UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño, construcción y programación de un robot móvil autónomo

TESIS

Que para obtener el título de

Ingeniero en computación

P R E S E N T A

Ricardo Santiago López

DIRECTOR DE TESIS

Dr. Jesús Savage Carmona

Ciudad Universitaria, Cd. Mx., 2022

Dedicatorias

Este trabajo lo dedico especialmente a mis padres Abel y Ernestina, quienes siempre han estado para mí, acompañándome en cada momento importante de mi vida y teniendo fé en que lograría llegar muy lejos.

Este gran logro es para ustedes. Muchas gracias.

Agradecimientos

A mis padres Abel y Ernestina

Por ser el pilar de mi vida, enseñándome constantemente el valor del trabajo duro, la responsabilidad y el noble sacrificio. Por acompañarme todos los días cuidando mis pasos y motivándome para convertirme en un hombre de provecho.
Muchas gracias.

A mi hermana Elsa

Por ser mi amiga mas intima que siempre me escucha y me aconseja, quien se atreve a hacerme ver mis errores y me ayuda a corregir. Por motivarme a mejorar mi persona buscando ser un buen ejemplo para ella, tanto como ella lo ha sido para mi.
Gracias.

A mi tío Antonio y mi Abuelo Romualdo

Por creer en mí, por animarme y por mostrar lo verdaderamente orgullosos que estaban de mis logros.
Hasta el cielo, Gracias.

A mis amigos Omar, Misael y Sandra

Por acompañarme por tantos y tantos años compartiendo alegrías y tristezas y por hacerme sentir que existen personas en las que verdaderamente puedo depositar toda mi confianza.
Gracias.

Especiales

Dr. Jésus Savage Carmona

Agradezco su apoyo durante el desarrollo de este proyecto. Gracias por su enseñanza de los temas que fueron especialmente cruciales para el desarrollo de este trabajo y por su enorme paciencia y confianza que me tuvo durante todo el periodo que tardé en completarlo. Sin duda usted ha sido uno de los mejores profesores de los que tuve el gusto de conocer y tomar clase durante mi estancia en esta facultad.
Gracias.

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1. Introducción

1.1 Planteamiento del problema

Desde hace varios años el desarrollo de la robótica ha sido con el fin de proporcionar al ser humano herramientas más sofisticadas que le permitan resolver tareas tediosas, complejas o incluso peligrosas de una manera más eficiente y además automática.
Si bien la robótica desde sus inicios, tal y como la conocemos, ha tenido una mayor participación y desarrollo en la realización de tareas de carácter industrial para optimizar la producción o atender actividades sumamente peligrosas y complejas para el ser humano; en la actualidad también ha tenido un gran impacto en muchas más áreas donde se ha podido utilizar para automatizar actividades cotidianas que normalmente hacen los humanos, liberándolos así de tareas que pueden llegar a ser muy repetitivas y tediosas, o también asistiendo los para optimizar el desempeño de una actividad común, o sirviendo como herramientas en el desarrollo de actividades académicas, científicas y tecnológicas. Este tipo de robótica es conocida como robótica de servicio.
Dentro del área de la robótica de servicio nos podemos encontrar comúnmente con los robots móviles, los cuales realizan sus actividades navegando a través de los entornos en los que operan. Sin embargo, estos entornos normalmente se encuentran llenos de obstáculos (mayormente aleatorios) que bloquean su paso e impiden que siga una trayectoria fija, o también existen irregularidades en el terreno que recorren y que pueden afectar su navegación.
En el presente trabajo se tratará acerca de esta problemática: la navegación autónoma del robot móvil a través de un entorno experimental, para lograr un objetivo muy simple: Alcanzar una meta determinada.
Para ello, en el desarrollo de este trabajo se irán estudiando y aplicando los conceptos teóricos y prácticos que permitan, entre otras cosas; diseñar la estructura del robot móvil, implementar el sistema de control que permita ajustar su estado interno y desempeñe correctamente su navegación, y el desarrollo de los programas necesarios para que el robot ejecute todas sus tareas de forma autónoma, buscando cumplir con los objetivos planteados a continuación.

1.2 Objetivos del proyecto

Existe un objetivo de forma general para el proyecto, el cual es:

Lograr que el robot realice una correcta navegación a través de un entorno experimental de forma autónoma, para llegar desde un punto de partida cualquiera a una meta que estará señalizada por una torre de luz.

Se ha decidido también, separar el objetivo general en objetivos particulares para poder llevar a cabo el desarrollo del proyecto de una forma más ordenada.
Los objetivos particulares que planteo son los siguientes:

I. El robot posee la capacidad de medir su entorno para reconocer la existencia de posibles obstáculos u objetos que le obliguen alterar su funcionamiento y ejecute diversas acciones para que esta forma pueda adaptarse a las condiciones actuales.
II. El robot posee la capacidad de medir su estado interno para ajustar sus valores de modo que esto le permite mejorar el desempeño de sus funciones de navegación en el medio.
III. El robot posee la capacidad de ejercer sus tareas de manera autónoma, es decir, que no requiera de la intervención humana para realizar todas sus actividades.

A lo largo de este trabajo se retoman los objetivos particulares para ir desarrollando los conceptos teóricos y prácticos que permitan cumplir cada uno de ellos. Una vez cumplidos esos objetivos, se buscará conjuntar los resultados obtenidos y lograr que el proyecto cumpla con el objetivo general planteado.

1.3 Estructura general de la tesis

El desarrollo de este trabajo estará estructurado de la siguiente manera.

SECCIÓN PRIMERA: Comprende el capítulo 2 y aquí se comenzará con un marco teórico en el cual se presentan los antecedentes históricos de la robótica y varios de los conceptos básicos que tienen que ver con el área.

SECCIÓN SEGUNDA: Comprende los capítulos 3, 4 y 5, donde se presentarán los componentes elegidos para ensamblar al robot que cumplirá con los objetivos, así como su diseño y montaje de dichos componentes. Durante este apartado también se mostrará la forma en la que se hará uso de cada uno de los componentes y sus configuraciones iniciales para que funcionen. Esta sección está dedicada a cumplir principalmente con el objetivo particular I.

SECCIÓN TERCERA: Comprende los capítulos 6, 7 y 8, donde se presentarán e implementarán los conceptos teóricos y prácticos que harán posible que el robot sea capaz de medir su estado interno y realizar ajustes automáticos para controlar su funcionamiento y conseguir un desplazamiento adecuado por su medio. Esta sección estará enfocada principalmente al desarrollo del objetivo II.

SECCIÓN CUARTA: Comprende el capítulo 9 y es aquí donde se desarrollarán los algoritmos y/o programas que describirán el comportamiento del robot al interactuar con su entorno, y a partir de dichas interacciones, tomará decisiones de forma autónoma que alterarán su funcionamiento, de modo que pueda adaptarse a las condiciones actuales de su medio. Esta sección estará enfocada a desarrollar el objetivo particular III.

SECCIÓN QUINTA: Comprende el capítulo 10. En esta última sección, se conjuntan los resultados obtenidos de las secciones anteriores y se busca cumplir con el objetivo general.

Capítulo 2. Antecedentes y Marco Teórico

2.1 Robótica y antecedentes históricos

La palabra “ROBOT” tiene su origen en la palabra eslava Robota, que puede definirse como esclavitud o trabajo forzado y fue utilizada por primera vez en el teatro nacional de Praga en 1921 en la obra Rossum’s universal Robot del escritor checo Karel Kapec [Barrientos, 1997], la cual cuenta la historia de unos androides creados a partir de la formula desarrollada por el científico Rossum y que estaban obligados a realizar tareas manuales para atender a los humanos, hasta que estos androides se revelan y exterminan a la humanidad, menos a Rossum, de quien esperan les enseñe a reproducirse.

En años posteriores siguieron apareciendo más escritores de ciencia ficción que se inspiraron del trabajo de Karel Kapec y siguieron haciendo uso del término “Robot” en sus obras, de entre las cuales aparece la famosa metrópolis escrita por Thea von Harbou. Sin embargo a quien finalmente se le atribuye no solo la mayor difusión, sino también la creación del término “*Robótica*” para referirse al estudio y desarrollo de los robots, fue al escritor norteamericano de ciencia ficción Issac Asimov [Barrientos, 1997], quien también creó las famosas 3 leyes de la robótica las cuales también han servido como base para el desarrollo de esta área.

El interés de los seres humanos por desarrollar máquinas capaces de imitar los movimientos y la inteligencia de los seres vivos (incluyendo a los humanos) se remonta desde mucho tiempo atrás, comenzando desde los autómatas creados en la antigua Grecia hasta los creados por las culturas árabes y europeas en los siglos posteriores.

La palabra autómata tiene su origen etimológico en Automatos, la cual define a una máquina con la capacidad de imitar el movimiento o acciones de un ser vivo animado. Estos autómatas estaban principalmente desarrollados con elementos hidráulicos y mecánicos pertenecientes a su época, tales como poleas, palancas, cuerdas o los complejos sistemas mecánicos creados por los gremios de relojería. El uso principal que se le daba a estos mecanismos era meramente lúdico, pues solo se construían con el fin de entretener o servir a la realeza o a las cortes reales.

Fue hasta mucho tiempo después, entre el siglo XVIII y XIX, cuando los autómatas tomaron rol muy importante para la industria, sirviendo en procesos automáticos para hacer más eficiente la producción en las fábricas. Uno de los ejemplos más directos que se pueden dar de este tipo de autómatas es el telar mecánico de Jacquard, el cual hacía uso de una cinta de papel perforado que funcionaba a modo de programa para alterar el funcionamiento de dicho telar [Barrientos, 1997].

Regresando a los tiempos más modernos nos encontramos con los que se consideran los predecesores más directos que tienen los robots como máquinas automáticas, los Telemanipuladores.

El primer tele-manipulador fue desarrollado por R.C. Goertz en 1948 [Barrientos, 1997] y estaba pensado para la manipulación de sustancias radiactivas a partir de un sistema mecánico maestro-esclavo. El mecanismo maestro estaba ubicado en una zona segura y era desde aquí donde el operador lo manipulaba. El mecanismo esclavo era la estructura que manipula directamente los materiales radiactivos, y era este quien reproducía

los movimientos que eran ejecutados desde el mecanismo maestro. La siguiente fase de estos dispositivos empleaba el uso de servo-controles y la electrónica.

Se construyeron otras variantes como la Handy-Man de Ralph Mosher en 1958 [Barrientos, 1997], la cual consistía en dos brazos mecánicos controlados desde una estructura maestra llamada exoesqueleto. El principal empleamiento de estos dispositivos era en la industria nuclear. Posteriormente se introdujeron otras industrias como la Militar, la espacial y la submarina.

Con la aparición de los microprocesadores y posteriormente la sustitución del operador por un software que realizará el control, se concibió el concepto de robot, tal y como lo conocemos hasta nuestros días.

En la misma década de los 50 's en Estados Unidos, el Ingeniero George C. Devol, a quien se le considera ser el autor de las bases de la robótica industrial moderna, propuso la idea de “un dispositivo de transferencia de artículos programada que patentó en Estados Unidos en 1961” [Barrientos, 2007]. Esta misma idea la presentó con Joseph F. Engelbergen quien junto a él comienza el desarrollo y aplicación industrial de su maquinaria fundando la empresa Consolidated Controls Corporation y que después sería llamada Unimation.

Posteriormente Joseph F. Engelbergen lleva estos conceptos de robótica a Japón, donde se lleva a cabo un desarrollo acelerado de esta tecnología, gracias al interés que tenían empresas como Nissan, quien formaría también a la primera asociación de robótica en el mundo, conocida como JIRA (Asociación de Robótica industrial de Japón).

A partir de las invenciones y estudios desarrollados por los inventores de los tele-manipuladores y robots industriales, se generó un gran impacto en el mundo y la rama de la robótica comenzó a aumentar su desarrollo, creando y configurando máquinas que ya no solo persiguen un enfoque dedicado tareas industriales que tienen una naturaleza hostil y repetitiva para los seres humanos, y donde los protagonistas son los manipuladores, un tipo de robots que permanecen fijos a un solo sitio donde desempeñan sus tareas, sino que ahora el desarrollo se ha extendido a otros tipos más evolucionados como los robots móviles, los cuales tienen la capacidad de explorar el medio donde deben de realizar tareas más complejas de manera autónoma y que buscan con ello limitar cada vez más a la intervención humana.

Una definición formal que se le puede dar al robot en la actualidad según la Asociación de Industrias Robóticas (RIA, por sus siglas en inglés) es:

“manipulador multifuncional y reprogramable diseñado para desplazar materiales, componentes, herramientas o dispositivos especializados por medio de movimientos programados variables, con el fin de realizar diversas tareas.”

Sin embargo esta definición aplica más hacia un tipo más específico de robot, el cual es el industrial. Una definición que se podría inferir y que sea más acorde a los fines de este trabajo sería:

- Un robot es un dispositivo electromecánico reprogramable capaz de desempeñar diversas operaciones de forma autónoma o teleoperada y con la finalidad de automatizar procesos

2.2 Componentes de la robótica

Se puede visualizar a un robot como un sistema compuesto por un conjunto de diversos componentes, cada uno desempeñando una tarea específica, ya sea de recolección de datos necesarios para realizar operaciones y otros para ejecutar las operaciones como moverse o señalizar estados. Dichos componentes son los sensores y

los actuadores. Adicionalmente, a estos componentes se suman los elementos computacionales que permiten que el robot sea capaz de procesar la información recolectada para generar diversos estados que describen el comportamiento del mismo.
A continuación hago una breve descripción de cada uno de estos componentes.

Sensores

Los sensores son los dispositivos por los cuales el robot puede realizar mediciones del medio en el que desempeña sus actividades, de modo que a través de la lectura de diversos datos se pueda conocer el medio al que se enfrenta el robot y a partir de ello pueda tomar decisiones sobre cómo reaccionar.

Los principales inconvenientes que presentan los sensores en general se debe a la necesidad de aplicar sistemas electrónicos o programas informáticos adicionales para operar las señales que estos captan del medio, ya que normalmente vienen acompañadas de ruido, siendo necesarios los filtros y además, debido a la naturaleza analógica de la misma se llegan a requerir módulos que las discretizan para que puedan ser procesadas por un sistema digital.

Podemos encontrar las siguientes clasificaciones:

- Según el tipo de señal que manejen: **Analógicos o Digitales**
- Según la forma en que operen:
    - **Pasivos**: reciben directamente las señales del medio;
    - **Activos**: necesitan enviar una señal al medio y esperar su respuesta
- Según el medio en el que actúen y su finalidad:
    - **Externos**: censan el medio en el que el robot desempeña sus tareas.
    - **Internos**: censan al robot mismo para conocer su estado y advertir sobre ello

Actuadores

Los actuadores son dispositivos cuya finalidad es realizar la ejecución de una acción para que posteriormente el robot pueda modificar su estado y su entorno.

Tenemos 3 tipos de actuadores conocidos que implementan la robótica: los de energía Neumática, los de energía Hidráulica y los de energía eléctrica siendo estos últimos los más famosos.

- Actuadores Neumáticos: funcionan a partir de la energía proporcionada por la presión de gases (como el aire) y se dividen en dos tipos, cilindros neumáticos y motores neumáticos. Estos actuadores son muy poco populares en el área de la robótica debido a su baja precisión en el control del movimiento y normalmente se llegan a utilizar para sistemas sencillos del tipo “Todo o nada”, o sea, donde se requiere un movimiento total o completamente nulo, como las pinzas que utilizan algunos robots manipuladores.
- Actuadores Hidráulicos: Tienen un funcionamiento similar al de los actuadores neumáticos, solo que estos utilizan la presión de los líquidos (aceites minerales), para generar movimiento en el émbolo. También se dividen en los mismos tipos que los neumáticos.

Las principales ventajas en cuanto a los neumáticos es que debido a la baja presión que ejercen los líquidos respecto a los gases, resulta más sencillo ejercer un control lineal de la presión y el movimiento en los actuadores hidráulicos. Además resultan ser muy efectivos cuando se trata de ejercer enormes cantidades de fuerza para mover o soportar cargas estáticas de pesos gigantescos; por ello llegan a ser utilizados en robots que están destinados para mover, elevar o soportar cargas muy pesadas. Sin embargo, su implementación y mantenimiento llega a ser demasiados costosos respecto a los otros tipos de actuadores.

- Actuadores Eléctricos: Consisten en dispositivos que utilizan corriente eléctrica para operar y ofrecen una mayor sencillez en cuanto a su implementación y mantenimiento, además de una elevada precisión en el control de acciones y de ser más económicos. Razones por las cuales son los más utilizados en la construcción de los robots modernos. Entre los tipos más famosos de estos actuadores se tienen: Motores DC, Motores AC, Motores a Pasos y Servomotores.

