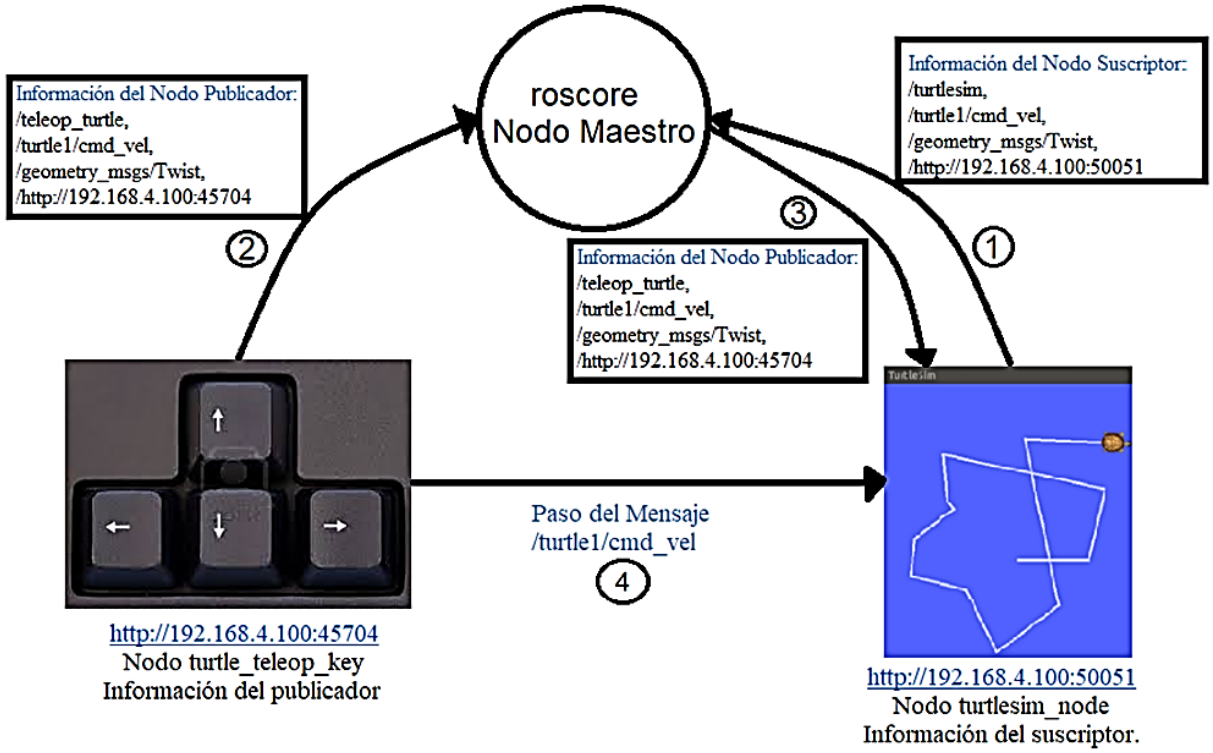

Figura 30: Ejemplo de comunicación de un mensaje en ROS.

3.5.8 Rosserial.

El rosserial es un paquete que convierte mensajes de ROS y tópicos que se utilizan en comunicación serial. Los microcontroladores utilizan la comunicación UART en lugar del protocolo TCP/IP el cual es el que utiliza por defecto ROS, por lo que hay convertir la comunicación de mensajes entre un microcontrolador y una computadora utilizando ROS, rosserial debería interpretar mensajes para cada dispositivo.

En un computador con ROS corre un servidor de rosserial, y el microcontrolador conecta a la computadora convirtiéndose un cliente rosserial. Ambos tanto el servidor como el cliente reciben datos usando el protocolo rosserial, esto hace posible que cualquier hardware pueda enviar y recibir datos atreves de los mensajes de ROS con UART (ver Figura 31).

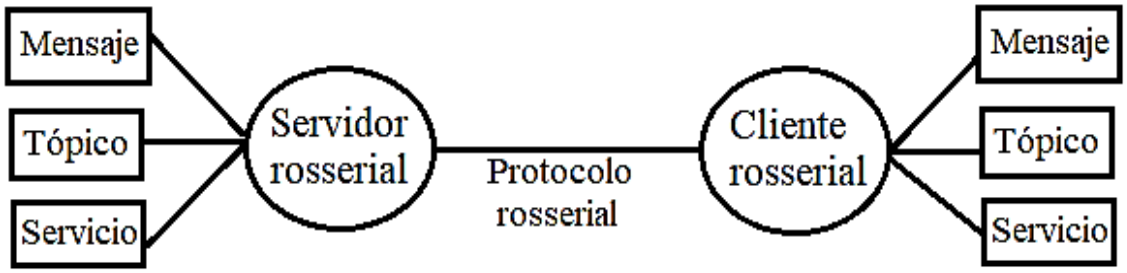

Figura 31: Servidor rosserial (Computadora con ROS) y Cliente rosserial (Sistema Embebido).

Servidor rosserial.

El servidor rosserial es un nodo el cual realiza la comunicación entre una computadora corriendo ROS y un sistema embebido. Dependiendo del lenguaje de programación se puede emplear uno de los siguientes nodos.

- **rosserial_python:**
  Este nodo se implementa con el lenguaje Python y es muy usado.

- **rosserial _server:**
  Aunque se ha mejorado el rendimiento para el lenguaje C++, todavía presenta algunas limitaciones funcionales en comparación con rosserial_python.

- **rosserial_java:**
  La librería rosserial_java es usada cuando módulos basados en java son requeridos, frecuentemente usado con el SDK de Android.

- **Cliente rosserial:**
  La librería rosserial_client es cargada al microcontrolador de la plataforma embebida para que esta pueda ser utilizada como cliente. Esta librería está extendida a varias plataformas embebidas de desarrollo se muestran a continuación:
- **rosserial_arduino:**
  Esta librería está diseñada para plataforma Arduino y soporta las tarjetas Arduino UNO, Leonardo y pude ser utilizada en otras plataformas mediante modificaciones del código fuente.

- **rosserial_windows:**
  Esta librería es soportada por el sistema operativo Windows y puede comunicarse con aplicaciones de Windows.

- **rosserial_tivac:**
  Esta librería es usada para la tarjeta Launchpad manufacturada por Texas Instruments.

Protocolo rosserial.

El servidor rosserial y el cliente rosserial envían y reciben datos por medio de paquetes basados en la comunicación serial. El protocolo rosserial está definido a nivel de bytes y contiene información para la sincronización de paquetes y validación de datos.

Configuración de paquetes.

Los paquetes de rosserial incluyen el campo de encabezado para el envío y recepción de datos en el mensaje estándar de ROS y un campo para la suma control y validación de datos. (Ver Tabla 12).

1º Byte	2º Byte	3º Byte	4º Byte	5º Byte	6º Byte	7º Byte	N Byte	Byte N+8
Bandera de Sincronización	Bandera de Sincronización /Versión del protocolo	Tamaño del mensaje(N)	Tamaño del mensaje(N)	Suma de Control del Tamaño del Mensaje	ID del Tópico	ID del Tópico	Datos del mensaje serializado.	Suma de control del ID del tópico y de los datos del mensaje.
	(Valor 0xFE)	(Low Byte)	(High Byte)		(Low Byte)	(High Byte)		

Tabla 12: Configuración de paquetes rosserial.

Bandera de Sincronización:

Este byte siempre tiene el valor 0xFF e indica el inicio del paquete.

Bandera de Sincronización/Versión del protocolo:

Este campo indica que versión del protocolo de ROS, donde Indigo y Kinetic son 0xFE.

Tamaño del mensaje:

Estos 2 bytes indican el tamaño de los datos del mensaje transmitido atreves del paquete. El byte menos significativo viene primero y le sigue el byte más significativo.

Suma de control sobre el tamaño del mensaje.

La suma de control verifica la validación del tamaño de mensaje y se calcula como:
255-((Tamaño del mensaje (Low Byte) +Tamaño del mensaje (High Byte)) %256)

ID del Tópico.

El campo ID consta de dos bytes que son utilizados pata distinguir el tipo de mensaje, Los ID de tópicos de 0 a 100 están reservados para funciones del sistema.

Datos de mensajes serializados.

Este campo de datos contiene mensajes serializados.

Checksum del ID topic.

Esta suma es para la comprobación y validación de la ID del tópico y los datos del mensaje, y se calcula de la siguiente forma.
255-((Low Byte ID tópico + High Byte ID tópico + byte datos valores) % 256)

Paquete de consulta.

Cuando el rosserial servidor inicia, esto requiere información como el nombre del tópico, nombre y tipo de cliente. Cuando solicitamos información, se utiliza el paquete de consulta. El id de tema del paquete de consulta es 0 y el tamaño de los datos es 0.

Los datos en el paquete de consulta se muestran a continuación.
0xff 0xfe 0x00 0x00 0xff 0x00 0x00 0xff

Cuando el cliente recibe el paquete de consulta, esta manda un mensaje al servidor con la siguientes:
uint16 topic_id
string topic_name
string message_type
string md5sum
int32 buffer_size

Capítulo 4
Metodología.

La realización de este proyecto se dividió en cuatro bloques generales a desarrollar establecidos de la siguiente forma:

- Analizar el sistema ya implementado para la detección de la dirección fuente sonora en el robot.
- Investigación y selección de un nuevo método para establecer el origen de la fuente del sonido.
- Adaptación e implementación del mecanismo seleccionado al sistema ya existente.
- Pruebas y depuración de los errores presentados en la implementación del mecanismo seleccionado.

4.1 Análisis del sistema ya implementado.

El robot de servicio tiene integrada una placa electrónica con una tarjeta de desarrollo Arduino Uno y cuatro sensores de sonido FC-04, estos cuentan con un micrófono de condensador, una etapa de amplificación y otra de comparación; al incidir una señal de sonido sobre un micrófono se presenta una valor de 0V en su pin OUT, solo si esta señal sobrepasa el nivel de umbral ajustado por medio del potenciómetro en la etapa de comparación , si no sobrepasa este valor o no existiera algún impacto de sonido sobre algún micrófono el valor presente seria de 5v en el pin OUT. Los micrófonos se encuentran distribuidos en la cabeza del robot como se muestra en la Figura 32.

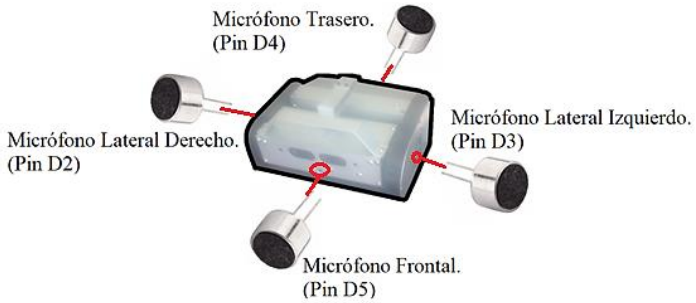

Figura 32: Posición de sensores en la cabeza del Markovito.