Es pertinente señalar que también dentro de los actuadores eléctricos se podrían incluir diodos led, lámparas, displays u otros elementos electrónicos que le permitan al robot señalizar sobre sus estados.

Microcontroladores

Se tratan de pequeñas unidades de circuitos integrados que tienen la capacidad de almacenar y ejecutar programas con los cuales pueda procesar la información que recibe. Como si se tratase de una pequeña computadora, este circuito posee una unidad de procesamiento, unidades de memoria, y puertos de entrada y salida. Su principal objetivo es ofrecer la posibilidad de crear aplicaciones embebidas diseñadas para atender un cierto tipo de problema. En el caso de la robótica, podría enfocarse en ejecutar programas que procesen las señales que recibe de los sensores y generar las señales de salida para controlar los actuadores.

Computadoras

Son los dispositivos electrónicos más importantes dentro del área de la informática. Su principal objetivo es la resolución de diversos problemas a partir del desarrollo de programas diseñados para ello. Posee al igual que un microcontrolador, pero con mucho mayor potencia, procesadores de propósito general y procesamiento grafico, unidades de memoria para el almacenamiento de datos e instrucciones requeridas durante la ejecución de los programas, unidades de almacenamiento masivo para guardar información de manera persistente, puertos de entrada y salida para comunicar los datos desde y hacia el exterior, sistemas operativos para la administración de los recursos de cómputo, etc.

En el área de la robótica se utilizan comúnmente para la ejecución de programas que procesaran la información recibida del medio y la ejecución de maquinas de estados o algoritmos que determinen las diferentes acciones que deberá de realizar el robot, como la planeación de rutas y la navegación, la detección de obstáculos y su evasión, el reconocimiento de objetivos, etc; para que posteriormente comuniquen las señales con las que los actuadores lleven a cabo la ejecución de acciones. Prácticamente, son el cerebro que desempeña la inteligencia del robot.

2.3 Clasificaciones de la robótica

El impacto que la robótica ha generado en el mundo ha sido enorme, y por consecuencia su desarrollo se ha extendido demasiado logrando estar presente en diversos campos como los profesionales, educativos o incluso domésticos. De modo que en la actualidad la robótica ha adoptado un rasgo multidisciplinario gracias a la enorme cantidad de aplicaciones que se le ha dado. Esto también resulta un problema a la hora de generar definiciones precisas sobre el concepto de lo que es un robot y sobre cuáles son las diversas clasificaciones del mismo.

Basándose en aspectos generales que comparten los robots que son desarrollados frecuentemente se puede generar una clasificación en base a el tipo de actividades que desempeña, su morfología o arquitectura general, la configuración de sus movimientos, el medio en el que ejercen sus actividades y el nivel de inteligencia que poseen.

Según el tipo de actividades que desempeñan

- *Robot industrial*: Son robots destinados a realizar actividades de carácter meramente industrial. Típicamente se tratan de brazos mecánicos automatizados (o semi automáticos) compuestos por elementos unidos en serie cada uno con un cierto número de grados de libertad, pensados para la realización de tareas que tiene que ver con productividad y manipulación de herramientas, piezas para ensamblaje u otros objetos.
- *Robot de Servicio*: Son robots cuya finalidad es asistir a los humanos en la realización de diversas tareas que pueden ser complejas, peligrosas o incluso aburridas. En consecuencia, se pueden tener aun más clasificaciones dependiendo el tipo de servicio que ejecuten. Algunos ejemplos:
    - *Servicio doméstico* → Ejecución de tareas del hogar con el fin de liberar al humano de las tediosas actividades que representan.
    - *Servicio médico* → Asisten a los profesionales de la salud para realizar actividades que hagan eficiente el tratamiento en los pacientes. Además de servir como complementos artificiales (prótesis robóticas) en pacientes con discapacidades.
    - *Interactividad Social* → Robots destinados a desempeñar actividades que requieren interacción directa con los humanos. Suponen un reto en el desarrollo de habilidades comunicativas y sobre todo en las cuestiones estéticas.
    - *Exploración* → Robots desarrollados para realizar tareas de exploración en terrenos que pueden ser de difícil acceso para los humanos.

Según su morfología

- *Androides y Robots zoomorfos*: Robots que guardan similitud con los seres humanos y los animales imitando su estructura, movimientos y comportamientos.
- *Robots Móviles*: Robots con la capacidad de navegar a través de un medio haciendo uso de diversos mecanismos (como ruedas, orugas, piernas, etc) y configuraciones (direccionamiento diferencial, Ackerman, Triciclo, etc)

- *Poliarticulados*: Robots sedentarios y estructurados de modo que puedan mover y manipular objetos a través de sus efectores finales (Robots manipuladores).

Según la configuración de sus movimientos

- *Cartesianos*: realizan desplazamientos lineales en sus componentes. Las posiciones que adoptan se logran de acuerdo a las coordenadas cartesianas X, Y y Z.
- *Cilíndricos*: Combinan desplazamientos lineales y de rotación en sus componentes para posicionarse de acuerdo a las coordenadas del sistema de referencia cilíndrico
- *Esférico o de Rótula*: realizan movimientos rotatorios en sus componentes para lograr posiciones de acuerdo a las coordenadas del sistema de referencia esférico o polar.

Según el medio donde actúan

- *Terrestres*: Son el tipo más famoso (y también económico) que se puede desarrollar. Desempeñan sus tareas sobre diversos terrenos haciendo uso de variados tipos de sistemas de locomoción como ruedas, orugas, piernas, etc
- *Marinos*: Realizan sus actividades en medios acuáticos. Se caracterizan por su capacidad para desplazamientos tridimensionales y regularmente representan un gran reto en su construcción, tanto en la parte mecánica como en la electrónica, debido al medio hostil al que se enfrentan.
- *Aéreos*: Contienen características similares a los robots acuáticos, en cuanto a su libertad para poder realizar movimientos tridimensionales. Representan grandes retos en su construcción debido a que deben de tener la capacidad de levitar y mantener el control de sus movimientos en el medio aéreo.

Según el nivel de la Inteligencia que poseen

- *Autónomos*: Tienen la capacidad de procesar la información que recolectan de su medio, para que de esta forma puedan modificar su comportamiento, decidiendo o planificando sus acciones, sin la intervención humana.
- *Control Automatizado o telemanipulados*: Son robots automáticos que poseen sistemas de control centralizados y manipulados por un operador (humano), quien es el encargado de recibir los datos, procesarlos y tomar las decisiones o planificar sobre las acciones que realizará el robot.

2.4 Morfología del Robot

La morfología del robot se refiere a las características físicas con las que cuenta la estructura mecánica del robot describiendo las propiedades que tienen sus componentes tales como: los grados de libertad, sistema de referencia con la que se describen sus trayectorias, así como las configuraciones necesarias para realizar ciertas actividades

Manipuladores

Se pueden definir como cadenas cinemáticas abiertas, compuestas por eslabones unidos por articulaciones que le permite a cada elemento realizar movimientos relativos e independientes entre sí [Ollero B. Anibal, 2001]. Al número de direcciones o parámetros independientes hacia los que se puede mover cada elemento se les llama grados de libertad.

Algunos tipos de articulaciones comúnmente utilizadas son: de rotación, prismática, cilíndrica, plana y esférica o rotular (*Tipos de articulaciones en manipuladores*, Figura 2.1).

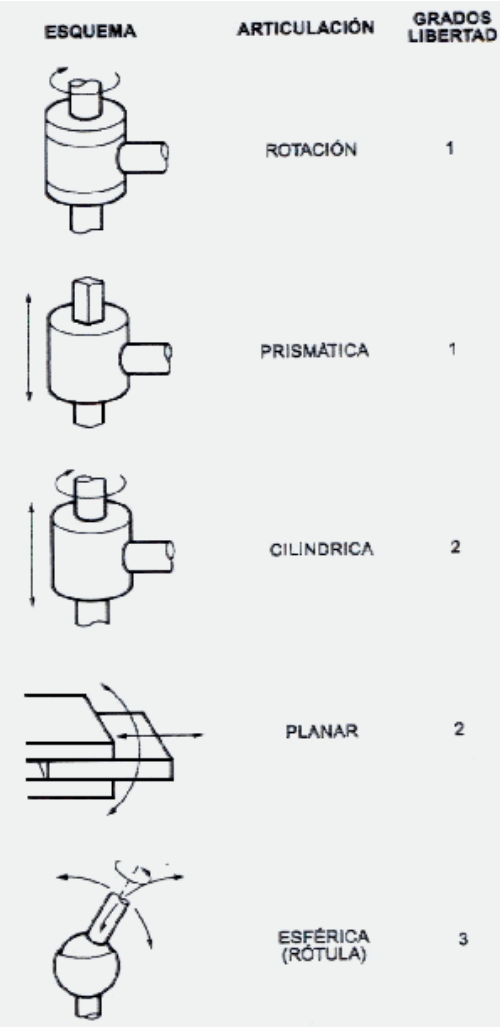


Figura 2.1. Tipos de articulaciones en manipuladores [1]

En la articulación de rotación se tiene un grado de libertad el cual consiste en un movimiento de giro alrededor de un eje fijo. En la prismática, se tiene un grado de libertad permitido por el desplazamiento lineal de un elemento a lo largo de un eje fijo. En la cilíndrica se tienen dos grados de libertad, que resultan de la combinación de una rotación y una traslación. En la planar existen dos grados de libertad como resultado de un movimiento a través de un plano. Finalmente en la esférica se cuenta con 3 grados de libertad obtenidos por 3 movimientos de rotación con respecto a 3 ejes perpendiculares.

Dentro de las configuraciones básicas de los manipuladores se encuentran:

Cartesiana: hace uso de articulaciones prismáticas para generar desplazamientos sobre 3 ejes perpendiculares. El posicionamiento de su efecto final se define a partir del sistema de referencia cartesiano.

Cilíndrica: hace uso de articulaciones de rotación y de traslación para generar dos desplazamientos sobre un plano y una rotación respecto a un eje perpendicular al plano. El posicionamiento del efector final se define a partir del sistema de referencia cilíndrico.

Esférica: hace uso de 2 movimientos de rotación y uno de desplazamiento. El posicionamiento del efector final se define a partir del sistema de referencia polar.

Robots Móviles

Los robots móviles son aquellos que tienen la capacidad de desplazarse a través del medio en el que operan.

La forma más rápida de dotar a un robot con movimiento es a partir del uso de ruedas, las cuales son muy útiles en terrenos duros y libres de obstáculos además de

permitirle al robot alcanzar velocidades altas en el desplazamiento, tener una mejor eficiencia en el consumo de energía, disminución las dimensiones del modelo, un mayor control sobre las cargas útiles que puede soportar y la maniobrabilidad que puede alcanzar. La máxima maniobrabilidad que se puede tener, esta en los vehículos omnidireccionales, los cuales pueden desplazarse de manera simultánea e independiente a los ejes del sistema de coordenadas.

Sin embargo se puede enfrentar a situaciones como deslizamientos en el momento del impulso al encontrarse en superficies demasiado lisas o suaves, o la incapacidad de alterar su estabilidad interna para lograr adaptarse a las variaciones del terreno en el que trabajan siendo estas algunas de sus desventajas.

Configuraciones típicas.

Ackerman

Es la comúnmente utilizada en vehículos de cuatro ruedas. Consiste en dos ruedas que realizan el impulso y dos que proporcionan la direccionalidad del vehículo; dichas ruedas están unidas por un eje cuyas prolongaciones a su vez intersecan con las prolongaciones del eje de las llantas traseras (*Configuracion Ackerman*, Figura 2.2).

Cuando se produce un giro para dar dirección al vehículo, la rueda más interna (la más próxima al punto respecto al que se gira) tiende a girar un ángulo ligeramente más amplio que la más externa para eliminar los deslizamientos. Su principal desventaja tiene que ver con la limitación de la maniobrabilidad del vehículo.

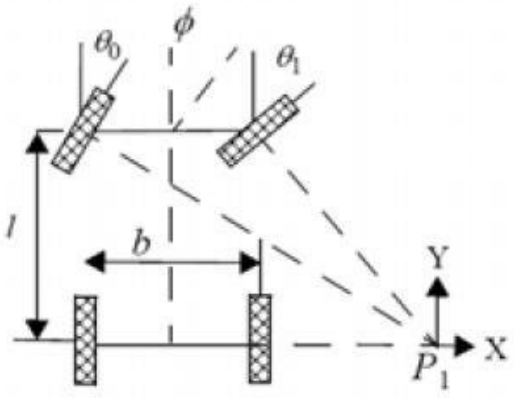


Figura 2.2. Configuración Ackerman[2]

Triciclo Básico.

Posee 3 ruedas, una delantera y dos traseras. En su rueda delantera tiene la capacidad de direccionamiento así como de tracción. Las ruedas traseras son de apoyo y se mueven libremente (*Triciclo Básico*, Figura 2.3). Su grado de movilidad es mayor a la Ackerman. Presenta problemas de estabilidad al enfrentar terrenos complicados además de que su centro de gravedad se desplaza al recorrer pendientes y en consecuencia su tracción disminuye.

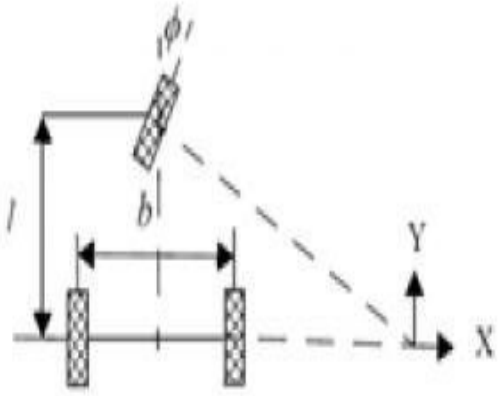


Figura 2.3. Triciclo Básico [3]

Direccionamiento Diferencial.

Es un modelo muy utilizado en robots móviles actuales. Su locomoción se da por dos ruedas laterales que poseen la tracción del sistema y aparte tienen otras ruedas más que son solo de apoyo. Para realizar un giro hacia alguna dirección, basta con variar las velocidades en sus dos ruedas principales (*Configuración Diferencial*, Figura 2.4).

Si la izquierda gira menos que la derecha, el robot gira hacia la izquierda. Si la derecha gira menos que la izquierda, el robot gira a la derecha. Si ambas giran en sentidos contrarios, el robot gira sobre su propio eje. Si giran en el mismo sentido hacia el frente, el robot avanza. Si giran en el mismo sentido hacia atrás, el robot retrocede.

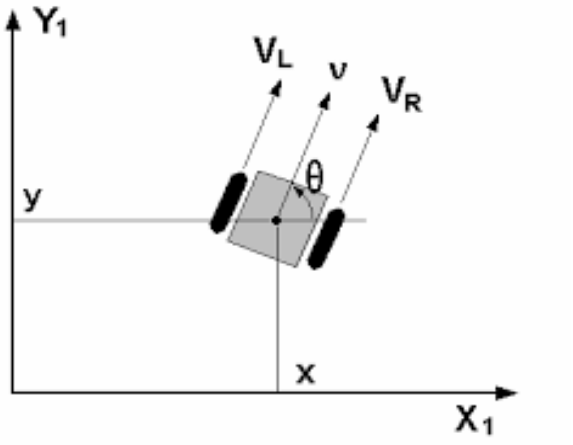


Figura 2.4. Configuración diferencial [4]

Otras configuraciones

- Skid Steer: Poseen varias ruedas en cada lado. Su movimiento se da a través de la combinación de las velocidades de cada rueda.
- Pistas de deslizamiento: Son vehículos que poseen “Orugas”. Su impulso y direccionamiento se dan a través de dichas orugas.

Locomoción mediante patas

En este tipo de locomoción, el cuerpo del robot se mantiene aislado del terreno empleando únicamente puntos discretos de soporte (*Robot spot-mini*, Figura 2.5). El polígono de soporte se puede adaptar para mantener la estabilidad y superar diversos obstáculos. Ejercen mayor omnidireccionalidad, pero representan mecanismos más complejos y que consumen mucha energía.

Figura 2.4. Robot spot-mini, Boston Dynamics [5]

Configuraciones Articuladas

Fueron pensadas específicamente para la exploración de terrenos más hostiles. Generalmente este tipo de robots están articulados mediante eslabones que representan varias ventajas; tales como, explorar zonas estrechas, y garantizar mayor seguridad de movimiento, ya que cada eslabón es un módulo que puede poseer locomoción por ruedas o patas.