La determinación de la dirección de la fuente de sonido se realiza por medio de conteos, cuando una onda impacta sobre algún micrófono produce un cambio de nivel de voltaje de 5V a 0v en el pin en el que está conectado (ver Figura 33); este cambio se registra como un incremento en el conteo para cada uno de los pines conectados a un micrófono, se toman muestras por un determinado tiempo, iniciando cuando recibe una señal START por medio de la suscripción a un tópico de ROS. El pin que registre el mayor conteo es la dirección donde procede el sonido. Una vez determinada esta dirección es publicada a un tópico de ROS, para su posterior uso en la prueba HEAR AND ROTATE la cual hace girar al robot hacia la dirección del origen del sonido.

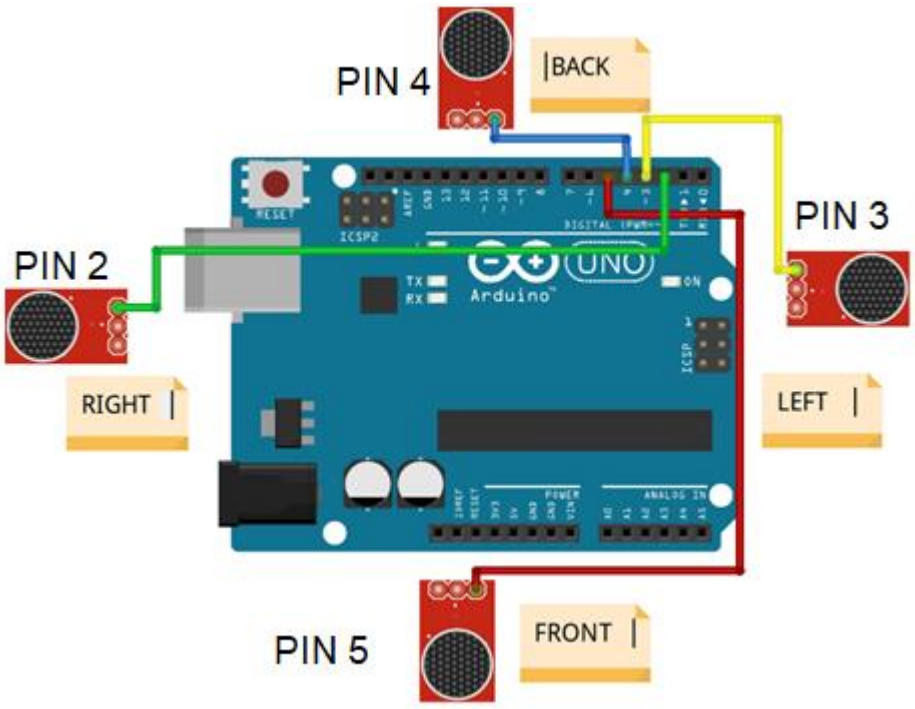

Figura 33: Conexión de los sensores de sonido a la tarjeta Arduino UNO.

Haber realizar un análisis del sistema implementado genero una perspectiva de cómo ser abordo el problema anteriormente, la forma de emplear los sensores para registrar la presencia de sonido, el método de procesamiento de estos datos con Arduino UNO, para la determinar el origen del sonido y su posterior publicación al robot.

4.2 Investigación y selección de un nuevo método para establecer el origen de la fuente del sonido.

4.2.1 Investigación de los diversos métodos para determinar el origen del sonido.

Se realizó una investigación sobre los métodos para la determinación de la dirección de la fuente de sonido, dentro los métodos que destacan son:

- *El método por de diferencia de intensidades*: Es un método que mide la energía entre dos señales en un determinado momento, esto puede ser útil para determinar si la fuente proviene del frente, atrás, derecha o izquierda, este método nos probé una determinación de cuatro posibles orientaciones, este método es el empleado en la anterior versión del robot para determinar la fuente de sonido.
- *El método de la diferencia del tiempo de arribo (TODA):* En este método se emplea el tiempo de diferencia en la incidencia de una onda de sonido a un arreglo de dos micrófonos separados por una distancia determinada, nos permite por medio de una correlación entre las señales determinar el origen del sonido.

4.2.2 Selección de un método para determinar la dirección de una fuente sonora.

El método que se seleccionó para su implementación en el sistema ya existente es el método de la diferencia del tiempo de arribo (TDOA Time Difference Of Arrive), la selección de este método se realizó en base a dos criterios primordialmente, primero que fuera posible su implementación con el hardware ya existente, y segundo que este sistema ofreciera mayor resolución en las posibles estimaciones de la dirección de la fuente del sonido, una vez lograda su correcta implementación.

La implementación del método TODA para la localización del sonido en este hardware se determinó como viable debido a que:

- La señal de sonido se encuentra normalizada, es decir si impacta una onda de sonido en el micrófono del sensor se proporcionan 0V a la salida de sensor y en caso de que no exista impacto de sonido en el micrófono, este sensor proporciona a su salida 5V; debido a esto en la presencia de sonido se registra

como un cambio de niveles de voltaje entre 0 y 5V. En la figura 34 se muestra la señal generada por el sensor FC-04, al impactar una onda de sonido en el micrófono del sensor.

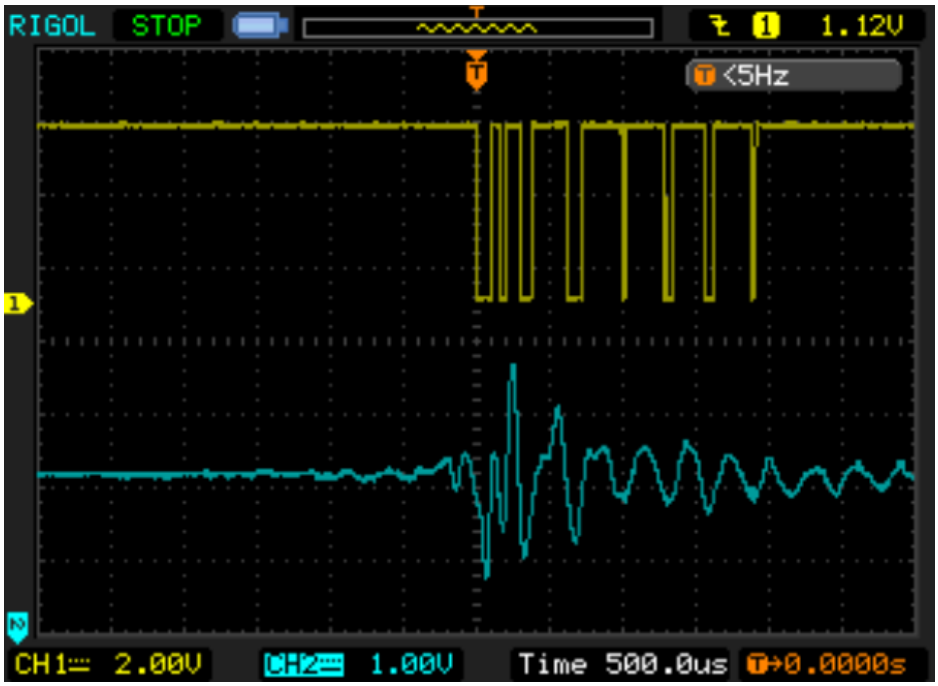

Figura 34: Captura en un osciloscopio de la señal generada por el impacto de sonido en el sensor FC-04.

Al estar normalizada la señal de sonido, para poder determinar el arribo de una onda sonido a un micrófono basta con solo detectar el primer flanco de bajada de la señal producida por el sensor (ver Figura 34), para saber que el micrófono comenzó a detectar sonido. Esto ayuda al microcontrolador evitando que tenga que procesar la señal de sonido de forma analógica por medio de un ADC, demandando una mayor cantidad de recursos como memoria y tiempo de ejecución.

- EL microcontrolador Atmega328p de 8bits con el que cuenta la tarjeta Arduino Uno, trabaja a una frecuencia de reloj de 16 MHz, esta frecuencia de trabajo es suficiente para poder utilizar un timer y medir la diferencia del tiempo entre el primer arribo de la onda de sonio al micrófono más cercano y el segundo arribo al segundo micrófono más cercano. Esta diferencia medida se emplea en la estimación de la dirección de la fuente del sonido.

La selección de este método, en lugar del método de intensidades radica en la resolución de direcciones para determinar el origen de la fuente, el método de intensidades ofrece cuatro posibles direcciones enfrente (0 rad), atrás (π rad), derecha ($-\frac{\pi}{2}$ rad) e izquierda ($\frac{\pi}{2}$ rad) para los cuatro micrófonos. El método TODA, ofrece una rango de mediciones por medio de cuadrantes del micrófono frontal al izquierdo de 0 a $\frac{\pi}{2}$ rad, del micrófono izquierdo al trasero de $\frac{\pi}{2}$ a π rad, del micrófono trasero al derecho de $-\pi$ a $-\frac{\pi}{2}$ rad y del micrófono derecho al frontal de $-\frac{\pi}{2}$ a 0 rad (ver Figura 35), esto otorga un rango de mediciones muy variado para localizar la fuente de sonido.

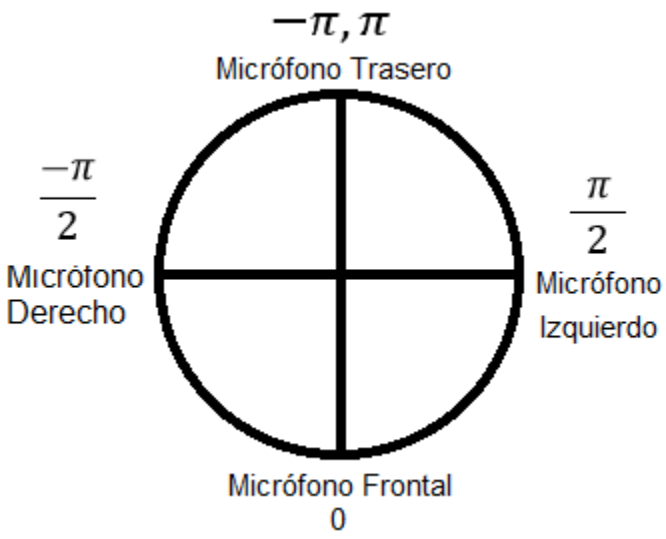

Figura 35: Cuadrantes para determinar la dirección del sonido respecto al robot.

4.3 Adaptación e implementación del mecanismo seleccionado al sistema ya existente.

La idea general que se planteó para implementar con la placa Arduino Uno y los sensores de sonido FC-04 es, crear un nodo en el microcontrolador que se suscriba al tópico "/isdf/isdf_ss" este tópico indica cuando iniciar a adquirir datos de sonido por medio de los sensores y procesarlos en microcontrolador Atmega328p con el método TDOA, una vez determinada la dirección de la fuente de sonido el nodo debe publicarla al tópico "/isdf/isdf_pos" para su posterior uso por un nodo que realizará el giro del robot en orientación de la fuente emisora de sonido. La parte que se desarrolló este

proyecto es la está involucrada con el nodo de Arduino (ver Figura 36), la sección delimitada por el recuadro rojo.