Capítulo 3. Componentes del robot I: Unidades de control e inteligencia

En este capítulo se comenzará a desarrollar un tema que tienen que ver con el objetivo particular III, que propone:

- El robot posee la capacidad de ejercer sus tareas de manera autónoma, es decir, que no requiera de la intervención humana para realizar todas sus actividades

Se ha decidido comenzar a desarrollar a partir de este objetivo, ya que para lograrlo se hará uso de componentes que permitan implementar el control de las señales de entrada y salida del robot, además de su inteligencia; y como el resto de los componentes de los que se hablará más adelante dependen del desarrollo de algoritmos en lenguajes de programación que a su vez son dependientes de la plataforma con la que se esté trabajando, se ha considerado que es un buen comienzo para familiarizarse y entender de lo que se hablara en los capítulos posteriores cuando se trate los temas de sensores, actuadores, control e implementación de los algoritmos para la máquina de estados que definen el comportamiento del robot.

Los módulos elegidos para lograr este objetivo son: El micro-controlador Arduino y la placa Raspberry PI, que se describen a lo largo de este capítulo tanto en sus cuestiones técnicas como en sus implementaciones y roles que cumplirán en el robot.

3.1. Placa Arduino UNO

Arduino es un conjunto de elementos que constan de:
Hardware: Básicamente una placa impresa en la que se encuentra adherido un microcontrolador de la marca ATMEL, una serie de pines de conexión que se comunican a los puestos de I/O del Micro-controlador, y circuitos de extras que sirven para la alimentación y la comunicación serial (por USB) del microcontrolador con la computadora desde donde se programa.
Software: Un entorno de desarrollo integrado (IDE) que contiene las herramientas necesarias para crear programas Arduino, tales como el editor de texto, el compilador, terminales, bibliotecas, etc.
Lenguaje de Programación: Al igual que otros microcontroladores, Arduino cuenta con su propio lenguaje de programación basado directamente en el lenguaje de programación C/C++, llamado processing, por lo que se ofrece la posibilidad de desarrollar programas de forma sencilla para el microcontrolador que utiliza.

Para este proyecto se decidió utilizar Arduino como el micro-controlador principal del robot debido a la serie de ventajas que representa su uso

- Arduino es libre: sigue la filosofía del Software (y Hardware) libre y de código abierto; por lo que se tendría accesos libre al kit de herramientas necesarias para desarrollar (IDE + Bibliotecas + Lenguaje) los programas que van a controlar a los sensores y actuadores del robot.
- Arduino es sumamente popular: Debido a que se trata de un proyecto de software/hardware libre, existe una enorme comunidad de desarrolladores que comparte bibliotecas, esquemas, documentación, foros y una enorme cantidad de soporte. Gracias a esto, es posible tener acceso a muchas herramientas que serán

muy útiles para sacarle provecho a la placa en el desarrollo de este proyecto de robótica, sobre todo cuando lleguemos a utilizar una de las herramientas más interesantes que se tratarán en este proyecto, el cual es ROS y del que ya hablaremos más adelante.

- Arduino es Accesible: Aunque se haya dicho que Arduino es Hardware libre, no quiere decir que sea totalmente gratuito. Como estos micro-controladores vienen ya ensamblados en una placa de desarrollo, se requiere de comprarlas; lo cual no es mucho problema, debido a que sus precios (según el modelo de la placa, UNO, NANO, MEGA) son bastante accesibles; aproximadamente unos $23 USD para el tiempo en el que se escribe este trabajo. Además como se puede acceder a la documentación del hardware, es posible armar una placa propia o comprarla de origen genérico, sin comprometer el funcionamiento lo cual reduce aún más su precio.

Teniendo en cuenta las ventajas anteriormente presentadas y también las necesidades que se requieren atender para garantizar el buen funcionamiento del robot se ha decidido utilizar la placa Arduino UNO (Figura 3.1) y que además, también es la más común en el mercado.

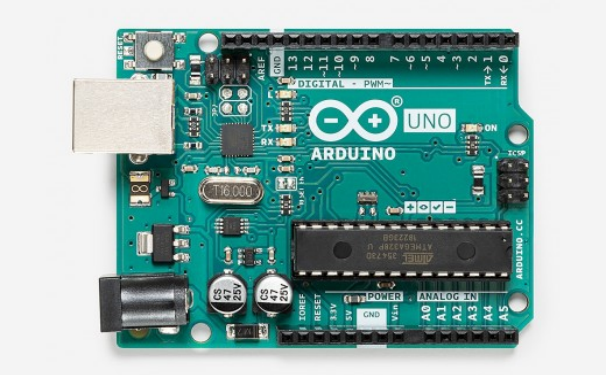


Figura 3.1. Placa Arduino UNO [6]

A continuación se describen las características de dicha placa.

Se trata de una PCB en la que nos podemos encontrar: el encapsulado del microcontrolador (DIP o SMD), Los pines hembra que son los que se utilizan para comunicarse con los puertos Digitales y analógicos, Entradas y salidas de voltaje (+5V +3V), el puerto USB tipo B, el conector para fuente de voltaje externo (+9V ~ +12V), etc.

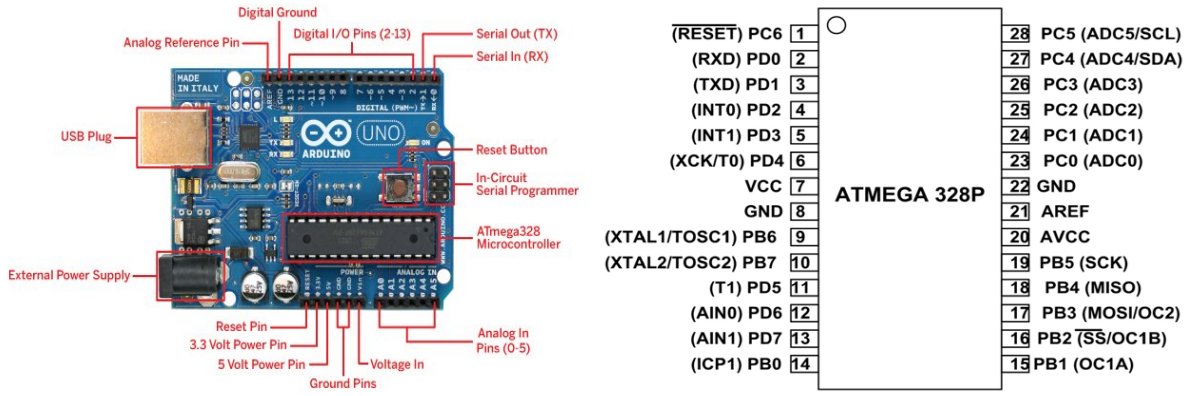
- El modelo que se está utilizando es la versión ARDUINO UNO R3 (revisión tercera) y tienen un microcontrolador ATmega328P de bajo consumo de energía y con arquitectura AVR. El encapsulado es de tipo DIP (Dual In-line Package).
- Contiene una memoria flash de 32Kb que se utiliza para almacenar el programa de forma permanente o hasta que se sobreescriba. En dicha memoria ya están ocupados 512 bytes porque está montado un gestor de arranque para facilitar el uso del microcontrolador. Sin embargo, dicha característica nos limita en el tamaño del programa que le podemos cargar para que opere el micro.
- Contienen una memoria SRAM de 2Kb para alojar datos utilizados en la ejecución del programa.
- Contiene una EEPROM de 1Kb para almacenar de forma permanente datos de programa con los que se desee arrancar la próxima vez el micro-controlador.

- Se pueden extender sus memorias internas con memorias externas mediante los protocolos de comunicación SPI o IIC. Esto último no será necesario hacerlo en este proyecto.

Descripción de los pines del Arduino UNO R3:

- Vin: Puede funcionar como salida de voltaje no regulada, es decir, el voltaje que entre a la placa por el conector de adaptador (9V~12V) o por el conector USB (5V) será el mismo que salga por este pin y puede servir para conectar y alimentar a un dispositivo externo. También se puede usar para conectar una fuente de alimentación externa, en este caso, si se regula el voltaje de entrada a 5V, que es lo que utiliza la placa para operar.
- GND: Conexión a tierra.
- Pin 5V: Entrada o salida de tensión máxima de 5V para alimentar a la placa u otro dispositivo externo.
- Pin 3.3V: Salida de tensión máxima de 3.3v para alimentar dispositivos que operan con bajo voltaje.
- Pines analógicos: Son 6 pines que van de A0 a A5; tienen una resolución de 10 bits y pueden medir un voltaje que va de entre 0 a 5, pudiendo modificar el tope usando el pin AREF.
- IOREF: Este pin se utiliza para indicar el voltaje que se maneja en los pines de entrada y salida (5V)
- RESET: tiene la misma función que el botón de RESET de la placa.
- PINES DIGITALES: Son 14 y van del 0 al 13. Tienen una tensión máxima de 5V
- A5 SCL y A4 SDA: Atienden comunicaciones usando la biblioteca WIRE
- AREF: Establece un voltaje de referencia para las entradas analogicas
- 1 TX y 0 RX: Pines para establecer una comunicación Serial.

Puede visualizar en la **figura 3.2 (a)** a la placa arduino UNO con todos los componentes listados arriba etiquetados para facilitar su identificación. Así mismo, puede ver en la **figura 3.2 (b)** el patigrama del circuito integrado Atmega328P, también con sus respectivos pines etiquetados.



(a) componentes etiquetados [7] (b) patigrama del CI Atmega328P [8]

Figura 3.2 Componentes etiquetados de la placa arduino uno

La cantidad de pines de I/O del microcontrolador son suficientes para los componentes que se requieren. No obstante, si se llegasen a utilizar más conexiones por ejemplo digitales, se puede añadir hardware extra como multiplexores para poder extender y controlar esas entradas y salidas extras, pero en este caso no será muy necesario.

En la **figura 3.3**[1] se muestra un esquema de la placa Arduino UNO R3 con sus respectivos pines de conexión. Cada Puerto, Digital y Analógico, contiene etiquetas que indican para qué componentes se planean utilizar ya sean sensores o actuadores, dependiendo del tipo de señal manejan cada uno de ellos.

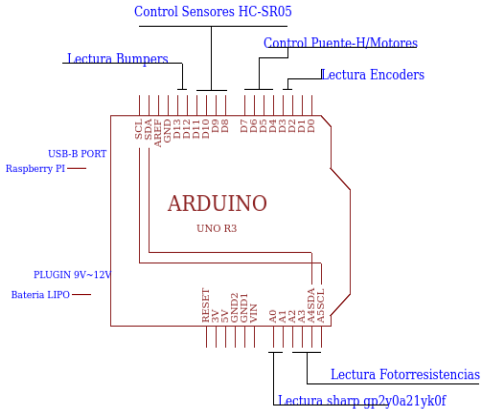


Figura 3.3. Etiquetado de puertos a utilizar

Tentativamente, el principal objetivo que tendrá la placa Arduino UNO en el proyecto, será únicamente para controlar las señales de Entrada y Salida de los sensores y actuadores del robot (aunque también es posible programar la inteligencia del robot en este mismo microcontrolador). Así, los programas alojados en esta placa solamente se encargaran de recibir y entregar a la mini-computadora los datos recolectados y después recibir los resultados del procesamiento de dichos datos y emitir las señales que permitirán que los actuadores operen para llevar a cabo las actividades correspondientes.

3.2. Placa Raspberry PI

Como se mencionó en la sección anterior, se planea hacer uso de dos dispositivos informáticos para llevar a cabo la programación del robot separando la programación embebida que solo manejara las señales de entrada y salida, de la programación que se encargará de la inteligencia.

Para esta segunda tarea, se ha decidido utilizar una placa especial; la Raspberry Pi 3 B+.

La placa Raspberry PI tal cual, es una mini-computadora fabricada por la fundación inglesa Raspberry PI, que al igual que Arduino, comenzó con la idea de permitir la accesibilidad de los dispositivos informáticos a las comunidades académicas, por lo que se realizó un gran esfuerzo en fabricar una computadora muy potente pero pequeña y económica que cumpliera con esos fines.

Sin embargo, el éxito que tuvo Raspberry PI fue más allá de lo que originalmente se tenía planeado por lo que se extendió su accesibilidad a más grupos que cuya fascinación era la electrónica y la computación, logrando utilizar dicha computadora en numerosos proyectos sobre estos temas, y entre ellas por supuesto, la robótica.

Características de la Raspberry PI 3 B +

[1] Esta figura es parte de los diagramas técnicos donde se describe la conexión de los componentes del robot (sensores y actuadores). Puede visualizar el diagrama completo en el Apéndice A. Diagramas de conexión.

1. Se trata de la revisión final de la tercera generación de minicomputadoras Raspberry, por eso el número 3 y el símbolo + (plus)
2. Como en las generaciones anteriores, se vienen manejando dos modelos, A y B. Esto es solo para diferenciar ciertas características del hardware de cada placa:
    - Modelo A: Versión posterior de la placa Raspberry PI con un hardware reducido
    - Modelo B: Versión temprana de la placa Raspberry PI que contiene una capacidad de Hardware más extendida.

Especificaciones Técnicas tomadas desde el sitio oficial de Raspberry PI

- ➢ Procesador: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- ➢ Memoria: 1GB LPDDR2 SDRAM
- ➢ WIFI: 2.4GHz y 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- ➢ Ethernet: Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- ➢ Puerto GPIO: Extensión a 40 pines
- ➢ Salida de video: HDMI
- ➢ Puertos USB: 4 puertos USB 2.0
- ➢ Ranura de conexión para cámara Raspberry PI
- ➢ Puerto de conexión compatible con pantalla táctil para Raspberry PI
- ➢ Salida de Audio Stereo
- ➢ Puerto para conexión para MicroSD
- ➢ Alimentación: 5V/2.5A DC

Figura 3.4 Placa Raspberry PI 3 b+. Raspberry [9]

Para integrar esta placa en el robot (mostrada en la figura **Figura 3.4**), necesitamos realizar una serie de pasos extras a parte de la conexión física con los demás componentes.

Instalación del Sistema Operativo

Como se trata de una mini-computadora, es necesario tener instalado un sistema operativo compatible con su arquitectura. Para ello necesitamos utilizar un MicroSD de al menos 16 o 32 GB para poder instalar el sistema operativo más la paquetería que necesitaremos para trabajar con el robot.

El sistema operativo elegido es Raspbian Buster, que es una distro Linux basada en Debian y adaptada para funcionar en la arquitectura ARM de Raspberry. También es posible utilizar otros sistemas operativos como Ubuntu, Ubuntu Mate e incluso Windows. Sin embargo es de suma importancia tener en cuenta que necesitamos utilizar una distro Linux

ya que más adelante haremos uso del Sistema Operativo Robótico ROS, el cual tiene soporte estable únicamente para Linux.

Para montar el sistema operativo primero hay que descargar los archivos comprimidos o ISO, estos se pueden encontrar directamente en la web oficial de Raspberry PI (https://www.raspberrypi.org/software/operating-systems/) y descomprimirlo en la MicroSD y posteriormente insertarla en la placa.

Como únicamente se utilizará la Raspberry PI para trabajar en el robot, recomiendo elegir la versión “Raspberry Pi OS with desktop” o la “LITE”, para ahorrarnos espacio porque la versión recomendada por el sitio es la que contiene mucho más software que no será necesario.

Habilitando la comunicación SSH

Una de las primeras herramientas que se necesitan tener preparadas es la comunicación SSH; ya viene pre-instalada en Raspbian, solo es necesario habilitarla. Se puede hacer de dos formas:

1. Desmontar la MicroSD y desde otra computadora accederla y crear un archivo vacío en la raíz sin extensión llamado “ssh”
2. Desde el escritorio de Raspbian, accediendo a la configuración y en el panel de control buscar la opción SSH y habilitar.

El uso de la comunicación SSH será para facilitar el uso de la placa sin necesidad de conectarla a terminales como teclado, mouse y pantalla, únicamente habrá que conectarla a la alimentación y a la red de trabajo local; además teniendo instalada y configurada la comunicación SSH en una computadora externa al robot y conectada a la misma red, se podrán controlar desde una terminal de sistema las instalaciones y configuraciones de paquetes necesarios para que el robot opere.

Instalación de ROS

Una vez instalado y configurado el Sistema Operativo en Raspberry, hay que actualizar los paquetes y dependencias para que esté preparado para la siguiente instalación: El sistema Operativo ROS.

Para trabajar con ROS se necesita instalarlo tanto en la Raspberry PI como en la computadora externa, la cual obviamente debe tener una distro Linux.

En mi caso, estoy utilizando la distribución KUBUNTU 18.04.5 LTS (UBUNTU con escritorio de KDE) la cual al igual que Raspbian Buster es compatible con ROS MELODIC. Por su parte, ROS Melodic guarda retro-compatibilidad con paquetes desarrollados en versiones anteriores de ROS, por lo que no se tendría mucho problema para hacer uso de ellos en el robot.

La instalación de ROS en UBUNTU para la computadora externa es muy sencilla y basta de unos cuantos comandos que sirven para acceder a los repositorios de ROS para DEBIAN/UBUNTU, y posteriormente instalar y configurar de forma automática los paquetes.

Ejemplo de instalación siguiendo los pasos recomendados desde la documentación oficial en la Wiki de ROS (http://wiki.ros.org/):

1. configuración de los repositorios y las fuentes de software

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

2. configuración de claves de acceso

```
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key -Clave-de-Acceso-
```

3. instalación de los paquetes

```
#actualización del gestor de paquetes
$ sudo apt update

# instalación del paquete completo de ROS (Se pueden instalar solo paquetes específicos
# si asi se desea)
$ sudo apt install ros-melodic-desktop-full
```

La única configuración manual que es necesaria realizar tras la instalación es la configuración del PATH para poder acceder desde la terminal a los comandos de ROS.

```
# añadido permanente del paquete ros al path del sistema
$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Por otro lado, la instalación de ROS en Raspbian tiende a ser más “latosa”, porque hay que seguir una serie de pasos que no son más que la instalación manual de todas las paqueterías y dependencias. Para ello es recomendable seguir los pasos descritos en la Wiki de ROS (documentación oficial) de lo cual dejo el enlace aquí: http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Melodic%20on%20the%20Raspberry%20Pi

Al igual que en la instalación realizada en la computadora externa, solo se añade ROS al PATH del sistema para poder acceder a los comandos.

Retomaré ROS hasta el capítulo correspondiente donde ya se tratará más detalle sobre lo que es ROS y el porqué será utilizado en este proyecto. Mientras tanto, hasta aquí se tiene lista la placa RaspBerry para utilizarla en el robot.

Conexión de la Raspberry PI en el robot.

El último paso a realizar en esta sección tiene que ver con la integración de la placa al hardware del robot. En este paso se debe de tener en cuenta un par de cosas importantes:

1. La placa Raspberry PI servirá para ejecutar la inteligencia del robot, así que es muy necesario lograr la comunicación con la placa Arduino UNO, de quien obtendrá los datos que necesita procesar y posteriormente compartir los resultados para que se ejecuten las acciones correspondientes en los actuadores del robot.

2. La placa necesita alimentarse con un voltaje y corriente específicos, así que será muy importante tener cuidado con el suministro eléctrico, de lo contrario se pueden causar daños irreversibles.

Según lo anterior, para comunicar la placa Raspberry PI y la placa Arduino UNO se hará uso de la comunicación serial. Para ello hay dos opciones:

- Conectar ambas placas mediante un cable USB
- Conectar ambas placas mediante los pines de comunicación que poseen cada uno: Pines digitales 0 y 1 (RX y TX) del Arduino, Pines RX y TX del puerto GPIO de Raspberry. Para estos últimos se debe de tener en cuenta otra vez, los voltajes y corrientes de operación; 3.3V en GPIO Raspberry PI y 5V en Arduino.

Para este caso se ha decidido utilizar la comunicación serial mediante cable USB, ya que es mucho más práctico, debido a que aquí no va a causar mucha preocupación los voltajes y corrientes de operación de cada puerto de cada placa, ya que el puerto USB en cada caso regula la señal de entrada y salida, a diferencia de utilizar los puertos digitales y GPIO, donde si se necesita agregar más hardware para regular las señales y proteger ambas placas.
Por otro lado, en la alimentación también hay dos vías.

- Tener una batería adicional para alimentar a la Raspberry PI a través de su puerto microUSB
- Utilizar la misma batería del robot, pero añadiendo hardware adicional que permita regular la corriente de alimentación.

Aquí se ha elegido la segunda opción, esto para evitar poner más elementos que incrementen la carga y tamaño del robot y además se permita que desde un solo control de interrupción se puedan encender/apagar todos los módulos del robot.

El regulador utilizado se muestra en la **figura 3.5**:

Figura 3.5: Regulador de voltaje de bajada Pololu [10]

Este regulador conecta su pin de Voltaje de entrada a la batería del robot y la salida del voltaje regulado a 5V y ~2.5A se conecta en los pines GPIO que soportan entrada de 5V. El Pin GND se conecta al común (Tierra) del circuito del robot.

En la **figura 3.6**[2] se muestra el esquema de conexión que se utilizará para comunicar al microcontrolador arduino UNO y la Raspberry PI a través del puerto USB, indicando también la conexión del regulador para alimentarla debidamente.

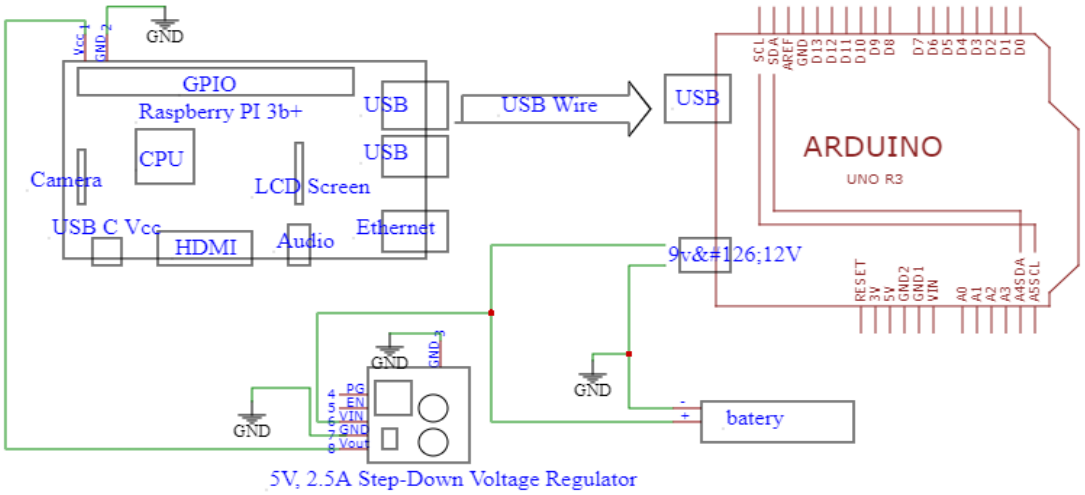


Figura 3.6 Conexión entre placas Arudino UNO y Raspberry Pi.

[2] Esta figura corresponde nuevamente a los diagramas de conexión de los componentes del robot. Puede consultarla en el Apéndice A.

Capítulo 4 Componentes del robot II: Sensores y Actuadores

Recordando nuevamente los objetivos particulares propuestos para este robot, encontramos lo siguiente:

- El robot posee la capacidad de medir su entorno para reconocer la existencia de posibles obstáculos u objetivos que le obliguen alterar su funcionamiento y ejecuta diversas acciones para que esta forma pueda adaptarse a las condiciones actuales.
- El robot posee la capacidad de medir su estado interno para ajustar sus valores de modo que esto le permite mejorar el desempeño de sus funciones de navegación en el medio.

Para conseguir el cumplimiento de estos objetivos, se debe de dotar al robot con componentes que le permitan, entre otras cosas:

- Ejecutar sus acciones: En este caso, desplazarse por su medio.
- Capturar datos de su entorno (Sensores externos): Debe de contar con elementos que le permitan conocer qué hay en el medio en el que trabaja y a partir de ello tomar acciones para adaptarse.
- Capturar datos de sí mismo (Sensores internos): Debe de ser capaz de monitorear su estado interno y de modo que realice los ajustes necesarios que le permitan realizar bien su acciones.

Los componentes referidos aquí son los sensores y actuadores del robot. A lo largo de este capítulo expondré sobre los diferentes sensores y actuadores que se eligieron para poder realizar cada una de las actividades que permiten cumplir con los objetivos propuestos. Además se describe a detalle sobre la forma en que se integrarán al robot y cómo se harán funcionar.

Por mera decisión propia sobre el orden correcto para desarrollar este capítulo, primero comenzare a explicar los actuadores del robot, ya que algunos de los sensores y su funcionamiento de los que trataré aquí, trabajan directamente en el actuador; así que lo más conveniente, es conocer primero el actuador y después al sensor.

También se hablará de otros componentes importantes para el robot que son la fuente de alimentación y el sensor que notificara cuando esta se requiera recargar para continuar con el funcionamiento.

4.1 Actuadores

Como una de las principales actividades que debe de realizar el robot es la navegación, se requiere de actuadores que le permitan desplazarse por el medio en el cual va a operar. Como ya se mencionó en la lista de componentes, los actuadores elegidos para este robot son los motores de corriente directa. A continuación describo sobre los motores elegidos y su integración en el robot.

4.1.1. Motores DC

Debido a los fines para los que se construye este robot, se eligió un tipo de motores que se ajustan a las necesidades y presupuesto para este proyecto.

En este caso se decidió utilizar un par de motores de Corriente Directa (DC) con eje extendido y con micro reductor metálico de relación 1:100 marca POLOLU (**Figura 4.1**). A continuación se lista una serie de ventajas que presentan este tipo de motores:

- Son motores bastante compactos. Sus dimensiones según su hoja de datos: 10 X 12 X 26 mm. Esto permite tener la posibilidad de ensamblar un robot bastante compacto.
- Los motores son de alta calidad y potencia, ya que poseen un torque de 1.7 kg-cm y pueden funcionar con un voltaje de alimentación que va desde 3V hasta 9v, iniciando su rotación desde los 0.5V. Esto permitirá que el robot pueda alcanzar velocidades altas, además de que va a tener la fuerza suficiente para moverse aún con la carga adicional de los demás componentes, y no se requerirá de demasiada energía para lograr ambas tareas.
- Poseen un eje estándar en forma de D de 9.3mm de largo y 3mm de diámetro a la salida del micro reductor, lo cual le ofrece compatibilidad con una amplia gama de neumáticos.
- El precio de estos motores, para el tiempo en el que se desarrolla este trabajo, es alrededor de $300 MXN por unidad.

Figura 4.1. Motor con micro reductor metálico de relación 100:1 marca pololu. [11]

Para poder utilizar estos motores en el robot será necesario instalar primero los módulos o sensores que permitirán la medición de la posición y velocidad del robot (esto se trata en la siguiente sección) además de colocar la rueda en el eje de salida del micro-reductor con el cual se podrá desplazar el robot.

Normalmente estos motores con micro-reductor se venden junto a unos soportes de plástico para que puedan ser fijados en el chasis del robot, por lo que para terminar su instalación, solo hay que colocar el motor en el sitio correspondiente en el chasis y después fijarlo con ese soporte.

Adicionalmente se deben de colocar las ruedas con las cuales el robot podrá desplazarse por el medio. Las ruedas utilizadas en este robot son las llantas de 49X19 mm para micro-reductores marca Pololu (**Figura 4.2**).

Fig 4.2. Par de Llantas 42x19mm Para Micro Motorreductor [12]

Características principales:

- Ruedas de goma de 49x19mm
- Entrada tipo D para ejes de salida de 3mm
- Soporte de plástico dentado internamente que es de utilidad para utilizar un encoder óptico.

Instalación: Únicamente insertar la entrada de la rueda en el eje de salida del micro-reductor.

4.2. Sensores

En esta sección se explicará sobre los sensores con los que se ha equipado este robot para cumplir con los objetivos del proyecto, así como el código utilizado para manejar adecuadamente las señales recibidas por los sensores y de qué forma se planean utilizar esos datos.

4.2.1. Sensores de posición y velocidad

Medir la posición y velocidad del robot serán una tareas sumamente importantes que nos permitirán la implementación de sistemas para controlar la navegación y la posición del robot.

Para llevar a cabo dichas tareas, se eligió utilizar el encoder magnético para micro-reductores metálicos de eje extendido marca POLOLU (**Figura 4.3**); el cual nos permite medir mediante pulsos digitales el número de giros que ha realizado el eje del motor eléctrico y a partir de esos datos, calcular a través un análisis cinemático del robot, las distancias recorridas y sus respectivas velocidades lineales y angulares.

A continuación se lleva a cabo la descripción de dichos encoders:

El encoder magnético de la marca POLOLU, consta de una placa de circuito impreso en el que están conectados dos sensores de efecto hall, un disco magnético que se coloca en el eje del motor y seis pines de entrada y salida repartidas como sigue:

- 2 pines que se conectan a los polos del motor
- 2 pines para los canales digitales A y B que son las salidas producidas por las lecturas de los sensores de efecto Hall
- 2 pines de alimentación Vcc y GND para los sensores de efecto hall.

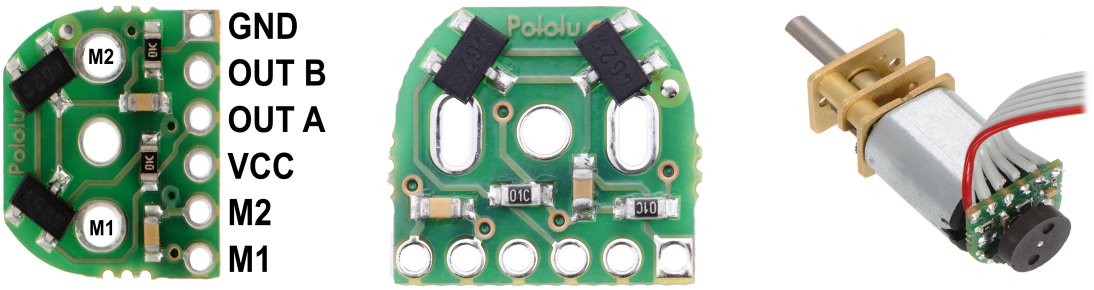


Figura 4.3. Encoder magnético Pololu e instalación en el motor [13]

Este encoder está pensado para usarse en motores con micro reductor metálico de eje extendido, ya que el encoder se instala en la parte trasera del motor de modo que este módulo esté soldado en los pines de los polos y sobre su eje extendido esté colocado el disco magnético.
Su funcionamiento se puede explicar de forma sencilla:

- Cuando el eje del motor gira una vez (o se produce una revolución), el disco magnético genera 12 pulsos eléctricos en los sensores de efecto hall, ya que detectan los campos magnéticos generados por los micro imanes incrustados en el disco magnético instalado en el eje.
- Esos pulsos eléctricos son enviados como señales de onda cuadrada a través de los canales A y B. Estas señales digitales se pueden enviar directamente a un micro-controlador o circuito digital para ser procesadas.

Cabe resaltar que estas señales digitales que envían a la salida del encoder los sensores de efecto hall, pueden servir para calcular distancias recorridas por la rueda utilizando un contador de pulsos, así como también el sentido de giro de la rueda, ya que existe un desfasamiento de 90º entre las señales digitales de cada canal y, si por ejemplo, se analizan las señales en un Osciloscopio (**Figura 4.4**), se pueden presentar algunas de las siguientes situaciones:

- la señal del canal A está adelantada a B, entonces el motor gira en sentido horario
- la señal del canal B está adelantada a A, entonces el motor gira en sentido antihorario.

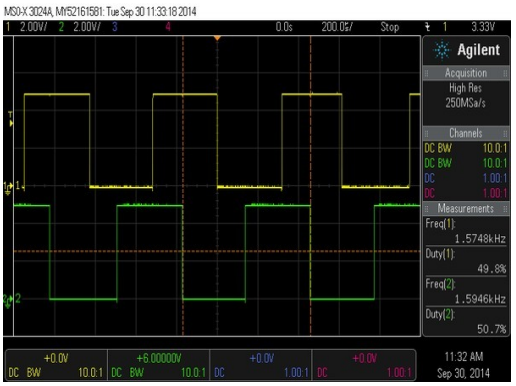


Figura 4.4. Salida de los canales A y B del encoder [14]

Conexión del encoder a Arduino y configuración de los puertos:

Una vez instalado el encoder en el motor, la forma en que se utilizará será conectando uno de sus canales digitales (A o B) al microcontrolador Arduino UNO, para procesar las señales que se reciba y llevar a cabo el cálculo de distancias y velocidades del motor. También se hará uso de un elemento extra que será de utilidad para controlar ambos motores a partir de las señales provenientes de Arduino con las que se activan los motores, y también se definirá la dirección del giro de cada motor. Este elemento es el circuito integrado L293B que es ni más ni menos que un puente H.
En los siguiente esquemas se muestran las conexiones de los elementos mencionados.

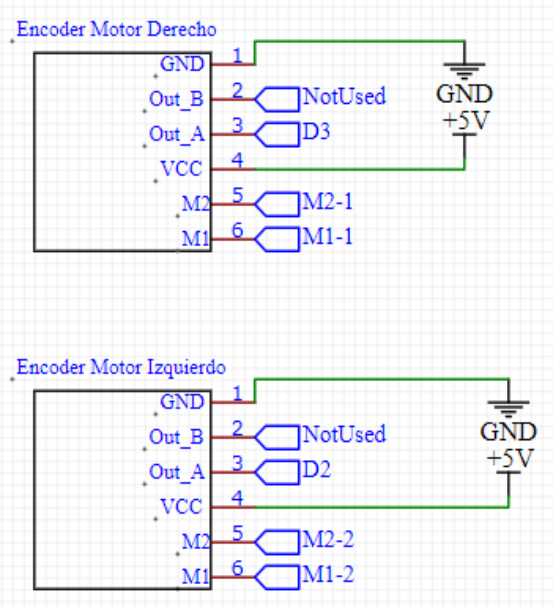


Fig 4.5. Conexión de los encoders

Encoders: Los encoders se conectan como se muestra en la figura 4.5. Los pines de alimentación Vcc y GND se conectan a los pines +5V y GND que provee Arduino UNO. Los pines M1 y M2 de cada encoder están conectados a los pines 3, 6 (para el derecho), 14 y 11 (para el izquierdo) del puente H; estos pines son las salidas de dirección. Los pines Out_A de cada esquema, representan el canal de salida digital A de cada encoder. Estos están conectados directamente a Arduino en los puertos digitales 3 y 2, los cuales también son pines de interrupción por Hardware; esto resulta ser bastante conveniente ya que de esta forma, cada vez que un encoder genera un pulso por el avance del giro de la rueda, Arduino interrumpe la ejecución de las demás funciones para simplemente incrementar el registro de pulsos de cada rueda y de esta forma tener la información que se necesita para calcular la posición del robot y su velocidad.
Puente H: El circuito integrado L293B será de apoyo para controlar las señales provenientes de Arduino con las cuales se activará o desactivará el giro del motor, se indicará la dirección de giro de cada rueda, y también se proveerá de la alimentación que requieren los motores para funcionar.

- Pines 1 y 9 se conectan a los puertos digitales 5 y 6 de Arduino. Estos pines tienen la capacidad de manejar señales PWM, siglas en inglés de Modulación de Ancho de Pulso. El PWM es capaz de imitar una señal analógica, lo que me permitirá enviar pulsos con que se puede hacer que el motor gire mas rápido o mas lento, o sea, pulsos que controlen la velocidad de giro de la rueda.
- En los pines 2 y 7 para el motor derecho y 15 y 10 para el motor izquierdo, solo se envían señales digitales (1 o 0) desde los puertos 4 y 7 de Arduino para indicar la dirección de giro. La compuerta lógica Not (7404) solamente es auxiliar para lograr este control.

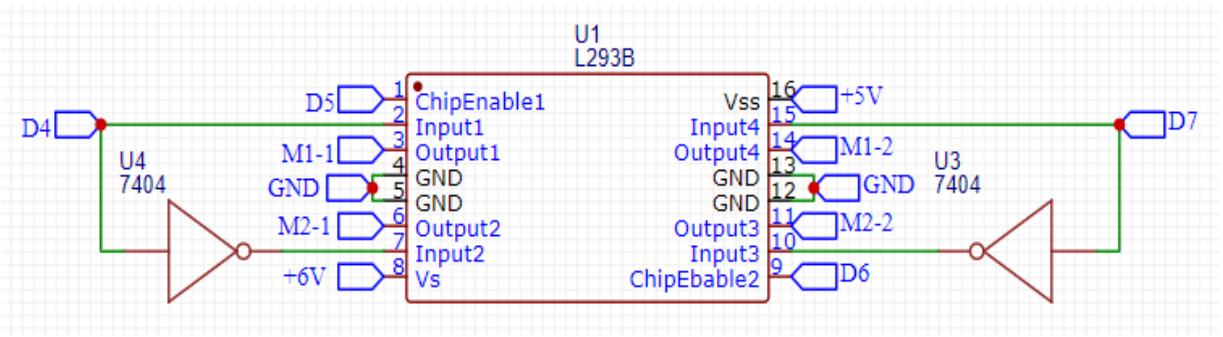


Fig 4.6. Conexión del Puente H

Conexión en Arduino: en la figura 4.7 se muestra mediante etiquetas, la conexión de los pines Arduino con los elementos anteriormente mencionados.

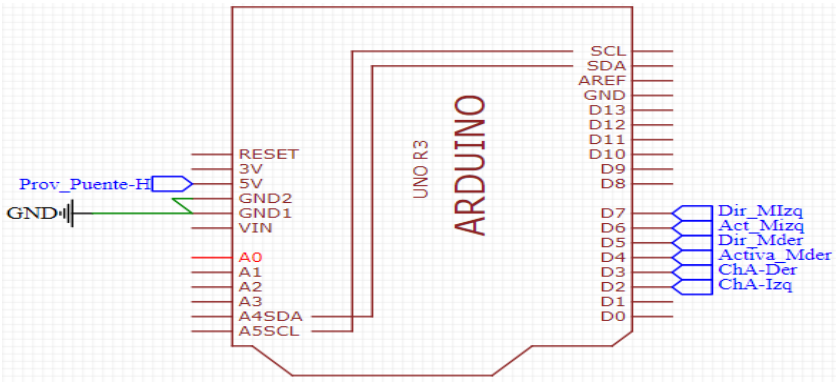


Fig 4.7. Conexión del Encoder y Puente H con la placa Arduino

Una vez conectados físicamente los puertos Arduino con los pines del Integrado L293B y los pines del módulo encoder, se debe de realizar la configuración correspondiente en el código de Arduino para asegurar la comunicación de las señales de entrada y salida que se van a manejar. A continuación se muestra la configuración utilizada.

Declaración de macros para identificar los puertos que van a manejar las señales de entrada y salida para los motores y el encoder.