Figura 36: Diagrama del Nodo a desarrollar en Arduino.

4.3.1 Desarrollo de Nodo de Arduino UNO.

¿Cómo determinamos el orden de arribo a los micrófonos?

Los cuatro sensores de sonido FC-04 se conectaron en el puerto D del microcontrolador Atmega328p, el sensor frontal se conectó al pin D5, el sensor trasero se conectó en el pin D4, el sensor de sonido izquierdo se conectó en el pin D3 y el sensor de sonido derecho se conectó en el pin D2. Para poder determinar el orden de arribo de la onda de sonido a los micrófonos se empleó un recurso de muy utilizado en los microcontroladores las interrupciones, cuya función es el de interrumpir el flujo de programa principal al ocurrir algún un evento determinado, ya sea un flanco descendente o ascendente en algún pin seleccionado, para después ejecutar una rutina con una serie de instrucciones que atiende a este evento y una vez terminadas regresar al programa principal. El tipo de interrupciones seleccionadas fueron las interrupciones por cambio de estado (PCI), debido a que estas interrupciones se pueden emplear en los cuatro pines conectados a los micrófonos, a diferencia de las interrupciones externas que únicamente detectan flancos de subida o de bajada, las cuales solo se encuentran presentes en los pines D2 y D3 (ver Tabla 13).

Pin Arduino	Micrófono conectado.	Tipo de interrupción disponible en el pin .
D2	Lateral derecho	Externa por flaco de subida, flanco de bajada (INT0) y cambio de estado del pin (PCI18).
D3	Lateral izquierdo	Externa por flaco de subida, flanco de bajada (INT1) y cambio de estado del pin (PCI19).
D4	Trasero	Cambio de estado del pin. (PCI20)
D5	Frontal	Cambio de estado del pin. (PCI21)

Tabla 13: Conexión de pines a los micrófonos.

Las interrupciones por cambio de estado se seleccionan por medio de puertos, en este caso se empleó la del puerto D, para activar su rutina de interrupción del puerto D, se debe establecer a 1 el bit 2 el registro PCICR. Si se desea activar o desactivar una interrupción de manera individual para cada pin del puerto es necesario aplicar una máscara en el registro PCMSK2 de pendiendo de a la interrupción a emplear, realizando esto tenemos control de cuándo y cuales interrupciones de los micrófonos están habilitadas.

Se desarrolló una rutina de interrupción que se ejecuta cada vez que ocurre un cambio de estado, en cualquiera de los pines conectados a un micrófono, esta rutina solo se activara si el nodo de Arduino Uno se ha suscripto al tópico "/isdf/isdf_ss” y este tiene un mensaje publicado. La rutina desarrollada nos permite guardar el número de pin del primer micrófono en ser impactado, iniciar a contar el tiempo en microsegundos, una vez que el segundo micrófono es impactado se guarda el número de pin al que está conectado y el tiempo que ocurrió, para después deshabilitar las PCI y detener el conteo de tiempo hasta que todos los micrófonos ya no registran sonido.

¿Cómo medimos el tiempo en microsegundos?

Para lograr medir el tiempo de diferencia de arribo entre el primer micrófono en ser impactado y el segundo micrófono impactado cuyo valores están en el rango de los microsegundos, se empleó la interrupción por timer para generar conteos de tiempos, el timer empleado es el timer 2 el cual tiene el registro TCNT2 de 8 bits para su conteo, su valor máximo de conteo es 255 antes de producir un desborde, la señal de reloj se empleó con un prescaler de 1, usando así los 16 MHz de la frecuencia de reloj por lo que el registro TCNT2 se desborda cada 16 μs, una vez que ocurre la interrupción por desborde del registro TCNT2 se incrementa un contador de 16 bits para tener una mayor rango de medición de tiempos. Para lograr la medición del tiempo actual en microsegundos se emplea la siguiente ecuación.

$$Tiempo\ Actual\ Microsegundos\ = ((\text{Contador de 16 bits} \ll 8) + \text{TCNT2}\ )/16$$

Donde:

Tiempo actual → Es el tiempo en microsegundos, desde que se inició el timer hasta que se realiza la medición:

Contador de 16 bits → Es un contador que guarda el número de desbordes en el registro TCNT2 han ocurrido.

TCNT2→El registro de 8 bits de timer 2 que se incrementa con una señal de reloj de 16MHz.

Al contador de 16 bits para desbordes del registro TCNT2 se le realiza un corrimiento de 8 bits a la izquierda y se le añaden los 8 bits del registro TCNT2 por lo que se llevaría un conteo del tiempo realizado a una frecuencia de 16 MHz, dividiendo el resultado entre 16 se obtuvo un conteo de tiempos a una frecuencia de 1MHz, lo que es equivalente a medir el tiempo en microsegundos.

¿Cómo determinamos la dirección de la fuente de sonido?

La forma que se planteó resolver el problema de estimar la dirección de la fuente de sonido se dividió en dos pasos:

En primera parte se debe establecer cual es orden de arribo, por lo que se necesita determinar cuál es el primer micrófono en ser impactado por una onda de sonido y cuál es el segundo micrófono en ser impactado por la onda del sonido. Una vez obteniendo

esta información ya es posible estimar en que cuadrante se encuentra la fuente de sonido (ver Tabla 14).

Orden de arribo.	Cuadrante de Localización.
Primero es el frontal y el segundo es el izquierdo. Primero es el izquierdo y el segundo es el frontal.	$0 \quad a \quad \pi/2$
Primero es el izquierdo y el segundo es el trasero. Primero es el trasero y el segundo es el izquierdo.	$\pi/2 \quad a \quad \pi$
Primero es el trasero y el segundo es el derecho. Primero es el derecho y el segundo es el trasero.	$-\pi \quad a \quad -\pi/2$
Primero es el derecho y el segundo es el frontal. Primero es el frontal y el segundo es el derecho.	$-\pi/2 \quad a \quad 0$

Tabla 14: Cuadrante de localización por orden de arribo.

La segunda parte se trata de emplear la diferencia de tiempo entre que es impactado el primer micrófono por la onda de sonido y el segundo micrófono es impactado, para calcular un ángulo en radianes con la siguiente ecuación:

$$\theta = \sin^{-1}(\frac{\Delta t * v}{d})$$

$\theta \rightarrow$ ángulo en radianes calculado por tiempo de diferencia de arribo.

$\Delta t \rightarrow$Es la diferencia de tiempo entre el primer micrófono impactado y el segundo micrófono impactado.

$v \rightarrow$ Es la velocidad del sonido (343m/s).

$d \rightarrow$Es la distancia en metros entre el primer micrófono impactado y el segundo micrófono impactado las distancias entre micrófonos pueden encontrarse en la Tabla 15.

Micrófonos	Distancia (m).
frontal y derecho	0.1805
frontal e izquierdo	0.1805
Trasero y derecho	0.155
Trasero e izquierdo	0.159

Tabla 15: Distancias entre micrófonos.

Una vez calculado el ángulo θ ya podemos

aproximar la dirección de la fuente de sonido. A la estimación generada por el orden de arribo se le añadió el ángulo θ para así obtener el ángulo de la dirección de sonido en los cuadrantes (ver Tabla 16).

Orden de arribo.	Estimación por orden de arribo.	Ángulo de la dirección de sonido en los cuadrantes
Primero es el frontal y el segundo es el izquierdo.	0	$0+\theta$
Primero es el izquierdo y el segundo es el frontal.	$\frac{\pi}{2}$	$\frac{\pi}{2}-\theta$
Primero es el izquierdo y el segundo es el trasero.	$\frac{\pi}{2}$	$\frac{\pi}{2}+\theta$
Primero es el trasero y el segundo es el izquierdo.	π	$\pi-\theta$
Primero es el trasero y el segundo es el derecho.	$-\pi$	$-\pi+\theta$
Primero es el derecho y el segundo es el trasero.	$-\frac{\pi}{2}$	$-\frac{\pi}{2}-\theta$
Primero es el derecho y el segundo es el frontal.	$-\frac{\pi}{2}$	$-\frac{\pi}{2}+\theta$
Primero es el frontal y el segundo es el derecho.	0	$0-\theta$

Tabla 16: Estimación del ángulo de la dirección de sonido en los cuadrantes.

Después de haber obtenido la dirección de sonido esta se publica al tópico "/isdf/isdf_pos" para su posterior uso por otros nodos, después de hacer la publicación, el nodo se reinicia para esperar que en el tópico "/isdf/isdf_ss" exista un mensaje publicado y empezar todo su proceso otra vez.

4.4 Pruebas y depuración de los errores presentados en la implementación del mecanismo seleccionado.

Para la realización de pruebas del correcto funcionamiento del sistema, se creó una pequeña versión con arreglo de sensores de sonido (ver Figura 37), en la cual se llevó a cabo la depuración de errores presentes. Para poder analizar el comportamiento del sistema no solo fue necesario publicar el ángulo de la dirección del sonido en los cuadrantes, sino que también fue necesario la publicación de datos como el orden de llegada de la onda de sonido a los micrófonos y la diferencia en el tiempo de arribo. Atreves de estos datos poder saber que parte o sección del programa generado no se encontraba funcionando de manera correcta. Para la realización esta prueba se empleó un Arduino Nano, esta tarjeta cuenta con el mismo microcontrolador Atmega328p que es con el que cuenta la tarjeta de desarrollo Arduino Uno simplemente varía el tipo de encapsulado del microcontrolador, se decidió usar la tarjeta Nano ya que es ideal para colocarla en un protoboard.

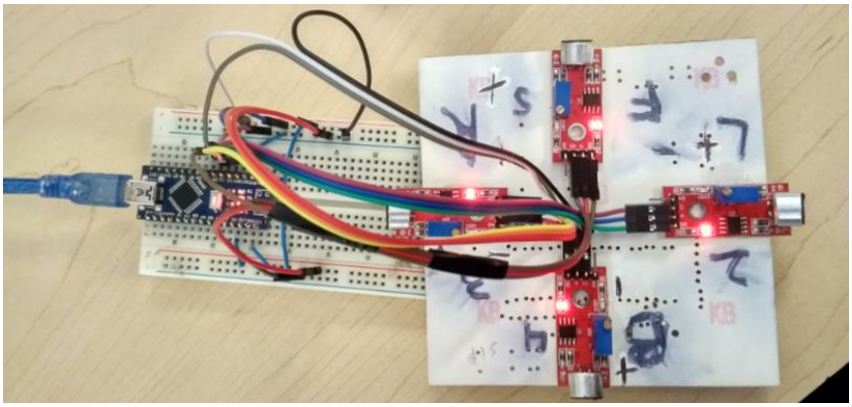

Figura 37: Sistema empleado para pruebas.