```
1 //puertos de salida para el control de dirección y velocidad mediante PWM
2 #define V_MRIGTH 5
3 #define D_MRIGTH 4
4 #define V_MLEFT 6
5 #define D_MLEFT 7
6
7 //puertos de entrada para la lectura de las señales digitales del encoder
8 #define RH_ENCODER_A 3
9 #define LH_ENCODER_A 2
```

Declaración de variables que van a guardar el registro de pulsos censados desde el encoder. Estas variables son del tipo volatile unsigned long.

```
1 //variables para almacenar el registro de pulsos generados en el encoder
2 volatile unsigned long left_count_ticks = 0;
3 volatile unsigned long right_count_ticks = 0;
```

El modificador volatile en el lenguaje C++ (Processing para Arduino), le indica al programa que la variable que se va a manejar cambia rápidamente su valor. Al declarar la variable del tipo unsigned long, se prepara a Arduino para que reserve el espacio suficiente para almacenar números bastante grandes (32 bits), ya que no se sabe cuantos pulsos se van a registrar cuando el robot esté en movimiento.

Configuración de los puertos en la función setup() de Arduino para que al compilar el programa, el microcontrolador “sepa” cómo va a manejar sus puertos (para entrada o salida de señales).

```
1 //funcion Arduino para configuración del microcontrolador
2 setup(){
3  Serial.begin(9600);
4  //configuracion de pines de salida, direccion
5  pinMode(D_MRIGTH, OUTPUT);
6  pinMode(D_MLEFT, OUTPUT);
7  //configuracion de los pines de interrucion para los encoders
8  attachInterrupt(digitalPinToInterrupt(LH_ENCODER_A), leftEncoderEvent, FALLING);
9  attachInterrupt(digitalPinToInterrupt(RH_ENCODER_A), rightEncoderEvent, FALLING);
10 }
```

En el código anterior dentro de la función setup() se indica mediante las funciones pinMode() (Líneas 5 y 6), el número de puerto y tipo de señal que maneja, en este caso, el número está indicado por las macros y la señal es de salida (output). El método begin del objeto Serial de Arduino solo es para indicar la frecuencia con la que el microcontrolador se comunicará con la computadora externa mediante USB.

La función attachInterrupt() recibe tres parámetros: el primero le indica a partir de qué pin va a recibir la señal de interrupción. La función que se pasa como parámetro aquí, digitalPinToInterrupt(), recibe a su vez el número del pin de interrupción, lo configura y devuelve un valor entero (0 o 1), que es el que utiliza attachInterrupt() para identificar el pin desde el que va a atender la interrupción.

El segundo parámetro de attachInterrupt(), recibe un apuntador a la función que va a ejecutarse cuando se genere la interrupción. El tercer parámetro, recibe una constante que le indica específicamente el evento que activara la interrupción; esta puede ser LOW (0 a la entrada del puerto), CHANGE (cambio de valor en el puerto), RISING (valor cambia de 0 a 1 en el puerto), FALLING (valor cambia de 1 a 0 en el puerto), HIGH (valor alto en el puerto).

En este caso, yo decidí utilizar el FALLING, o sea que la interrupción se lanzará cuando se detecte un flanco de bajada desde la señal entregada por el encoder. No hay algún motivo en especial, la decisión de que evento utilizar queda en manos de quien realice la programación.

Hay que tomar en cuenta también que en el modelo de Arduino que se está utilizando, solamente se cuentan con 2 pines para interrupciones. Al inicio de esta sección señala que el encoder utilizado entrega dos canales, y que el uso de ambos nos puede permitir el conteo de pulso generados y también el sentido del giro de la rueda.

Como se está limitado en pines de interrupción y se requiere censar dos motores, se ha decidido utilizar únicamente un canal para cada encoder, en este caso el canal A; por lo que al realizar el registro, se debe de tener en cuenta que el encoder solo me estaría entregando la mitad de pulsos que realmente se están generando. Este dato se va a tomar en cuenta para el capítulo 6, cuando se trate sobre el cálculo de posición del robot.

Algoritmo para utilizar el encoder y motores en Arduino.

Una vez ya realizada la configuración de los pines del puerto digital que se usarán para las señales de control del motor, se muestra y explica a continuación el algoritmo necesario y su programación correspondiente para atender las interrupciones y actualizar el registro de pulsos enviados desde los encoders, además de enviar las señales de salida que indican la velocidad y dirección de giro de los motores.

En el siguiente diagrama de flujo se resumen las tareas que se deben de ejecutar para activar el giro de los motores y censar los pulsos generados.

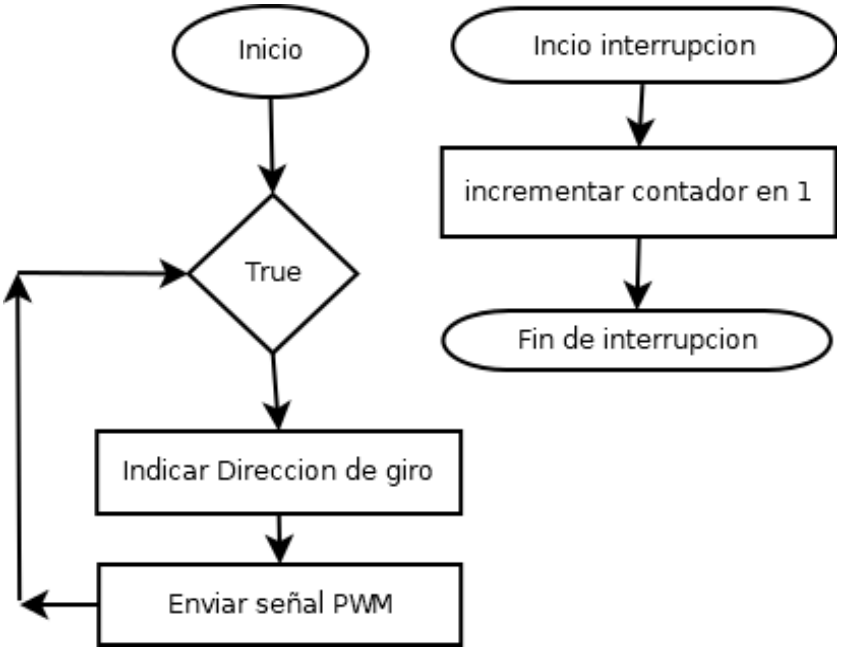


Figura 4.8. Diagramas de flujo para control de motores DC y encoder.

El algoritmo planteado para resolver esta primer tarea es demasiado simple. Únicamente se hace uso la función loop() de Arduino, la cual es simplemente un bucle infinito (representada en el diagrama con la condición True) que va a ejecutar todas las instrucciones escritas dentro de ella; estas instrucciones son:

Indicar la dirección de Giro: para la configuración que tiene este robot en su hardware, las ruedas giran hacia adelante o hacia atrás si el puente H recibe una señal digital en BAJO o ALTO como sigue:

- Rueda (motor) Izquierda: Con una señal HIGH (en ALTO o 1 lógico) gira hacia adelante, con una señal LOW (en BAJO o 0 lógico) gira hacia atrás.

- Rueda (motor) Derecha: Con una señal HIGH (en ALTO o 1 lógico) gira hacia atrás, con una señal LOW (en BAJO o 0 lógico) gira hacia adelante.

Con la descripción anterior, si quiero que el robot avance, tengo que enviar una señal HIGH para el motor izquierdo y una LOW para el motor derecho; si quiero que retroceda entonces tengo que enviar una señal LOW para el motor izquierdo y una HIGH para el motor derecho; si quiero que gire sobre su propio eje en sentido antihorario (Izquierda), ambos motores reciben una señal LOW y si quiero que gire en sentido horario, ambos motores reciben una señal HIGH.

Enviar señal PWM: El uso de una señal de PWM será para que las ruedas del robot reciban una señal que les permita definir su velocidad de giro. Como mencioné antes, la modulación por ancho de pulso (pwm) permite que una señal digital imite a una señal analógica, como indica la figura 4.9. Así, cuando el pulso es demasiado corto, el voltaje de salida se aproxima a cero y el motor giraría muy lento o nada; cuando el pulso es muy largo, el voltaje de salida se aproxima al referente del puerto digital (5V) por lo que el motor giraría muy rápido.

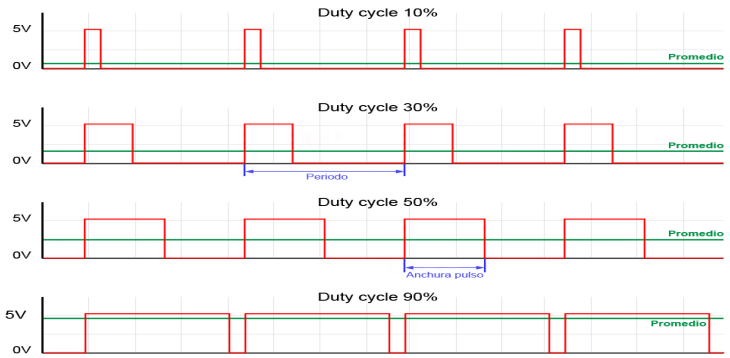


Figura 4.9. Ciclo de trabajo de la señal PWM [15]

Para poder hacer uso de la señal PWM en Arduino, se utiliza la función analogWrite() que recibe dos parámetros: el primero es el número del pin por el cual se enviará la señal, el segundo es el ciclo de trabajo representado como un número entero de 8 bits que va de 0 a 255, ya la salida entregada en este caso sería de 0 a 5 volts. Como los motores necesitan un cierto voltaje mínimo para comenzar a girar, sobre todo si tienen carga adicional, el valor del PWM elegido empíricamente es de 60 (aproximadamente 23% del ciclo de trabajo). Por el momento esta será la velocidad inicial para que el motor (y el robot) comience a moverse. El censado y ajuste de velocidades se harán en los próximos capítulos cuando se trate sobre la cinemática del robot.

El subproceso que atenderá la interrupción, solo se ejecutará cada vez que detecte un flanco de bajada en la señal entregada por el encoder. La única tarea que tiene por el momento, es la de incrementar un contador para cada rueda.

Codificación del Algoritmo

A continuación muestro y explico la implementación del algoritmo usando el lenguaje C/C++ de Arduino:

En el siguiente Fragmento de código se atienden las interrupciones y se incrementan los registros de pulsos que se reciben desde los encoders. Estas funciones están referidas por el apuntador del segundo argumento de la función attachInterrupt(); esto quiere decir que cada vez que se genere el evento que se señaló anteriormente, se mandara a llamar esta función y ejecutara los pasos señalados.

```
1 //funciones que realizan el conteo de pulsos que generan los encoders en los motores
2 void leftEncoderEvent()
3 {
4  left_count_ticks++; //acumulador de pulsos encoder izquierdo
5 }
6
7 void rightEncoderEvent()
8 {
9  right_count_ticks++; //acumulador de pulsos encoder izquierdo
10 }
```

En este otro fragmento de código se muestran las instrucciones alojadas en la función loop() de Arduino, las cuales se encargaran de indicar el sentido del giro de las ruedas para que el robot realice sus desplazamientos, en este caso, harán que se desplace hacia el frente, ya además indican el valor entero de la señal PWM para la velocidad de giro de los motores, en este caso un valor de 60.