La primera prueba estaba enfocada en verificar que el programa determinara de manera correcta el orden de arribo de la onda de sonido, para poder saber si los datos eran correctos se visualizaron por medio de la terminal un con echo al tópico "/isdf/isdf_pos" es el tópico en que el nodo publica los datos obtenidos de una muestra de sonido (ver Figura 38).

```
angel@angel-HP-EliteBook-8470p:~$ rostopic echo /data: start
---
data: stop
---
```

```
data: FIRST: 2
---
data: SECOND: 3
---
data: THIRD: 5
```

Figura 38: Visualización de orden de arribo por medio de consola.

Al realizar las pruebas de sonido, ocurrió un fenómeno muy particular al iniciar las primeras pruebas eran exitosas en un 90% de las muestras tomadas, pero tiempo después pasando una hora de pruebas los sensores de sonido comenzaban a realizar un fenómeno muy particular se disparaban automáticamente, esto se debía al calentamiento que ocurría del transistor Q1 (ver Figura 39), que provocaba disparando en su corriente de colector, lo que generaba un desajuste en el nivel de umbral establecido, causando un disparo a un nivel se sonido diferente de los de más micrófonos.

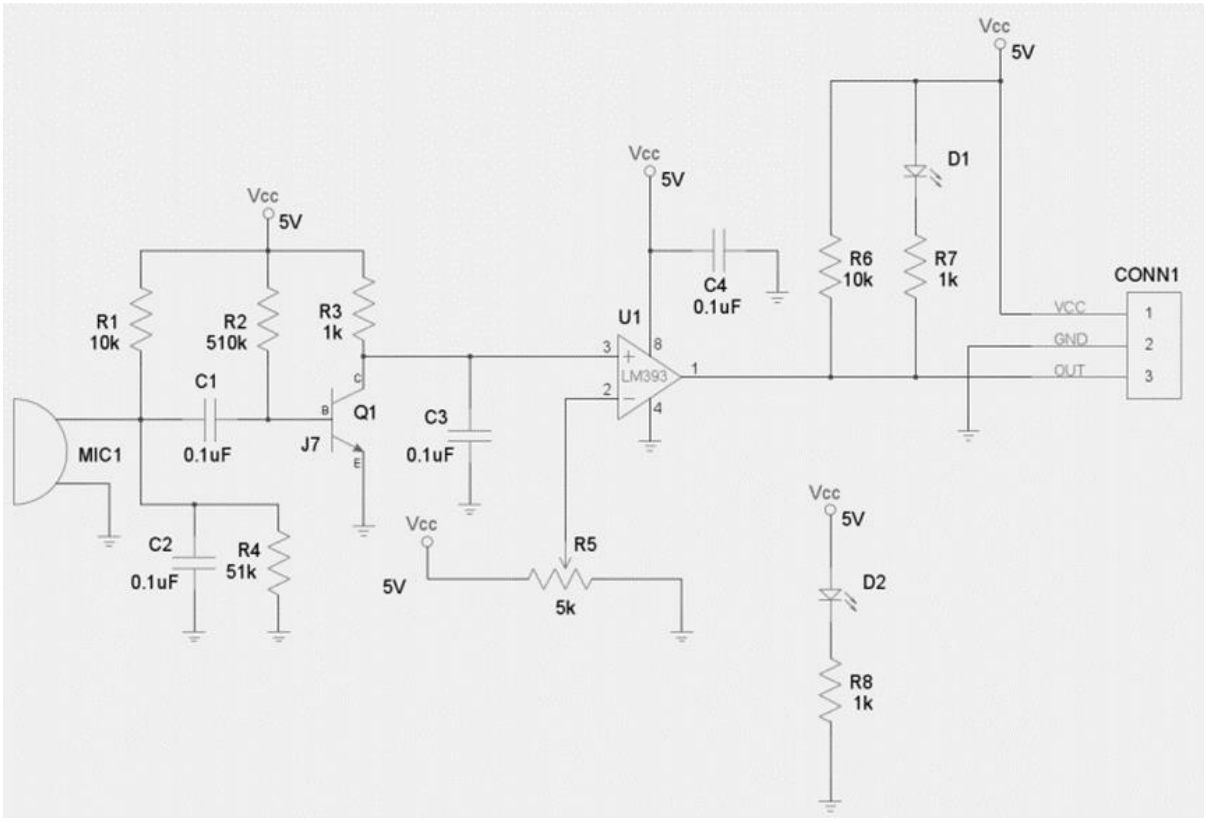

Figura 39: Diagrama eléctrico del sensor FC-04.

Las pruebas posteriores se concentraron en verificar que el tiempo de la diferencia de arribo era estimado correctamente, en el desarrollo de esta prueba es donde se presentaron las mayores complicaciones. Para la visualización de los datos se recurrió a la terminal, en la cual se mostraban datos como el orden de arribo más la diferencia de tiempo de arribo (ver Figura 40).

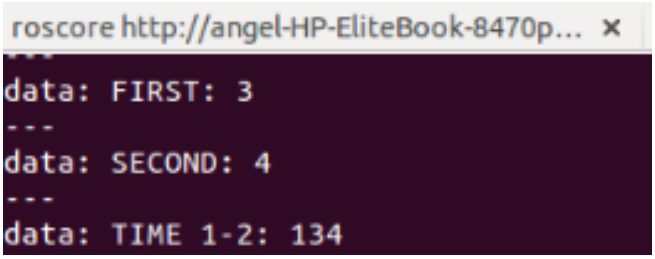

Figura 40: Visualización de la diferencia de tiempo de arribo.

El problema que se presentó en esta prueba es que la estimación de la diferencia de tiempo de arribo que en muchas muestras era más grande comparado al valor máximo que debería presentarse (ver Tabla 17), este se calcula de la siguiente forma:

$$Tiempo_{máx} = \frac{distancia_{1-2}}{velocidad_{sonido}}$$

Donde:

$Tiempo_{máx}$→ Es el tiempo de diferencia de arribo máximo que se puede producir.

$distancia_{1-2}$→Es la distancia entre el primer micrófono en ser impactado y el segundo en ser impactado.

$velocidad_{sonido}$→Es la velocidad del sonido en el aire cuyo valor es de 343 m/s.

Micrófonos	Diferencia de tiempo máximo de arribo que puede ocurrir.
Frontal e Izquierdo	526 μs
Frontal y Derecho	526 μs
Trasero e Izquierdo	469 μs
Trasero y Derecho	452 μs

Tabla 17: Diferencia de tiempo máxima que puede ocurrir por combinación de micrófonos.

En este punto se tenían dos posibilidades que el programa no funcionara correctamente o que los sensores presentaran algún problema en su funcionamiento. Para poder determinar si el programa era capaz de medir correctamente las diferencias de tiempo en el rango de 0 a 526 μs se diseñó un experimento que permitiera validar el programa, se emplearon dos microcontroladores ver Figura 41, un microcontrolador servía como nodo para publicar el orden de arribo y la diferencia del tiempo de arribo a los dos primeros pines y un segundo microcontrolador se empleó como generador de pulsos, estos se conectaron en lugar de los sensores de sonido para simular la señal de sonido normalizada.

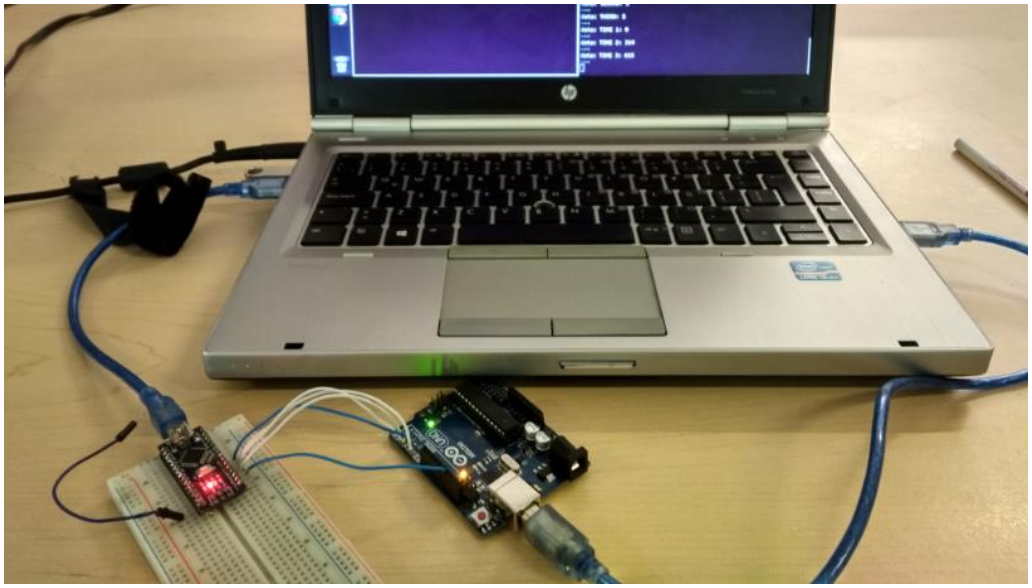

Figura 41: Experimento para validar programa en la medición de la diferencia de tiempo de arribo.

Las señales generadas en el segundo microcontrolador siguen la siguiente secuencia, inicialmente las cuatro señales se encuentra en 5v, esperábamos hasta que ocurra la señal de inicio simulando el comienzo del sonido, después comenzábamos a contar el tiempo en microsegundos hasta que se igualara el tiempo de prueba propuesto una vez ocurrido esto cambiamos el estado de la primera señal a 0v, después otra vez iniciábamos a contar el tiempo en microsegundos hasta que se igualara el tiempo de prueba propuesto una vez ocurrido esto cambiamos el estado de la segunda señal a 0v y repetimos el proceso para las otras dos señales. En la ejecución del experimento se varió el tiempo de prueba de 1 a 1500 μs, examinado las mediciones y

comparándolas con diversos tiempos de prueba se determinó que se podían medir tiempos con mucha precisión a partir de tiempos mayores a los 10 μs, medidas menores a este tiempo presentaban errores de precisión, a partir de los 10 μs estos errores de precisión comenzaban a disminuir.

Una vez que logramos validar que el programa es capaz de medir las diferencias de tiempo de 15 μs a 526 μs, se procedió a realizar un análisis de los sensores de sonido FC-04. Se utilizó un osciloscopio para visualizar las señales generadas por el impacto de sonido en los micrófonos, al realizar las mediciones con el osciloscopio logramos visualizar la diferencia del tiempo de arribo que ocurre entre los dos micrófonos el cual se manifestó como un desfase de las señales generadas, pero también ayudo observar una disminución en la amplitud de la señal generada en el segundo micrófono debido a que la onda pierde energía conforme se propaga en el medio (ver Figura 42), este fenómeno de perdida de amplitud de la señal es la fuente del problema de no poder determinar correctamente la diferencia del tiempo de arribo, puesto que los sensores tienen un umbral de voltaje para detectar presencia de sonido , si este no es rebasado no cambiara el estado de la salida de 5V a 0V, por lo que la señal al ser reducida en su amplitud por perdida de energía, podría no rebasar el umbral y pasar sin ser detectada hasta que una la amplitud de la señal rebase el umbral.

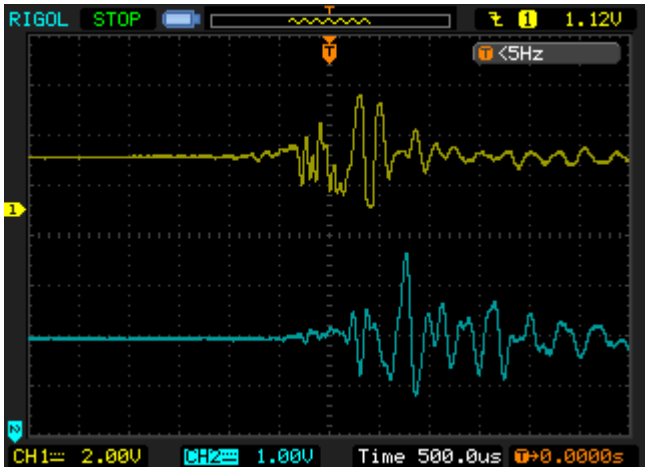

Figura 42: Visualización de las señales analógicas generadas por los micrófonos.

Una solución para poder tratar con este problema es tomar más de una muestra de la diferencia del tiempo de arribo entre el primer micrófono en ser arribado y el segundo micrófono en ser arribado, mientras una onda de sonido este impactando en los micrófonos, una vez adquiridas las muestras se validan las que son iguales o menores a la diferencia de tiempo máximo de arribo que puede ocurrir, estas se promedian y ocupamos este valor como la diferencia de tiempo de arribo promedio.

Capítulo 5

Resultados.

Como producto final para integración, se generó un nodo (un programa) de ROS en Arduino Uno, que se suscribe a un tópico de ROS el cual le indica por medio de un mensaje publicado el momento en el que el que el nodo de Arduino deberá comenzará adquirir datos de una muestra de sonido y utilizando el método de la diferencia del tiempo de arribo estimar la dirección de la fuente de sonido, para su posterior publicación a un tópico y esta dirección pueda ser usada por otros nodos.

A lo largo del desarrollo del nodo, se generaron diversos subprogramas para realizar tareas específicas (ver Figura 43), que en conjunto generaran un solo programa capaz de llevar a cabo todos los procesos para los que fue diseñado el nodo, estos programas se unieron por medio de una librería llamada “sensor_sound” facilitando mucha la integración de todos los programas desarrollados y el uso de estos, para poder visualizar el código del nodo y la librería desarrollada ir al anexo A.

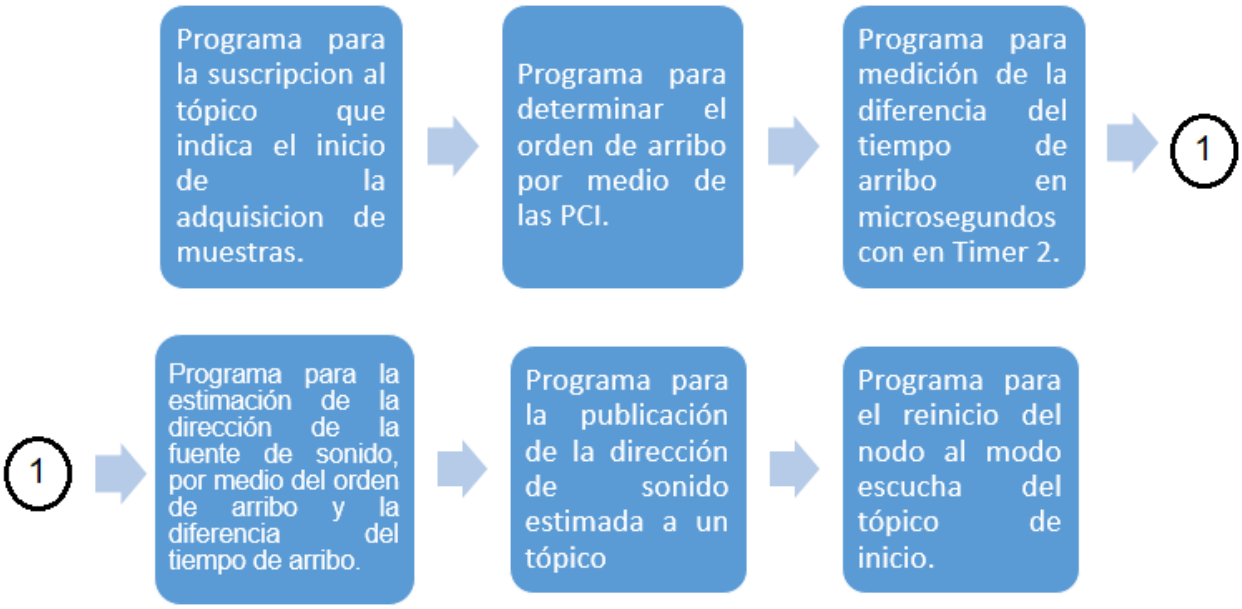

Figura 43: Subprogramas del Nodo de Arduino.

La finalidad para la creación de este nodo es poder integrarlo a la prueba Hear and Rotate, la cual consiste en indicar a una persona que produzca un sonido y robot debe girar hacia la persona que ha producido el sonido, por lo que es necesario conocer los resultados de la integración de este nodo para la estimación de la dirección de la fuente de sonido en esta prueba y validar su correcto funcionamiento.

La prueba consiste en correr un nodo en el robot que coordina las acciones de escuchar y girar, el robot da una señal para que la persona genere un sonido, la persona debe generar un sonido después que ha terminado la señal en el caso esta prueba se empleó un aplauso (ver Figura 44), el nodo de Arduino se suscribe al tópico "/isdf/isdf_ss" en el que el robot ha publicado un mensaje de tipo string, como la señal de inicio para adquirir datos, una vez que el nodo de Arduino a estimado la dirección de la fuente de sonido este la publica en el tópico "/isdf/isdf_pos" en forma de un string, después el nodo coordinador se suscribe al tópico "/isdf/isdf_pos" para obtener la dirección de la fuente de sonido y girar hacia a ella, una vez finalizado el giro da una señal de que ha termino la prueba.

Figura 44: Prueba hear and rotate

En la ejecución de las pruebas que obtuvimos solo se presentaron dos tipos de resultados:

El giro era exacto en dirección de la fuente de sonido (ver Figura 45), por lo que le error en la estimación de la dirección de la fuente de sonido era muy pequeño, indican

dando que el sistema había podido determinar la diferencia del tiempo de arribo, generando de esta forma un ángulo intermedio en el cuadrante en el que se ubicó la fuente de sonido, proporcionando una estimación de alta resolución.

Figura 45: Giro exacto a la ubicación de la fuente de sonido.

El giro era inexacto en dirección a la fuente de sonido (ver Figura 46), por lo que el error en la estimación de la dirección de la fuente de sonido podría ser de hasta $\pi/2$ rad, indicando que el sistema no habría podido determinar correctamente la diferencia del tiempo de arribo, posicionando al robot únicamente en el cuadrante en el que se ubicó la fuente de sonido.

Figura 46: Giro inexacto a la ubicación de la fuente de sonido.

Para visualizar la prueba completa ir al enlace:

- https://www.youtube.com/watch?v=EvL9_qHDkKY

Conclusiones.

Bajo los resultados obtenidos en la prueba Hear and Rotate, se puede decir que el sistema implementado es capaz de estimar la dirección de la fuente sonora con una mayor resolución que el sistema anterior. Con la utilización del método de la diferencia del tiempo de arribo para estimación la del origen del sonido, en esta pruebas se detectó que el sistema es afectado en su funcionamiento, produciendo un error entre la estimación de la dirección de fuente de sonido y la dirección real de la misma fuente, que puede llegar a ser de hasta $\pi/2$ rad . Este problema se presenta debido a que no se es capaz de determinar correctamente la diferencia del tiempo de arribo entre el primer micrófono en ser impactado y segundo micrófono en ser impactado, debido a que la onda de sonido que impacta en el segundo micrófono se reduce su intensidad por la pérdida de energía en su propagación hasta este punto, generando en el segundo micrófono una señal similar a la producida en el primer micrófono , desfasada en el tiempo pero reducida en su amplitud, ocasionando que esta no pudiera rebasar el nivel del umbral establecido en los sensores y no lograr detectar correctamente la incidencia del sonido en el micrófono, sino hasta tiempo después donde la amplitud de la señal logra rebasar el nivel de umbral. Por lo que se concluye que el sistema es muy dependiente de intensidad de la onda de sonido que muestrea, para que realice una correcta estimación de la dirección de la fuente de sonido.

Como una posible solución a este problema, se propone la implementación de un ajuste automático de ganancia en cada uno de los micrófonos, para que la perdida en la amplitud de la señal generada no afecte la medición en momento del impacto, pues al existir un cambio en la tensión de referencia este pueda se amplificado y detectado, por falta de tiempo el estudio e implementación de esta posible solución se dejara como una propuesta para trabajos futuros.

Competencias desarrolladas y aplicadas.

En la realización del proyecto se desarrollaron y aplicaron tanto competencias personales, como competencias técnicas que se describen a continuación. La habilidad en el trabajo en equipo fue una competencia indispensable empleo a lo largo del proyecto ya que para poder llevar a cabo un proyecto y culminarlo se requiere la cooperación de diversas personas, por lo que es muy importante la coordinación y colaboración en equipo a pesar del que asignación del proyecto es individual. Dentro de las habilidades técnicas aplicadas se encuentran materias como instrumentación y electrónica las cuales ayudaron al análisis del funcionamiento de los sensores de sonido para poder emplearlos de manera correcta en el proyecto. Materias como microcontroladores, programación y robótica, fueron el eje central en el desarrollo del programa para la estimación de la dirección de la fuente sonora. Asignaturas como formulación y evaluación de proyectos fueron fundamental para la planeación, estructuración y ejecución del proyecto. Por lo que el desarrollo y aplicación de estas competencias es integral debido a que todas son importantes en la generación de un proyecto de esta índole.

Fuentes de información.

Referencias:

[1], [2] "INAOE - Ciencias Computacionales", *Ccc.inaoep.mx*, 2017. [En línea]. Disponible en: https://ccc.inaoep.mx/index.php. [Accedido: 15- Abril- 2019].

[3] K. Enoksson, "Robot Seguidor de Sonido", Diva-portal.org, 2017. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:1200481/FULLTEXT01.pdf. [Accessed: 05- Feb- 2019].