```
1 void loop(){
2
3  digitalWrite(D_MLEFT, HIGH); //D_LEFT D_RIGHT macros
4  digitalWrite(D_MRIGTH, LOW);
5  analogWrite(V_MLEFT, M_FRONT); //variable PWM int M_FRONT = 60
6  analogWrite(V_MRIGTH, M_FRONT); //V_MRIGTH V_MLEFT macros
7
8 }
```

Al correr el programa anteriormente descrito se puede mostrar en la terminal serial el valor entero de los pulsos de cada encoder que está registrando Arduino.

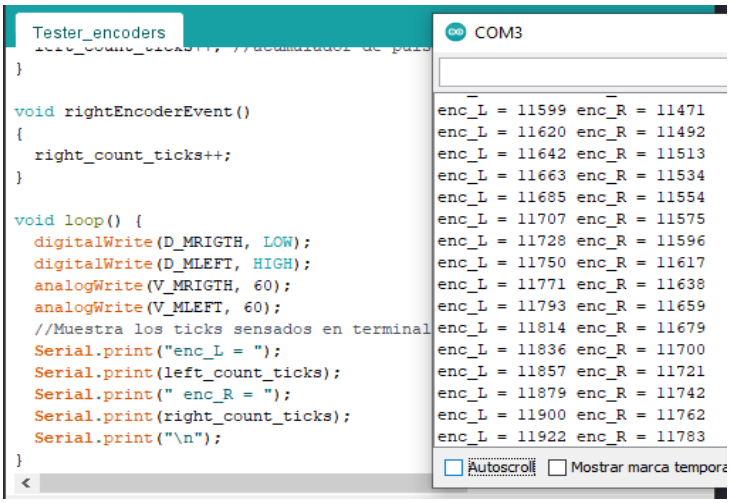


Figura 4.10. Pulsos generados por los encoders

Con lo explicado anteriormente estoy consiguiendo que el robot se desplace hacia el frente a través de una trayectoria lineal, mientras recolecta los datos que me permitirán

calcular mediante las ecuaciones de la cinemática del robot, las distancias recorridas en metros y velocidades m/s, así como también poder proponer un umbral con el cual puedo obligar a que el robot recorra una distancia D antes de detenerse o girar hacia X grados hacia alguna dirección.

Las ecuaciones que representan la cinemática del robot y su implementación se verán en los capítulos 6 y 7 respectivamente, donde se muestra un análisis sobre cómo se obtienen y cómo se implementan en el robot.

4.2.2. Sensores para detección de obstáculos

El segundo objetivo que se desea implementar en este proyecto es la capacidad de detectar obstáculos y a partir de ello alterar el estado del robot para activar algoritmos que le permitan esquivarlos y continuar con su recorrido.

La estrategia elegida para llevar a cabo la detección de objetos que impidan la navegación del robot requiere del uso de sensores capaces de medir las distancias que existen entre alguno de los lados del robot y un objeto. Si se llegan a detectar distancias muy cortas entre los objetos y el robot se entenderá que existe un obstáculo en ese punto y esto entonces modificará el estado de funcionamiento del robot.

En esta sección se describen los sensores utilizados para detectar la presencia de objetos utilizando la estrategia anteriormente mencionada. De igual manera y como en la sección anterior, explicaré los algoritmos que se utilizarán para determinar que el robot se encuentra frente a un objeto y notifique al cerebro del robot, para que este tome las acciones correspondientes.

El Sensor ultrasónico HC-SR04

El HC-SR04 es un módulo compuesto por un transmisor ultrasónico, un receptor y un circuito de control. Tiene también 4 pines de conexión que corresponden a, voltaje de alimentación (VCC), Pin IO para activar el funcionamiento del TRIGGER, Pin de salida para enviar la señal de ECHO, y una Tierra (GND) (figura 4.10). El módulo en cuestión, tiene la capacidad de medir desde 2 cm como mínimo, hasta 4 metros como máximo.

Figura 4.11. Sensor ultrasonico HC-SR04, Hardware Libre [16]

El funcionamiento básico del circuito se puede describir en 3 pasos:

1. Se activa un disparador con una señal de nivel alto por al menos 10 µs
2. Se dispara inmediatamente una ráfaga de 8 señales ultrasónicas de 40 kHz y se espera a que haya una señal de regreso.

3. Cuando se recibe la señal de retorno, se mide entonces el tiempo de envió y recepción de la señal ultrasónica.

El cálculo de las distancias a través de este módulo se hace con la siguiente fórmula:

$$D = \frac{T_{hl} \times V_s}{2} \qquad (1)$$

Donde
D = Distancia medida
Thl = tiempo de la señal de nivel alto
Vs = Velocidad del sonido en m/s en el aire

Integración del sensor al robot

Como se describió antes, el sensor ultrasónico HC-SR04 tiene 4 pines de conexión; de estos cuatro pines 2 son de alimentación y además como el sensor opera con +5v, basta con alimentarlo con los 5 volt de salida del Arduino. Los otros dos pines, TRIGGER y ECHO manejan señales digitales, así que hay que conectarlos a los pines del puerto digital de Arduino. El pin donde se conecta TRIGGER maneja una señal de salida, ya que esta se usa para activar al sensor; el pin donde se conecta ECHO maneja una señal de entrada, ya que aquí se recibe la señal que indica el tiempo que tardo en rebotar la señal ultrasónica, y partir de ella se pueden calcular las distancias empleando la ecuación anterior.

La conexión utilizada se muestra en el siguiente esquema.

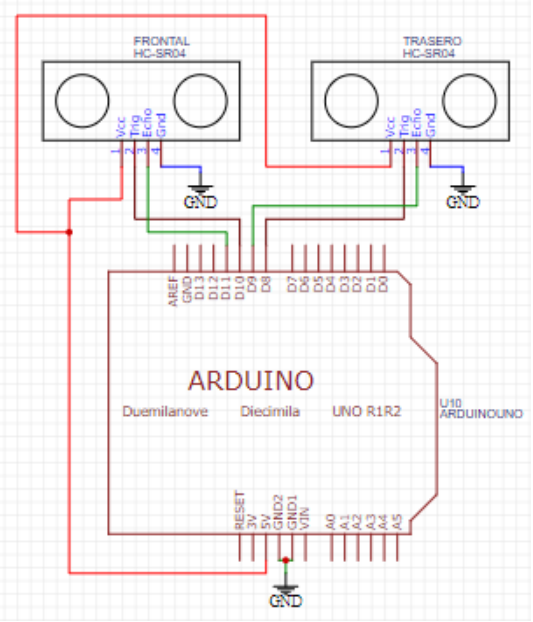


Figura 4.12. Esquema de conexión de los sensores ultrasónicos

En este robot se hace uso de dos sensores ultrasónicos para detectar obstáculos al frente y atrás.

Algoritmo para medir distancias con el HC-SR04

Para poder medir distancias con este sensor haciendo uso de Arduino, únicamente se necesita enviar señales de activación al pin conectado al TRIGGER del sensor durante unos micro segundos, enseguida se debe capturar la señal rebotada mediante el pin conectado a ECHO y medir el tiempo en que tardó en realizar ambas tareas. Por último emplear la ecuación (1), para calcular la distancia.

Para facilitar más el cálculo de la distancia y obtener directamente una medida en cm, solo será necesario utilizar una conversión de unidades de la velocidad del sonido en el aire, pasando de m/s a cm/us como indica la ecuación (2).

$$343\frac{m}{s} * 100\frac{cm}{m} * 1\frac{s}{1{,}000{,}000\mu s} = \frac{1}{29.2}\frac{cm}{\mu s}$$

El algoritmo se muestra a continuación:

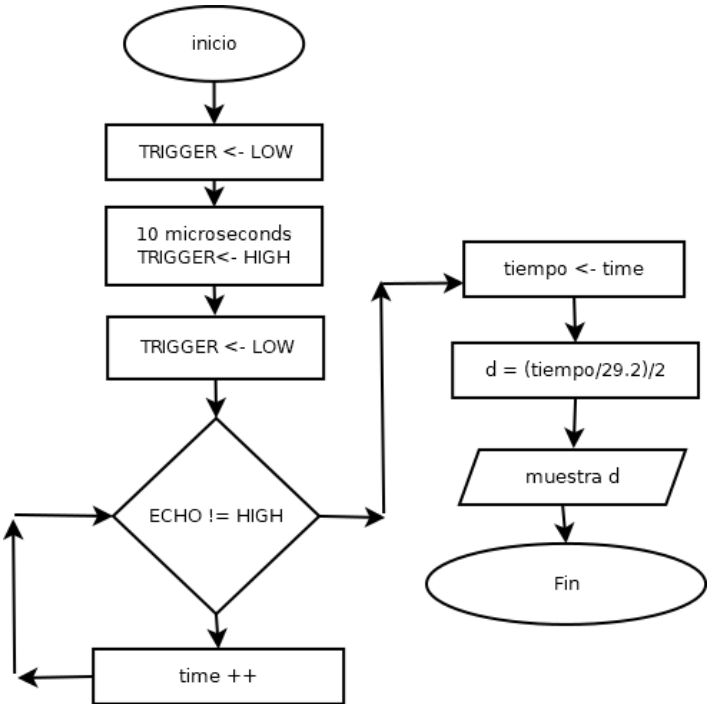


Figura 4.13. Diagrama de flujo para medición de distancias con el sensor HC-SR04

En el diagrama de flujo, el bucle que se ejecuta con la condición ECHO != HIGH únicamente es para representar que se va a contar un tiempo de respuesta, es decir, hasta que el pin conectado a ECHO reciba una señal en alto; luego ese valor medido es asignado a una variable “tiempo”.

Lo anterior se puede resumir en una sola función de Arduino que ya resuelve ese proceso. La función es pulseIn(), la cual recibe dos parámetros: El primero es el número de pin desde el que va a esperar un pulso, y el segundo es el tipo de pulso que se espera, en este caso HIGH.

Código Arduino

Ahora que ya está entendido el algoritmo para calcular distancias, se procede a mostrar el código C/C++ que lo hace funcionar en el microcontrolador.

Definición de macros: con el objetivo de facilitar la identificación de los pines Arduino y su función, tal y como lo hice en la sección anterior con los encoders, defino las Macros que resguardan el número de cada pin utilizado en un identificador para saber de quién se trata.

```
1 //macros para los pines conectados a los sensores ultrasonicos
2 #define TRIGGER_F 11
3 #define ECHO_F 10
4 #define TRIGGER_B 9
5 #define ECHO_B 8
```

La función sensorSonar(): Esta función va a trabajar en ambos sensores; solo recibirá dos parámetros los cuales servirán para indicar el número de pin en el que están conectados el TRIGGER y el ECHO.

```
1 float sensor_Sonar(int Trigger, int Echo)
```

Activación del TRIGGER: Lo primero que se tiene que hacer según el diagrama de flujo es activar el TRIGGER para que se lance la señal ultrasónica; para eso primero se manda una señal LOW durante unos cuantos microsegundos para asegurar un “tiro limpio”, enseguida se manda una señal HIGH durante 10 microsegundos, tiempo suficiente para que se genere la señal ultrasónica. Por último, se desactiva el TRIGGER con un valor LOW.

```
1 //generacion de la señal ultrasonica
2 digitalWrite(Trigger, LOW);
3 delayMicroseconds(4);
4 digitalWrite(Trigger, HIGH);
5 delayMicroseconds(10);
6 digitalWrite(Trigger, LOW);
```

Captura del tiempo y cálculo de distancias: Cuando se haya desactivado el TRIGGER, la siguiente instrucción sólo es extraer el tiempo que tardó en recibir una señal en alto el pin ECHO. Como ya mencione, se usa la función pulseIn() de Arduino. Al final ya solo quedaría calcular la distancia en cm utilizando las ecuaciones (1) y (2).