[4] A. Argawal, "Detection of Audio Source Direction Using Autonomous Robot", *Pdfs.semanticscholar.org*, 2014. [En línea]. Disponible en: https://pdfs.semanticscholar.org/6c6d/4ab95601455822d8ff1e6f8e72f489fc52a0.pdf. [Accedido: 05- Febrero- 2019].

[5] D. Gusland, "Arduino sound localization", *Piemp4.com*, 2015. [En línea]. Disponible en: http://piemp4.com/video/file/arduino-sound-localization?id=vGGvgp6u1-M. [Accedido: 05- Febrero- 2019].

Bibliografías:

"Interaural Intensity Difference (IID)", sweetwater.com, 2004. [En línea]. Disponible en: https://www.sweetwater.com/insync/interaural-intensity-difference-iid-2/. [Accedido: 20- Marzo- 2019].

I. Meza, "Robotics and Autonomous Systems", academia.edu, 2017. [En línea]. Disponible en: https://www.academia.edu/34244896/Localization_of_sound_sources_in_robotics_A_review. [Accedido: 20- Marzo- 2019].

ANEXO A: CÓDIGO FUENTE DEL NODO IMPLEMENTADO EN ARDUINO UNO.

SensorSoundMarkovito.ino

```
#include "sensor_sound.h"
SoundSensorMarkovito SoundSensorM;
void setup() {
  // put your setup code here, to run once:
  SoundSensorM.Init();
}
void loop() {
 // put your main code here, to run repeatedly:
 SoundSensorM.PublishAngleRadiansQuadrants();
 SoundSensorM.RestartSensor();
}
```

sensor_sound.h

```
/************************************************************
   Autor: Jose Angel Balbuena Palma.
   Puporse: This code was created for the detection
            of the sound source direction, with four
            microphones and arduino one to the service
            robot Markovito.
   Notes: The pins used for the inputs are the
                  at the pin 5 front sensor,
                  at the pin 4 back sensor,
                  at the pin 3 left sensor,
                  at the pin 2 right sensor.
           The logic state of the sensor is low,
           when this sensor is hit for a sound wave and
           is high when sensor isn't hit.
  Date: 30/05/2019.
************************************************************/
#ifndef SENSOR_SOUND_H
#define SENSOR_SOUND_H
/********************** Libraries ***********************/
#include <stdlib.h>//standard library
#include <math.h>// math library
/******************* AVR Libraries **********************/
#include <avr/io.h>/** Library of avr I/O pins**/
#include <avr/interrupt.h>/*Library of avr interrupts*/
/******************* ROS Libraries **********************/
#include<ros.h>/** Ros library **/
#include<std_msgs/String.h>/** Library String messages**/
/********************************************************/
#define distance_5to2 0.1805/*** Distance between the sensor 5 and sensor 2 ****/
```

```
#define distance_5to3 0.1805/*** Distance between the sensor 5 and sensor 3 ****/
#define distance_4to2 0.155/*** Distance between the sensor 4 and sensor 2 ****/
#define distance_4to3 0.159/*** Distance between the sensor 4 and sensor 3 ****/
#define samples_number 3 /*Number of samples of time to take*/
class SoundSensorMarkovito
{
  public:
    SoundSensorMarkovito();/***Contructor Class SensorSoundMarkovito ****/
    void Init();/*Method to initialize the sound sensor*/
    void RestartSensor();/*Method to restart the sensor data*/
    void PublishPosition();/*Method to publish the source sound position*/
    void PublishArrive();/*Method to publish the  arrive sound to robot*/
    void PublishAngleDegrees();/*Method to publish the source sound degrees angles
position*/
    void PublishAngleRadians();/*Method to publish the source sound  radians
angles position*/
    void PublishAngleRadiansQuadrants();/*Method to publish the sound source
radian angle position in the quadrants*/
  private:
    int angle_markovito_degrees;
    float angle_markovito_radians;
    float angle_markovito_radians_coord;
};
#endif
```

sensor_sound.cpp

```
/***Include the sensor_sound.h
file*****************************************************/
#include "sensor_sound.h"
/********************** Struct Sound
Sensor*********************************************/
typedef enum {zero, first, second, third, fourth} state;
struct sensor_sound {
  /*Save the position the sound arrive the pines *************/
  volatile uint8_t arrive[2]={0,0};
  /** Save the time delay to arrive between first and second**/
  volatile unsigned int delay_time = 0;
  /** Save the time delay to arrive sample  first and second**/
  volatile unsigned int delay_time_sample[samples_number];
  /*** Counter interrupt 16 us   ***************************/
  volatile unsigned int timer_counter = 0;
  /***** Count sound arrive *********************************/
  volatile unsigned int counter_sample=0;
  /***Enable or disable the sensor sound from Markovito*******/
```

```
  int start = 0;
  /************Control the arrive state *********************/
  volatile state State;
  /******Variable to save of the port hirtory ***************/
  volatile uint8_t portbhistory = 0xFF;
  /****Variable to save of the completion of pins samplings***/
  volatile uint8_t finish_sample_pins=0;
} sensor_sound;
/****************** Instances ROS
*****************************************************/
/****************** ROS Suscriber
*****************************************************/
ros::NodeHandle nh;
/************ CallBack Funtion to Suscriber
*****************************************/
void callback_start(const std_msgs::String& start_message){
     sensor_sound.start = 1;
}
ros::Subscriber<std_msgs::String> start_sensor("/isdf/isdf_ss",callback_start);
/******************  ROS Publisher
*****************************************************/
std_msgs::String sound_direction_message;
ros::Publisher sound_direction("/isdf/isdf_pos",&sound_direction_message);
/******************* Funtion
Prototype*************************************************/
/***************** Funtion to initialize pines
****************************************/
void init_pins();
/******Funtion configuration of the Pin Change Interrupt(PCI)
*************************/
void init_PCI_PortD();
/******Funtion configuration of the Timer2
Interrupt*********************************/
void init_TIMER2();
/******This funtion return the relative angle on grades using delay time to arrive
*****/
int angle_degrees(unsigned int time_delay_funtion, float distance);
/******This funtion return the relative angle on radians using delay time to
arrive ****/
float angle_radians(unsigned int time_delay_funtion, float distance);
/**********This funtion determine the delay time to arrive
****************************/
void  determine_time_delay();
```

```
/******This funtion return the absolut angle on radians
*******************************/
float determine_angle_radians();
/******This funtion return the absolut angle on radians coord
************************/
float determine_angle_radians_coord();
/******This funtion return the absolut angle on degrees
*******************************/
int determine_angle_degrees();

/***************************** Class SensorSoundMarkovito
***************************/
/***************** Contructor Class SensorSoundMarkovito
*******************************/
SoundSensorMarkovito::SoundSensorMarkovito() {
  angle_markovito_degrees= 0;
  angle_markovito_radians= 0;
  angle_markovito_radians_coord=0;
}
/************* Method to  initialize Sensor
Sound*************************************/
void SoundSensorMarkovito::Init() {
  init_pins();
  cli();//Stop interruptions
  init_TIMER2();//Initialize Timer 2
  init_PCI_PortD();//Initialize Pin Change Interrupt PortD
  sei();//Enable interruptions
  /****************** Initialize Node
*************************************************/
  nh.initNode();
  /****************** Suscriber to "/isdf/isdf_ss" topic
*******************************/
  nh.subscribe(start_sensor);
  /****************** Advertise "/isdf/isdf_pos" topic
*********************************/
  nh.advertise(sound_direction);
  /****************** Initialize State to zero
*************************************/
  sensor_sound.State = state::zero;
  /****************** Initialize sample time to zero
*********************************/
  for(int i=0;i<samples_number;i++){
      sensor_sound.delay_time_sample[i]=0;
    }
```

```
}
/*************** Method to publish the source sound position
*************************/
void SoundSensorMarkovito::PublishPosition() {
  if ((sensor_sound.State == state::fourth) && (sensor_sound.start)){
    angle_markovito_degrees = determine_angle_degrees();
    if (angle_markovito_degrees == 0 || angle_markovito_degrees == 360) {
      sound_direction_message.data  = "Front";
    }
    else if ((angle_markovito_degrees > 0) && (angle_markovito_degrees< 90)) {
      sound_direction_message.data  = "Front Left";
    }
    else if (angle_markovito_degrees == 90) {
      sound_direction_message.data  = "Left";
    }
    else if ((angle_markovito_degrees > 90) && (angle_markovito_degrees < 180)) {
      sound_direction_message.data  = "Back left";
    }
    else if (angle_markovito_degrees == 180) {
      sound_direction_message.data  = "Back";
    }
    else if ((angle_markovito_degrees > 180) && (angle_markovito_degrees < 270)) {
     sound_direction_message.data  = "Back Right";
    }
    else if (angle_markovito_degrees== 270) {
      sound_direction_message.data  = "Right";
    }
    else if ((angle_markovito_degrees > 270) && (angle_markovito_degrees < 360)) {
      sound_direction_message.data  = "Front Right";
    }
    sound_direction.publish(&sound_direction_message);
  }
}
/*************** Method to publish the  arrive sound to robot
**********************/
void SoundSensorMarkovito::PublishArrive() {
  /****Publish only if the state is fourth and start is enable *******/
  if ((sensor_sound.State == state::fourth) && (sensor_sound.start)){
    angle_markovito_degrees = determine_angle_degrees();
    char dataBuffer[20]; /** Char buffer to send data *****************/
    /** Publish the first to arrive ***********************************/
    sprintf(dataBuffer, "FIRST: %d", sensor_sound.arrive[0]);
    sound_direction_message.data = dataBuffer;
    sound_direction.publish(&sound_direction_message);
```

```
    /** Publish the second to arrive ********************************/
    sprintf(dataBuffer, "SECOND: %d", sensor_sound.arrive[1]);
    sound_direction_message.data = dataBuffer;
    sound_direction.publish(&sound_direction_message);
    /** Publish the delay time between the first and secont to arrive **/
    sprintf(dataBuffer, "TIME 1-2: %d", sensor_sound.delay_time);
    sound_direction_message.data = dataBuffer;
    sound_direction.publish(&sound_direction_message);
  }
}
/************ Method to publish the sources sound angle position
*********************/
void SoundSensorMarkovito::PublishAngleDegrees() {
  /****Publish only if the state is fourth and start is enable *******/
  if ((sensor_sound.State == state::fourth) && (sensor_sound.start)) {
    /*Publish the sources sound angle position ************************/
    angle_markovito_degrees = determine_angle_degrees();
    char dataBuffer[10];
    sprintf(dataBuffer,"%d",angle_markovito_degrees);
    sound_direction_message.data = dataBuffer;
    sound_direction.publish(&sound_direction_message);
  }
}
/************ Method to publish the sources sound angle position
*********************/
void SoundSensorMarkovito::PublishAngleRadians() {
  /****Publish only if the state is fourth and start is enable *******/
  if ((sensor_sound.State == state::fourth) && (sensor_sound.start)) {
    /*Publish the sources sound angle position ************************/
    angle_markovito_radians = determine_angle_radians();
    char float_str[10];
    dtostrf(angle_markovito_radians,4,2,float_str);
    char dataBuffer[10];
    sprintf(dataBuffer,"%s",float_str);
    sound_direction_message.data = dataBuffer;
    sound_direction.publish(&sound_direction_message);
  }
}
/****Method to publish the source sound  radians angles position
Reverse*************/
void SoundSensorMarkovito::PublishAngleRadiansQuadrants(){
/****Publish only if the state is fourth and start is enable *******/
if ((sensor_sound.State == state::fourth) && (sensor_sound.start)){
    angle_markovito_radians_coord = determine_angle_radians_coord();
```

```
    char float_str[10];
    dtostrf(angle_markovito_radians_coord ,4,2,float_str);
    char dataBuffer[10];
    sprintf(dataBuffer,"%s",float_str);
    sound_direction_message.data = dataBuffer;
    sound_direction.publish(&sound_direction_message);
  }
}
/*************** Method to restart the sensor data
*********************************/
void SoundSensorMarkovito::RestartSensor() {
  /****Publish only if the state is fourth and start is enable *******/
  if ((sensor_sound.State == state::fourth) && (sensor_sound.start)){
    /*********Reset variables*********************************************/
    sensor_sound.State = state::zero;
    sensor_sound.arrive[0] = 0;
    sensor_sound.arrive[1] = 0;
    sensor_sound.delay_time = 0;
    sensor_sound.counter_sample = 0;
    sensor_sound.portbhistory = 0xFF;
    sensor_sound.start = 0;
    sensor_sound.finish_sample_pins = 0;
    for(int i=0;i<samples_number;i++){
      sensor_sound.delay_time_sample[i]=0;
    }
    /************* Enable pin change interrupts again *********************/
    cli();
    PCMSK2 &= 0b00000000;
    PCMSK2 |= 0b00111100;
    PCICR  |= (1<< PCIE2);
    sei();
    delay(1000);
  }
  while(sensor_sound.State != state::fourth){
    if((!sensor_sound.finish_sample_pins)&&((((sensor_sound.timer_counter << 8) +
TCNT2) / 16)>10000)){
      TCCR2B &= 0b00000000;
      PCICR &=~(1 << PCIE2);
      sensor_sound.timer_counter=0;
      TCNT2=0;
      sensor_sound.State = state::fourth;
      }
    else if((sensor_sound.finish_sample_pins)&&(PIND & (1 << PD2))&&(PIND & (1 <<
PD3))&&(PIND & (1 << PD4))&&(PIND & (1 << PD5))){
```

```
      PCICR &=~(1 << PCIE2);
      sensor_sound.State = state::fourth;
      }
    nh.spinOnce();
    }
  nh.spinOnce();
}
/*********************Interrupt PIN CHANGE PortD*************************/
ISR(PCINT2_vect) { // handle pin change interrupt for D0 to D7 here
  /**************If any pin D2,D3,D4,D5 change of state*******************/
  uint8_t changedbits;
  changedbits = PIND ^ sensor_sound.portbhistory;
  /*************** CHANGE PIN  D2 to LOW *********************************/
  if(sensor_sound.start){
  /***only if D2 is LOW and change state and first to arrive isn't D2 *****/
  if (!(PIND & (1 << PD2)) &&(changedbits&(1<< PIND2))&&(PCMSK2 & (1 << PCINT18)))
{
    TCCR2B &= 0b00000000;
    if (sensor_sound.State == state::zero) {//First to arrive is D2
      TCCR2B |= 0b00000001;
      PCMSK2 |= 0b00111100;
      PCMSK2 &= ~(1 << PCINT18);
      TCNT2 = 0;
      sensor_sound.timer_counter = 0;
      sensor_sound.State = state::first;
      if(!sensor_sound.counter_sample){
         sensor_sound.arrive[0]=2;
        }
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::first) { //Second to arrive is D2
      TCCR2B |= 0b00000001;
      sensor_sound.delay_time_sample[sensor_sound.counter_sample]=
((sensor_sound.timer_counter << 8) + TCNT2) / 16;
      PCMSK2 &= ~(1 << PCINT18);
      sensor_sound.State = state::second;
      if(!sensor_sound.counter_sample){
         sensor_sound.arrive[1]=2;
        }
      sensor_sound.counter_sample++;
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::second) { //Third to arrive is D2
      TCCR2B |= 0b00000001;
```

```
      PCMSK2 &= ~(1 << PCINT18);
      sensor_sound.State = state::third;
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::third) {//Fourth to arrive is D2
      PCMSK2 |= 0b00111100;
      PCMSK2 &= ~(1 << PCINT18);
      sensor_sound.State = state::zero;
      sensor_sound.finish_sample_pins=1;
    }
  }
  /**************CHANGE PIN  D3  to LOW ***************************/
  /*only if D3 is LOW and chanege state and first to arrive isn't D3*/
  if (!(PIND & (1 << PD3)) &&(changedbits&(1<< PIND3))&&(PCMSK2 & (1 << PCINT19)))
{
    TCCR2B &= 0b00000000;
    if (sensor_sound.State == state::zero) {//First to arrive is D3
      TCCR2B |= 0b00000001;
      PCMSK2 |= 0b00111100;
      PCMSK2 &= ~(1 << PCINT19);
      TCNT2 = 0;
      sensor_sound.timer_counter = 0;
      sensor_sound.State = state::first;
      if(!sensor_sound.counter_sample){
         sensor_sound.arrive[0]=3;
        }
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::first) { //Second to arrive is D3
      TCCR2B |= 0b00000001;
      sensor_sound.delay_time_sample[sensor_sound.counter_sample]=