```
1 tiempo = (pulseIn(Echo, HIGH)); //tiempo variable tipo long
2 distance = float((tiempo / 29.2) / 2); //distance variable tipo float
```

En la figura 4.13 se muestra el resultado de las distancias en cm impresas en la terminal de Arduino.

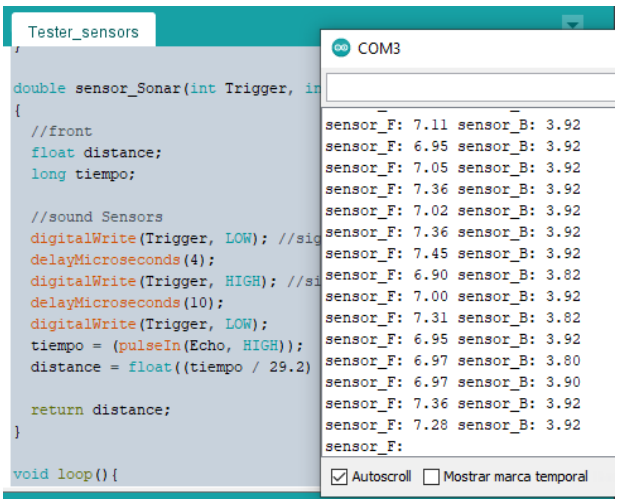


Figura 4.14. Muestra de distancias en cm medidas con el HC-SR04

Para detectar un objeto con este módulo, solo basta con programar un algoritmo que no es más que un comparador de distancias con un umbral que va indicar cual es la distancia mínima que debe de haber entre un objeto y el robot para que se considere como un obstáculo.
Por ejemplo: si programo un comparador utilizando una estructura de decisión (if) en donde condicione que la distancia mínima a la que debe de estar el robot de un objeto cualquiera sea de 10 cm, cuando el sensor detecte 10 cm o menos, entonces se lanzará una señal que le advierta al robot que hay un obstáculo y debe de evitarlo.

El sensor Infrarrojo SHARP GP2Y0A41SK F-86

Este módulo sensor de distancias está compuesto por un LED de luz infrarroja, un PSD (Detector sensitivo de posición) y un circuito para procesamiento de la señal (Figura 4.15). El sensor tiene la capacidad de medir una distancia que va desde 4 hasta los 30 cm, arroja a la salida un voltaje que representa la intensidad de señal infrarroja reflejada y se puede utilizar para calcular la distancia con algún objeto permitiendo utilizar este sensor como un sensor de proximidad.

Además ofrece una fuerte resistencia a las variaciones en el reflejo de las señales infrarrojas sobre diferentes superficies, los cambios en la temperatura ambiente y el tiempo de operación del módulo, ya que adopta un método de triangulación para el manejo de sus señales de operación, por lo que las condiciones anteriores no afectarán tan fácilmente en la medición de distancias (Figura 4.16).

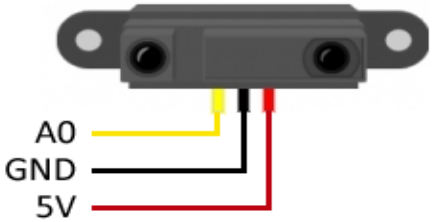


Figura 4.15. Sensor infrarrojo Sharp GP2Y0A02YK0F86 [17]

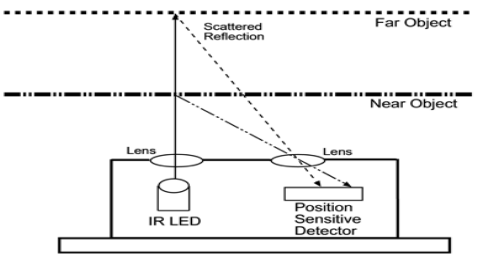


Figura 4.16. Método de triangulación de la señal en sensor SHARP [18]

El funcionamiento de este módulo sensor de distancias, se puede describir como sigue:

1. El sensor tiene tres pines de conexión. 2 que son de alimentación Vcc (-0.3v +7v) y GND, y uno de salida analógica Vo que indica el voltaje en función de las distancias detectadas.
2. Cuando el sensor se encuentra alimentado, se activa un Led infrarrojo que envía pulsos hacia el exterior.
3. La luz infrarroja se refleja en la superficie de algún objeto que se encuentre en el rango de medición, entonces el PSD captura la señal reflejada, y genera un voltaje de salida que envía a través del pin de salida analógica.
4. El voltaje de salida varía en función de la distancia detectada entre el sensor y algún objeto que se encuentre en el medio, como se muestra en el siguiente gráfico (figura 4.16) tomado desde la hoja de datos del sensor.

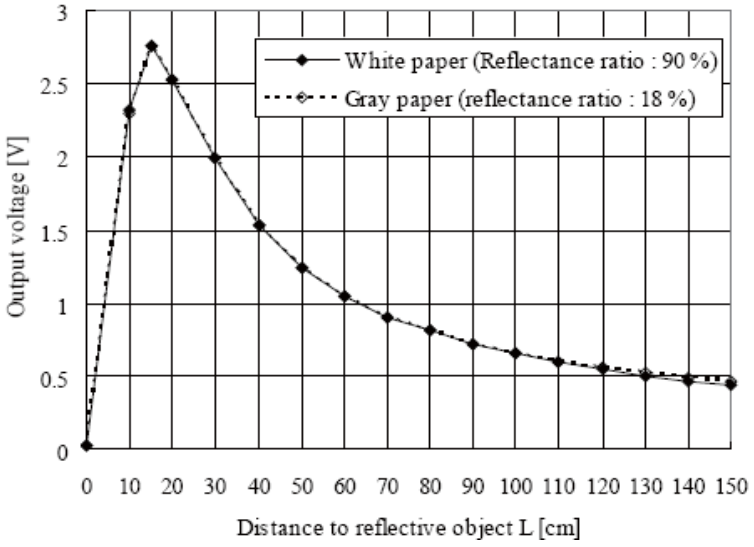


Figura 4.17. Curva de relación voltaje-distancia [19]

Integración del módulo sensor SHARP al robot

Para poderlo utilizar en el robot, primero hay que conectarlo al voltaje de alimentación (+5V) proveído por Arduino. El pin de voltaje de salida debe estar conectado a alguno de los puertos analógicos de la placa Arduino, tal y como se muestra en la figura 4.18.

Debo de aclarar también que en el esquema se muestran dos de estos módulos conectados a Arduino, ya que estos sensores serán utilizados para detectar objetos en los flancos izquierdo y derecho del robot, sirviendo de complemento al sensor ultrasónico frontal.

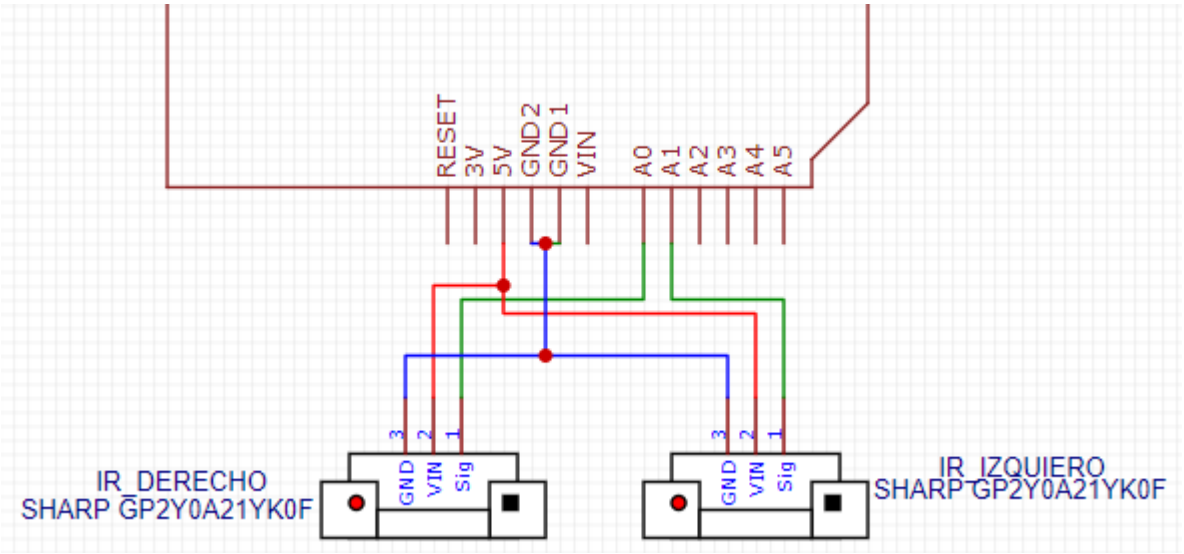


Figura 4.18. Esquema de conexión de los sensores infrarrojos

Algoritmo para medir distancias con el sensor SHARP

El algoritmo propuesto para utilizar este sensor de proximidad en el robot es mucho más simple que el utilizado anteriormente para el HC-SR04. Lo único que se hará, será recolectar los valores arrojados por el sensor en el puerto analógico, los cuales nos indican un valor de voltaje en función de la proximidad que tenga el sensor con un objeto: si el valor es muy grande, el sensor está muy cerca de un obstáculo, si el valor es muy pequeño, entonces está detectando un objeto a lo lejos.

Nuevamente se hace uso del umbral que indicará el valor de la distancia mínima a la que se debe de encontrar el sensor para identificar un obstáculo. Además será conveniente utilizar un método extra para quitar el ruido de la señal, ya que los valores pueden variar de forma muy abrupta y generar picos en la señal de entrada. Para solucionarlo, se hará uso de un filtro de media.

En la figura 4.18 se muestra el flujo del algoritmo anteriormente descrito

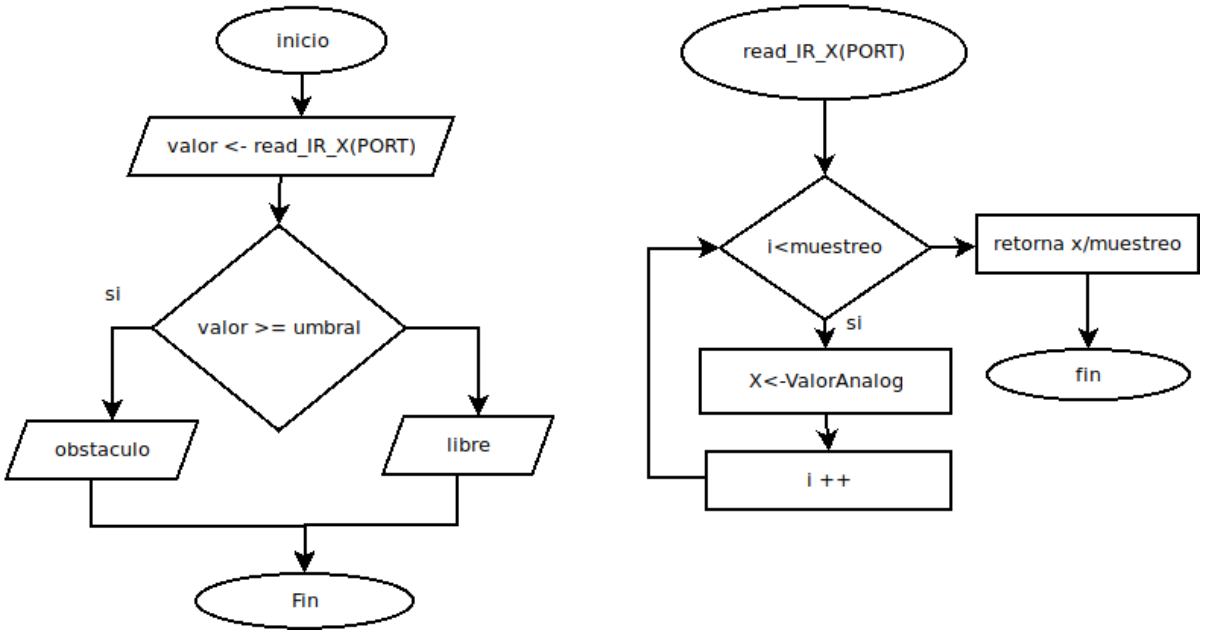


Figura 4.19. Diagrama de flujo para la lectura y filtrado de datos del sensor Infrarrojo

Código Arduino para el sensor SHARP

En el siguiente código se muestra primero una función read_IR_X() donde se va a llevar a cabo la lectura de los datos censados por el módulo SHARP desde el pin indicado por la macro PORT_ANALOGIC. Aquí mismo se encuentra implementado el filtro de media mediante un bucle for, el cual tiene como rango de operación, todos los valores comprendidos en el intervalo [0, muestreo). El parámetro muestreo es un valor entero que indica cuántas lecturas debe de tomar el sensor antes de calcular el promedio de esas lecturas. El promedio calculado, es el valor que retorna esta función.

```
1 int read_IR_X(int PORT_ANALOGIC, int muestreo)
2 {
3  int suma = 0;
4 //filtro de media para eliminar el ruido en la señal
5  for (int i = 0; i < muestreo; i++)
6  {
7   suma = suma + analogRead(PORT_ANALOGIC);
8  }
9  //conversion del voltaje analogico que representa una distancia
10  double adc = (suma / muestreo);
11  return adc;
12 }
```

En el código siguiente solo se muestra el comparador de valores con un umbral para determinar si existe un obstáculo o no. Es importante señalar que el valor el umbral, es un valor que representa una distancia mínima, la forma de obtenerlo puede ser revisando la datasheet del sensor, o de manera empírica como lo he realizado aquí.

El valor de umbral elegido es uno que representa aproximadamente 10 cm.

```
1 //comparador de valores
2 void loop(){
3   Serial.print(read_IR_X(20));
4   If(read_IR_X(20)>= 272){
5   Serial.print("obstaculo detectado");
6   }
7 }
```

Resultados obtenidos.

Figura 4.20. Distancias y detección de objetos con el sensor infrarrojo

Existe otra forma para simplificar el modo en el que se propone el umbral con el que delimitó la distancia mínima a la que debe de estar el robot antes de detectar un objeto. Eso es señalando el umbral con un valor flotante que es ni más ni menos que la medida en cm, y para lograrlo, hay que convertir primero el valor leído por el sensor a su representación en cm. Para ello se está haciendo uso de una ecuación que representa la curva de relación voltaje contra distancia[3]:

$$L = aX^{b} \quad (3)$$

Si se toman muestreos de las lecturas de distancia y valor de voltaje, y se resuelve el sistema de ecuaciones que se puede armar con esos datos, se pueden obtener los valores para a y b los cuales, según la fuente consultada, resultarían como sigue:

$$L = 17579X^{-1.2062} \quad (4)$$

[3] Este método esta explicado en el articulo de Internet que consulte mientras investigaba como calcular distancias con el sensor SHARP, por favor revisar el enlace https://naylampmechatronics.com/blog/55_tutorial-sensor-de-distancia-sharp.html para mas información.

Añadiendo la ecuación (4) en el código, entregaría los siguientes resultados

```
1 float read_IR_X(int PORT_ANALOGIC, int muestreo)
2 {
3  int suma = 0;
4 //filtro de media para eliminar el ruido en la señal
5  for (int i = 0; i < muestreo; i++)
6  {
7   suma = suma + analogRead(PORT_ANALOGIC);
8  }
9 //conversion del voltaje analogico que representa una distancia
10  float adc=suma/muestreo;
11  float distancia_cm = 17569.7 * pow(adc, -1.2062);
11  return distancia;
12 }
```

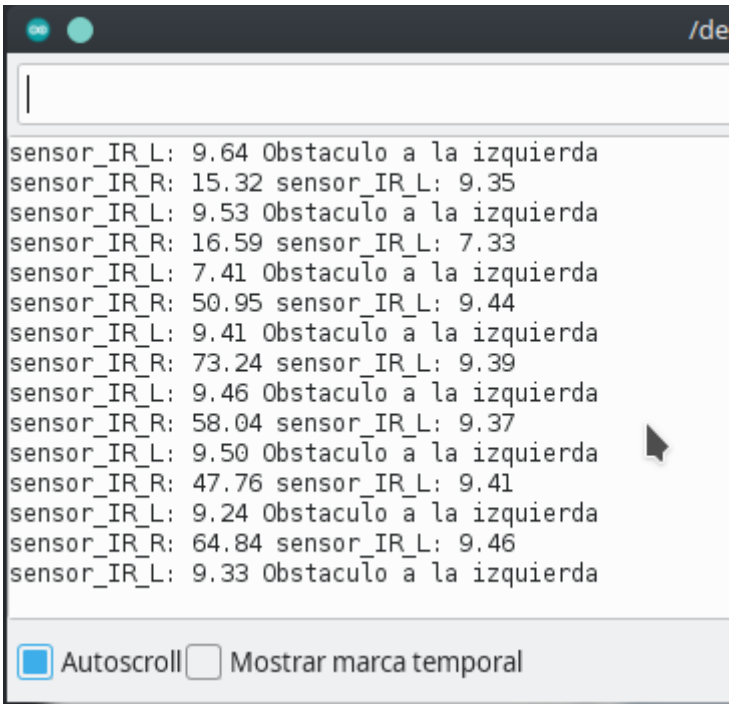


Figura 4.21. Distancias y deteccion de obstaculos
con mejora en el sensor SHARP

De manera adicional, se agregan también un par de sensores extras que servirán como auxiliares para el robot en caso de que sus sensores telemétricos no sean capaces de medir la distancia con algún obstáculo, ya sea porque se encuentran fuera de los rangos de medición de dichos sensores, o porque llegue a presentarse algún fallo a la hora de proporcionar las señales que requieren para su operación. Estos sensores serán un par de micro interruptores de contacto, conocidos también como bumpers o Switch Limit.

Sensor de contacto tipo Bumper

El micro-interruptor bumper se compone de un conmutador de dos posiciones controladas por un muelle que permite activarlo o retornarlo a su posición de reposo, además de utilizar una palanca de aluminio como activador. A continuación muestro un ejemplar:

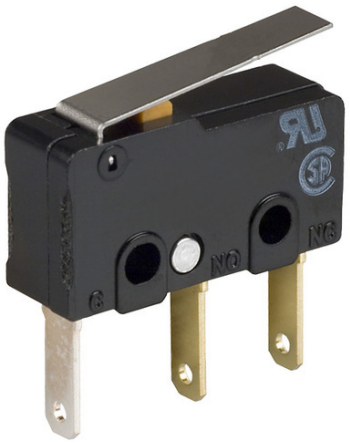


Figura 4.22. Switch bumper

El funcionamiento de dicho bumper se resume de una forma muy simple:
El bumper tiene 3 patillas (a veces 2) para conectarse al circuito.

- Común
- Reposo
- Activado

Si el Bumper se encuentra en reposo, es decir, cuando no hay algún elemento que ejerza presión sobre la palanca de activación, La corriente circulará desde el pin de entrada (Común) hacia el pin de salida del reposo

Si algún elemento ejerce presión sobre la palanca de activación, la corriente se desviara y pasará desde el pin de entrada (Común) hacia el pin de salida de activación

Conexión del Switch-Bumper al robot

Para poder utilizar este sensor en el robot, se debe de conectar ambos sensores al voltaje de alimentación (5V) y utilizar el pin de activación para indicar cuando se haya detectado una colisión con algún objeto, como se muestra en el siguiente esquema.

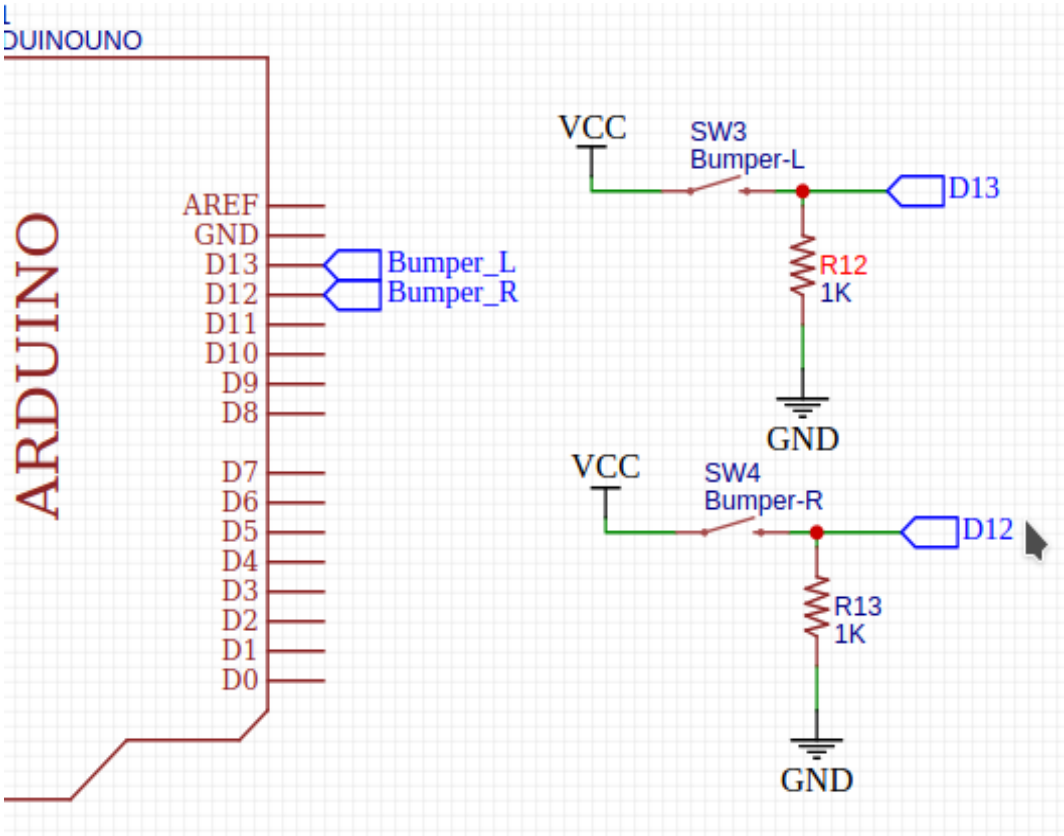


Figura 4.23. Esquema de conexión de los switch-bumpers

En vista de que el bumper solo proporcionará 5v o 0v a la salida, entonces se tiene ya una señal digital que se puede enviar directamente al microcontrolador, quien únicamente va a comprobar qué valor se recibió y desde qué bumper para saber en qué lado (izquierdo o derecho) está el obstáculo e iniciar el algoritmo de evasión.

Código para capturar la señal de activación del Switch-Bumper

La siguiente línea es un ejemplo sobre cómo se debe de tomar la lectura de cualquier bumper X (izquierdo o derecho) solo tomando el valor que retorna la función digitalRead() de Arduino, la cual solo necesita que se le indique en su parámetro el número del puerto desde el que se requiere la lectura; ese puerto es al que esté conectado el Bumper X (izquierdo o derecho).