((sensor_sound.timer_counter << 8) + TCNT2) / 16;
      PCMSK2 &= ~(1 << PCINT19);
      sensor_sound.State = state::second;
      if(!sensor_sound.counter_sample){
         sensor_sound.arrive[1]=3;
        }
      sensor_sound.counter_sample++;
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::second) { //Third to arrive is D3
      TCCR2B |= 0b00000001;
      PCMSK2 &= ~(1 << PCINT19);
      sensor_sound.State = state::third;
```

```
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::third) { //Fourth to arrive is D3
      PCMSK2 |= 0b00111100;
      PCMSK2 &= ~(1 << PCINT19);
      sensor_sound.State = state::zero;
      sensor_sound.finish_sample_pins=1;
    }
  }
  /**************CHANGE PIN  D4 to LOW*******************************/
  /**only if D4 is LOW and chanege state and first to arrive isn't D4**/
   if (!(PIND & (1 << PD4)) &&(changedbits&(1<< PIND4))&&(PCMSK2 & (1 <<
PCINT20))){
    TCCR2B &= 0b00000000;
    if (sensor_sound.State == state::zero) {//First to arrive is D4
      TCCR2B |= 0b00000001;
      PCMSK2 |= 0b00111100;
      PCMSK2 &= ~(1 << PCINT20);
      TCNT2 = 0;
      sensor_sound.timer_counter = 0;
      sensor_sound.State = state::first;
      if(!sensor_sound.counter_sample){
         sensor_sound.arrive[0]=4;
        }
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::first) { //Second to arrive is D4
      TCCR2B |= 0b00000001;
      sensor_sound.delay_time_sample[sensor_sound.counter_sample]=
((sensor_sound.timer_counter << 8) + TCNT2) / 16;
      PCMSK2 &= ~(1 << PCINT20);
      sensor_sound.State = state::second;
      if(!sensor_sound.counter_sample){
         sensor_sound.arrive[1]=4;
        }
      sensor_sound.counter_sample++;
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::second) {//Third to arrive is D4
      TCCR2B |= 0b00000001;
      PCMSK2 &= ~(1 << PCINT20);
      sensor_sound.State = state::third;
      sensor_sound.finish_sample_pins=0;
    }
```

```
    else if (sensor_sound.State == state::third) {//Fourth to arrive is D2
      PCMSK2 |= 0b00111100;
      PCMSK2 &= ~(1 << PCINT20);
      sensor_sound.State = state::zero;
      sensor_sound.finish_sample_pins=1;
    }
  }
  /************** CHANGE PIN D5 to LOW *******************************/
  /*only if D5 is LOW abnd chanege state and  first to arrive isn't 5**/
  if (!(PIND & (1 << PD5)) &&(changedbits&(1<< PIND5))&&(PCMSK2 & (1 << PCINT21)))
{
    TCCR2B &= 0b00000000;
    if (sensor_sound.State == state::zero) {//First to arrive is D5
      TCCR2B |= 0b00000001;
      PCMSK2 |= 0b00111100;
      PCMSK2 &= ~(1 << PCINT21);
      TCNT2 = 0;
      sensor_sound.timer_counter = 0;
      sensor_sound.State = state::first;
      if(!sensor_sound.counter_sample){
         sensor_sound.arrive[0]=5;
        }
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::first) { //Second to arrive is D5
      TCCR2B |= 0b00000001;
      sensor_sound.delay_time_sample[sensor_sound.counter_sample]=
((sensor_sound.timer_counter << 8) + TCNT2) / 16;
      PCMSK2 &= ~(1 << PCINT21);
      sensor_sound.State = state::second;
      if(!sensor_sound.counter_sample){
         sensor_sound.arrive[1]=5;
        }
      sensor_sound.counter_sample++;
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::second) { //Third to arrive is D2
      TCCR2B |= 0b00000001;
      PCMSK2 &= ~(1 << PCINT21);
      sensor_sound.State = state::third;
      sensor_sound.finish_sample_pins=0;
    }
    else if (sensor_sound.State == state::third) { //Fourth to arrive is D2
      PCMSK2 |= 0b00111100;
```

```
      PCMSK2 &= ~(1 << PCINT21);
      sensor_sound.State = state::zero;
      sensor_sound.finish_sample_pins=1;
    }
   }
  }
  sensor_sound.portbhistory = PIND;
}
/*** Timer 2 interrupt funtion******/
ISR(TIMER2_OVF_vect) {
  /**Increase counter every 16 us **/
  sensor_sound.timer_counter++;
}

/***************** Funtion to initialize pines ******************/
void init_pins() {
  /**Set the pins 2,3,4,5 like inputs***/
  DDRD &= ~((1 << 2) | (1 << 3) | (1 << 4) | (1 << 5));
  /**Enable the pullup for the pins 2,3,4,5***/
  PORTD |= ((1 << 2) | (1 << 3) | (1 << 4) | (1 << 5));
}
/******Funtion configuration of the Pin Change Interrupt(PCI) **********/
void init_PCI_PortD() {
  /***** Pin Change Interrupt Control Register for the port D***********/
  /****** Activating Pin Change Interrupt portD (PINES D2,D3,D4,D5)******/
  PCMSK2 |= 0b00111100;
  /************* Enable Activating Pin Change Interrupt ****************/
  PCICR |= (1 << PCIE2);
}
/******Funtion configuration of the Timer2 Interrupt*******************/
void init_TIMER2() {
  TCCR2A = 0;
  TCCR2B &= 0b00000000;
  TCNT2  = 0;
  TIMSK2 |= (1 << TOIE2);
}
/******This funtion return the relative angle on grades using delay time to arrive
****/
int angle_degrees(unsigned int time_delay_funtion, float distance) {
  return (int)((asin((((float)time_delay_funtion) * 0.000001 * 343.0000) /
distance)) * 180 / PI);
}
/******This funtion return the relative angle on radians using delay time to
arrive ****/
```

```
float angle_radians(unsigned int time_delay_funtion, float distance) {
  return (float)(asin((((float)time_delay_funtion) * 0.000001 * 343.0000)/
distance));
}
/**********This funtion determine the delay time to arrive
****************************/
void  determine_time_delay() {
  unsigned int time_delay_maximum=0;
  unsigned long times_sum=0;
  unsigned int times_samples_counter=0;
  if ((sensor_sound.arrive[0] == 5 && sensor_sound.arrive[1] == 2) ||
(sensor_sound.arrive[0] == 2 && sensor_sound.arrive[1] == 5)) {
    time_delay_maximum = (int)(distance_5to2 / (0.000001 * 343));
  }
  else if ((sensor_sound.arrive[0] == 5 && sensor_sound.arrive[1] == 3) ||
(sensor_sound.arrive[0] == 3 && sensor_sound.arrive[1] == 5)) {
    time_delay_maximum = (int)(distance_5to3 / (0.000001 * 343));
  }
  else if ((sensor_sound.arrive[0] == 4 && sensor_sound.arrive[1] == 2) ||
(sensor_sound.arrive[0] == 2 && sensor_sound.arrive[1] == 4)) {
    time_delay_maximum = (int)(distance_4to2 / (0.000001 * 343));
  }
  else if ((sensor_sound.arrive[0] == 4 && sensor_sound.arrive[1] == 3) ||
(sensor_sound.arrive[0] == 3 && sensor_sound.arrive[1] == 4)) {
    time_delay_maximum = (int)(distance_4to3 / (0.000001 * 343));
  }
  for(int i=0;i<samples_number;i++){

if((sensor_sound.delay_time_sample[i]>0)&&(sensor_sound.delay_time_sample[i]<=time
_delay_maximum)){
        times_sum+=sensor_sound.delay_time_sample[i];
        times_samples_counter++;
      }
    }
  if(!times_sum)times_samples_counter=1;
  sensor_sound.delay_time=round(times_sum/times_samples_counter);
}
/******This funtion return the absolut angle on radians
*****************************/
float determine_angle_radians(){
  float angle_radian=0;
  determine_time_delay();
  /*THE FIRST TO ARRIVE IS FRONT PIN*/
  if (sensor_sound.arrive[0] == 5) {
```

```
    /*THE SECOND TO ARRIVE IS RIGHT PIN*/
    if (sensor_sound.arrive[1] == 2) {
      angle_radian = (2*PI) - angle_radians(sensor_sound.delay_time,
distance_5to2);
    }
    /*THE SECOND TO ARRIVE IS LEFT PIN*/
    else if (sensor_sound.arrive[1] == 3) {
      angle_radian  = 0 + angle_radians(sensor_sound.delay_time, distance_5to3);
    }
    else {
      angle_radian  = 0;
    }
  }
  /*THE FIRST TO ARRIVE IS BACK PIN*/
  else if (sensor_sound.arrive[0] == 4) {
    /*THE SECOND TO ARRIVE IS RIGHT PIN*/
    if (sensor_sound.arrive[1] == 2) {
       angle_radian = PI + angle_radians(sensor_sound.delay_time, distance_4to2);
    }
    /*THE SECOND TO ARRIVE IS LEFT PIN*/
    else if (sensor_sound.arrive[1] == 3) {
      angle_radian = PI - angle_radians(sensor_sound.delay_time, distance_4to3);
    }
    else {
      angle_radian = PI;
    }
  }
  /*THE FIRST TO ARRIVE IS LEFT PIN*/
  else if (sensor_sound.arrive[0] == 3) {
    /*THE SECOND TO ARRIVE IS FRONT PIN*/
    if (sensor_sound.arrive[1] == 5) {
      angle_radian = (PI/2)-angle_radians(sensor_sound.delay_time, distance_5to3);
    }
    /*THE SECOND TO ARRIVE IS BACK PIN*/
    else if (sensor_sound.arrive[1] == 4) {
      angle_radian = (PI/2)+angle_radians(sensor_sound.delay_time, distance_4to3);
    }
    else {
      angle_radian = (PI/2);
    }
  }
  /*THE FIRST TO ARRIVE IS RIGHT PIN*/
  else if (sensor_sound.arrive[0] == 2) {
    /*THE SECOND TO ARRIVE IS FRONT PIN*/
```

```
    if (sensor_sound.arrive[1] == 5) {
      angle_radian = (3*PI/2)+angle_radians(sensor_sound.delay_time,
distance_5to2);
    }
    /*THE SECOND TO ARRIVE IS BACK PIN*/
    else if (sensor_sound.arrive[1] == 4) {
      angle_radian = (3*PI/2)-angle_radians(sensor_sound.delay_time,
distance_4to2);
    }
    else {
      angle_radian = (3*PI/2);
    }
  }
  return angle_radian;
}
/******This funtion return the absolut angle on radians coord
*****************************/
float determine_angle_radians_coord() {
  float angle_radian = 0;
  determine_time_delay();
  /*THE FIRST TO ARRIVE IS FRONT PIN*/
  if (sensor_sound.arrive[0] == 5) {
    /*THE SECOND TO ARRIVE IS RIGHT PIN*/
    if (sensor_sound.arrive[1] == 2) {
      angle_radian = 0 - angle_radians(sensor_sound.delay_time, distance_5to2);
    }
    /*THE SECOND TO ARRIVE IS LEFT PIN*/
    else if (sensor_sound.arrive[1] == 3) {
      angle_radian  = 0 + angle_radians(sensor_sound.delay_time, distance_5to3);
    }
    else {
      angle_radian  = 0;
    }
  }
  /*THE FIRST TO ARRIVE IS BACK PIN*/
  else if (sensor_sound.arrive[0] == 4) {
    /*THE SECOND TO ARRIVE IS LEFT PIN*/
    if (sensor_sound.arrive[1] == 2) {
       angle_radian = (0-PI) + angle_radians(sensor_sound.delay_time,
distance_4to2);
    }
    /*THE SECOND TO ARRIVE IS LEFT PIN*/
    else if (sensor_sound.arrive[1] == 3) {
      angle_radian = PI- angle_radians(sensor_sound.delay_time, distance_4to3);
```

```
    }
    else {
      angle_radian = PI;
    }
  }
  /*THE FIRST TO ARRIVE IS LEFT PIN*/
  else if (sensor_sound.arrive[0] == 3) {
    /*THE SECOND TO ARRIVE IS FRONT PIN*/
    if (sensor_sound.arrive[1] == 5) {
      angle_radian = (PI/2)-angle_radians(sensor_sound.delay_time, distance_5to3);
    }
    /*THE SECOND TO ARRIVE IS BACK PIN*/
    else if (sensor_sound.arrive[1] == 4) {
      angle_radian = (PI/2)+angle_radians(sensor_sound.delay_time, distance_4to3);
    }
    else {
      angle_radian = (PI/2);
    }
  }
  /*THE FIRST TO ARRIVE IS RIGHT PIN*/
  else if (sensor_sound.arrive[0] == 2) {
    /*THE SECOND TO ARRIVE IS FRONT PIN*/
    if (sensor_sound.arrive[1] == 5) {
      angle_radian = (0-(PI/2))+angle_radians(sensor_sound.delay_time,
distance_5to2);
    }
    /*THE SECOND TO ARRIVE IS BACK PIN*/
    else if (sensor_sound.arrive[1] == 4) {
      angle_radian = (0-(PI/2))-angle_radians(sensor_sound.delay_time,
distance_4to2);
    }
    else {
      angle_radian = (0-(PI/2));
    }
  }
  return angle_radian;
}
/******This funtion return the absolut angle on degrees
*****************************/
int determine_angle_degrees() {
  int angle_degree = 0;
  determine_time_delay();
  /*THE FIRST TO ARRIVE IS FRONT PIN*/
  if (sensor_sound.arrive[0] == 5) {
```

```
  /*THE SECOND TO ARRIVE IS RIGHT PIN*/
  if (sensor_sound.arrive[1] == 2) {
    angle_degree = 360 - angle_degrees(sensor_sound.delay_time, distance_5to2);
  }
  /*THE SECOND TO ARRIVE IS LEFT PIN*/
  else if (sensor_sound.arrive[1] == 3) {
    angle_degree  = 0 + angle_degrees(sensor_sound.delay_time, distance_5to3);
  }
  else {
    angle_degree  = 0;
  }
}
/*THE FIRST TO ARRIVE IS BACK PIN*/
else if (sensor_sound.arrive[0] == 4) {
  /*THE SECOND TO ARRIVE IS RIGHT PIN*/
  if (sensor_sound.arrive[1] == 2) {
    angle_degree = 180 + angle_degrees(sensor_sound.delay_time, distance_4to2);
  }
  /*THE SECOND TO ARRIVE IS LEFT PIN*/
  else if (sensor_sound.arrive[1] == 3) {
    angle_degree = 180 - angle_degrees(sensor_sound.delay_time, distance_4to3);
  }
  else {
    angle_degree = 180;
  }
}
/*THE FIRST TO ARRIVE IS LEFT PIN*/
else if (sensor_sound.arrive[0] == 3) {
  /*THE SECOND TO ARRIVE IS FRONT PIN*/
  if (sensor_sound.arrive[1] == 5) {
    angle_degree = 90-angle_degrees(sensor_sound.delay_time, distance_5to3);
  }
  /*THE SECOND TO ARRIVE IS BACK PIN*/
  else if (sensor_sound.arrive[1] == 4) {
    angle_degree = 90+angle_degrees(sensor_sound.delay_time, distance_4to3);
  }
  else {
    angle_degree = 90;
  }
}
/*THE FIRST TO ARRIVE IS RIGHT PIN*/
else if (sensor_sound.arrive[0] == 2) {
  /*THE SECOND TO ARRIVE IS FRONT PIN*/
  if (sensor_sound.arrive[1] == 5) {
```

```
      angle_degree = 270+angle_degrees(sensor_sound.delay_time, distance_5to2);
    }
    /*THE SECOND TO ARRIVE IS BACK PIN*/
    else if (sensor_sound.arrive[1] == 4) {
      angle_degree = 270-angle_degrees(sensor_sound.delay_time, distance_4to2);
    }
    else {
      angle_degree = 270;
    }
  }
  return angle_degree;
}
```