```
1 Int bumper_X = digitalRead(PORT_CONTACT) //lectura del valor binario en el puerto
```

En este caso, como se espera una señal de activación para saber si el sensor ha detectado una colisión, la variable entera “bumper_X” debe de contener el valor de 1. Si la lectura arroja 0, significa que no hay colisión.

4.2.3 Sensores para fuentes de luz

Para lograr la identificación de la meta a la que el robot se debe dirigir, se decidió utilizar unos sensores demasiado simples pero muy útiles para este fin; se trata de un arreglo de fotorresistencia (Figura 4.24. LDR), que estarán acomodadas de modo que puedan “ver” en diferentes direcciones y que aquella que detecte la mayor incidencia de luz, sea quien le indique al robot la dirección de dicha fuente y el robot se dirija hacia ella.

Figura 4.24 Celda de fotorresistencia y sus símbolos eléctricos [20]

A continuación describimos el funcionamiento de dichas Fotorresistencias.

Una Fotorresistencia (o LDR por sus siglas en inglés, Light Dependent Resistor) es un elemento eléctrico que hace uso del efecto fotoeléctrico para ofrecer resistencia en función de la cantidad de luz que entre en contacto con este elemento. Así un sensor LDR puede generar una Resistencia muy alta (Arriba de 1 M-Ohm) cuando se encuentra en la oscuridad o una resistencia muy baja (menor a los 100-50 Ohm) cuando está en contacto con una luz muy brillante.

Para utilizarlo en el robot, se ocupará un total de 4 LDRs distribuidos en 4 direcciones diferentes: al frente, atrás, izquierda y derecha respectivamente. Cada elemento estará conectado a un divisor de voltaje tomando como salida analógica, el voltaje proporcionado por el fotoresistor en cuestión, como se muestra en la figura 4.24. Este arreglo de foto-resistencias están acomodadas en una torre de en la parte superior del robot, esto con el fin de que pueda censar fácilmente la cantidad de luz en el medio y determinar la dirección de donde provenga la mayor incidencia luminosa.

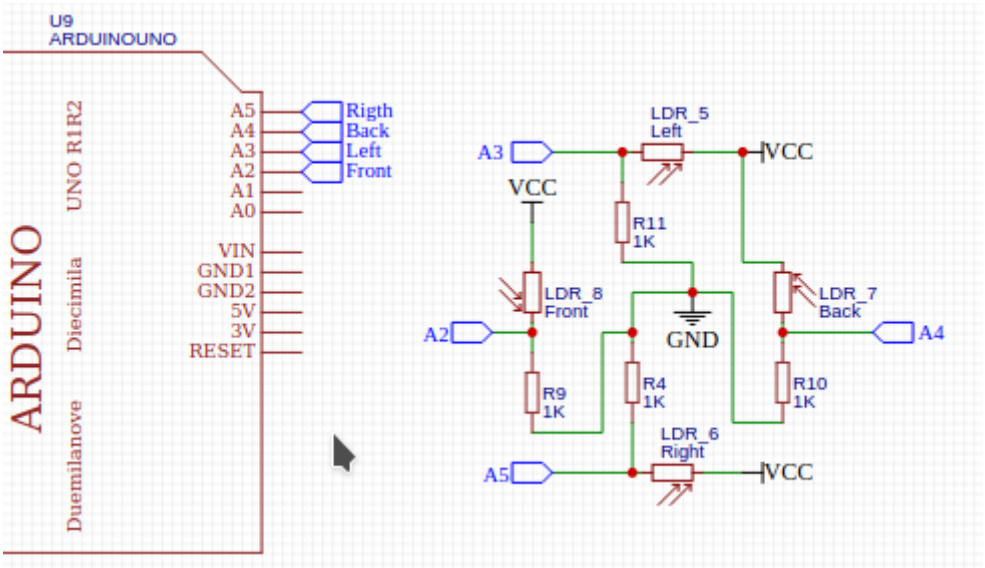


Figura 4.25. Conexión de sensores LDR

La salida de cada divisor de voltaje está conectada a un pin del puerto analógico del microcontrolador. Cuando alguno de los LDR detecte la presencia de luz, el voltaje en el divisor será superior y entonces el microcontrolador recibirá una señal que indique que sensor y en qué dirección se detectó la mayor incidencia de luz y en consecuencia, el robot deberá de ejecutar un algoritmo que le permita girar y avanzar hacia la dirección señalada..

Algoritmo para detectar incidencias de luz

El algoritmo que resuelve la dirección desde donde proviene la mayor incidencia de luz es muy sencillo; solo basta con crear un comparador para los valores de las señales analógicas recibidas desde cada LDR. Aquel que sea el valor más grande, lanzará una bandera que indicará la dirección detectada. Posteriormente existirá una función que recibirá esa bandera para activar los actuadores (motores) del robot y hacer que gire y se dirija hacia la dirección señalada.

Las banderas generadas por el comparador solamente son valores enteros. Ese valor entero representará cada dirección detectada como sigue.

Valor entero	Bandera (identificador)	Dirección detectada
1	LIGHT_FRONT	gira 0°
2	LIGHT_RIGHT	90° a la derecha
3	LIGHT_LEFT	90° a la izquierda
4	LIGHT_BACK	Atrás (180° izq o der)

Tabla 4.1. Banderas Indicadoras de direcciones

El algoritmo del comparador se resume en el siguiente diagrama de flujo

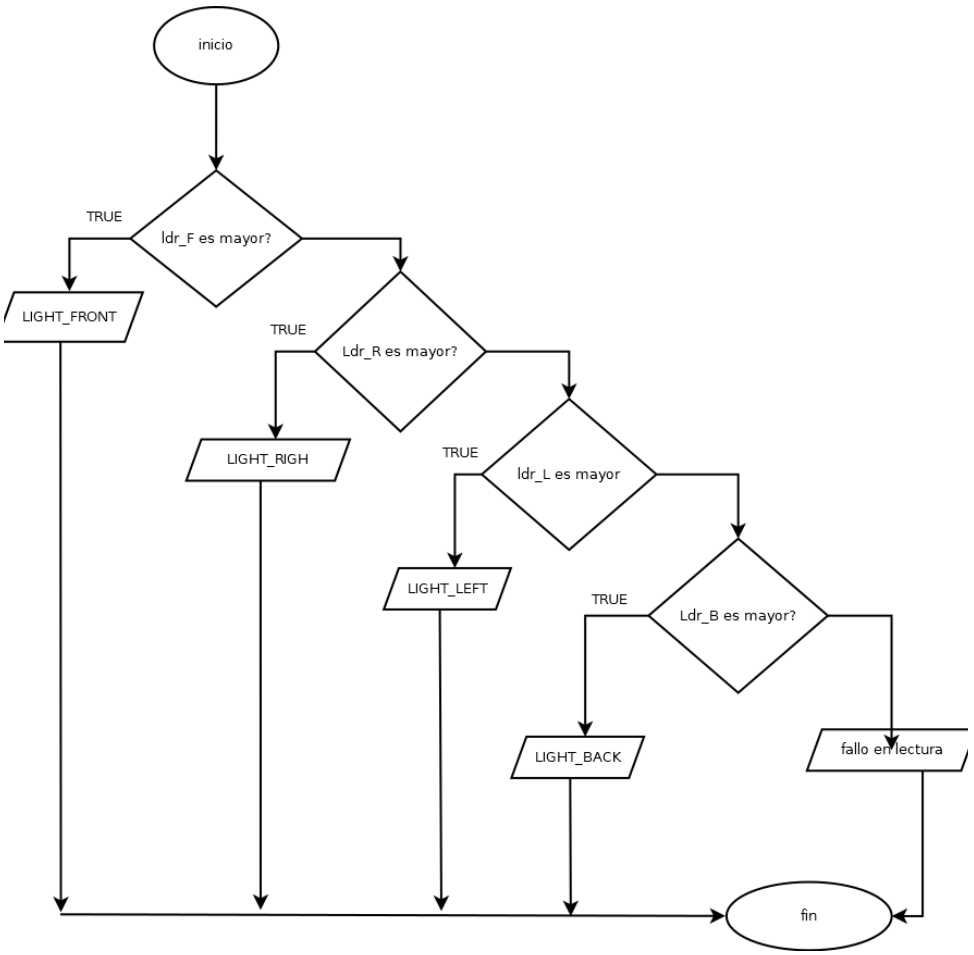


Figura 4.26. Algoritmo para la detección de las incidencias de luz

En el diagrama se encuentran representados en cada bloque de flujo condicional, la evaluación para saber en qué dirección se encuentra la mayor incidencia de luz; y si se evalúa algún caso como verdadero, entonces arroja una salida que es la que devolverá la función. En caso de que no se detecte incidencia de luz en alguna dirección, se arroja una bandera para indicar un fallo en la lectura.

Código Arduino para detectar incidencias de luz.

Macros para indicar direcciones: El siguiente bloque de código, solamente declara constantes simbólicas utilizando macros, las cuales serán las banderas que utilizara el comparador para determinar la dirección de donde proviene la mayor incidencia de luz.

```
1 #define LIGHT_FRONT 1
3 #define LIGHT_RIGHT 2
5 #define LIGHT_LEFT 3
7 #define LIGHT_BACK 4
```

Función detectar las incidencias de luz: El siguiente código muestra la implementación del comparador de valores entregado por el LDR de cada dirección. La función no recibe parámetros, solo retorna el valor entero de la bandera activada.

```
1 int get_direction_light(){}
```

Variables para los datos recolectados: aquí declaro variables locales que van a recibir los datos recolectados desde cada LDR. Las variables son del tipo doble, y reciben el valor entregado por la función Arduino analogRead(), la cual solo recibe como parámetro el número de pin del puerto analógico desde el que va a tomar la lectura.

```
1 double  ldr_Front = analogRead(A2);
2  double ldr_Rigth = analogRead(A3);
3  double ldr_Back = analogRead(A4);
4  double ldr_Left = analogRead(A5);
```

Comparador de valores: Este comparador está implementado con la estructura de selección if-else, en cada caso evaluado, retorna una bandera que indica la dirección donde fue encontrada la mayor incidencia de luz para que posteriormente otra función encargada de activar los motores, indique el ángulo de giro y el avance del robot.

```
1 if ((ldr_Front > ldr_Rigth) && (ldr_Front > ldr_Left) && (ldr_Front > ldr_Back))
2    return LIGHT_FRONT;
3  else if ((ldr_Rigth > ldr_Front) && (ldr_Rigth > ldr_Left) && (ldr_Rigth > ldr_Back))
4   return LIGHT_RIGHT;
5  else if ((ldr_Back > ldr_Front) && (ldr_Back > ldr_Rigth) && (ldr_Back > ldr_Left))
6   return LIGHT_BACK;
7  else if ((ldr_Left > ldr_Front) && (ldr_Left > ldr_Rigth) && (ldr_Left > ldr_Back))
8   return LIGHT_LEFT;
9  else
10   return -1; //Error en la lectura;
```

Muestro a continuación los resultados obtenidos.

Figura 4.27. Aplicación de una alta incidencia de luz
en el costado derecho del robot.

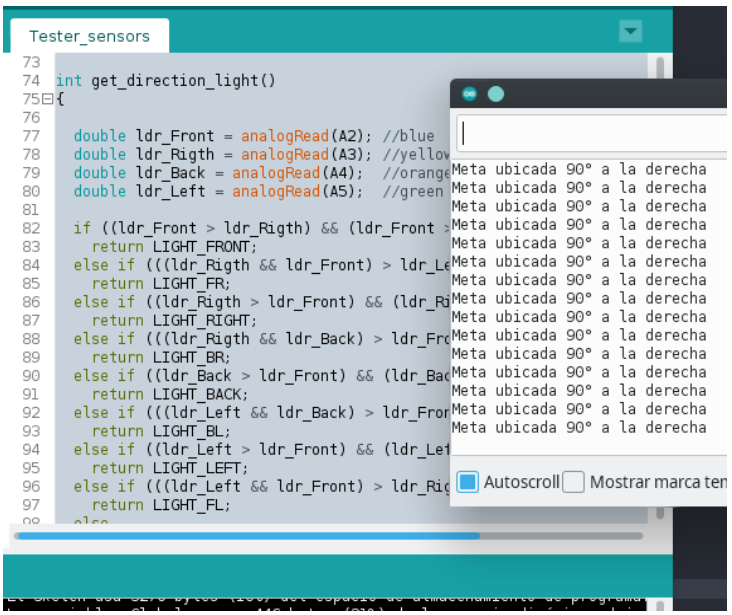


Figura 4.28. Resultados de la detección de incidencias de luz

4.3. Alimentación del Robot

4.3.1. Baterías LI-PO

Uno de los elementos más importantes que necesitará el robot será la fuente de alimentación; pues ésta deberá de proveer el voltaje y corriente necesarios para que todos sus elementos eléctricos (Motores, Microcontroladores, sensores, Computadoras, circuitos de control) funcionen sin problemas.

Para ello se ha elegido una batería recargable de doble celda tipo LI-PO (Polímero de Litio) con un voltaje nominal de 7.4V y una capacidad de 2200 mAh (figura 4.29), lo que resulta ideal para abastecer (principalmente) con energía suficiente a los motores, quienes requieren de un voltaje ideal de 6V para ofrecer un mejor funcionamiento y que además llegan a consumir hasta un máximo de 1600mA; también para abastecer al microcontrolador y a la mini-computadora, los cuales usan un voltaje de 5V y en el caso de la mini-computadora, se requiere de una corriente de 2.1 A para arrancar.

Figura 4.29. Bateria LiPo de 7.4V a 2200 mAh [21]

Para utilizarlo en el robot únicamente se hará uso del conector tipo T de la batería, ya que este es el conector de descarga. El otro conector se va a dejar a la vista porque es el

conector de carga, ya que en caso de que el voltaje de la batería llegue alrededor de los 5V, se necesitará recargar, de lo contrario se podría provocar daños irreversibles en la batería.

También se añadirán un par de circuitos que permitan primero interrumpir el suministro de energía, o sea, para poder encender o apagar al robot y un pequeño módulo que advertirá cuando el nivel de carga de la batería sea bajo; de esta forma sabremos cuando debemos de recargar la batería.

El módulo utilizado es una alarma de bajo voltaje para baterías LiPo de 2s y 3s (2 y 3 celdas), como se muestra en la figura 4.29.

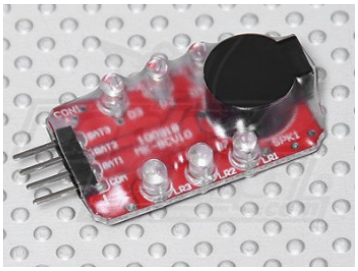


Figura 4.30. Alarma de bajo voltaje para baterías de 2s y 3s [22]

Capítulo 5: Diseño y Montaje del Robot

5.1. Diseño del Robot

Antes de iniciar con el ensamblaje del robot, va a ser necesario tener muy claro la forma en la que se van a montar los componentes para que el resultado sea satisfactorio y permita que el robot logre un buen funcionamiento.

Para lograr lo anterior he decidido utilizar un chasis comercial para robot diferencial debido a que el robot que se va a construir es precisamente un robot de configuración diferencial, que como ya se describió cuando se habló de la morfología del robot en el capítulo 2, el robot únicamente utiliza dos motores con ruedas, izquierdo y derecho, para realizar sus desplazamientos y giros.

La ventaja de adquirir un chasis comercial es que resultará más fácil armar al robot, ya que está diseñado para proporcionar todos los espacios necesarios para montar los elementos que hacen funcionar al robot, pudiendo evitar así varios problemas que interfieran con el buen desempeño de las actividades, por ejemplo, la navegación; si no están bien montadas las ruedas, pueden existir desniveles o que no haya simetría en ambas ruedas, por lo que se puede llegar a complicar demasiado la ejecución de dicha tarea. Además, un chasis ya fabricado resulta ser bastante económico y se ajusta muy bien para los fines de este proyecto.

Se describe a continuación la distribución que se le da a los componentes que va a llevar el robot.

Base: Consta de una placa de MDF de corte circular donde se ubican los orificios izquierdo y derecho para fijar los micro reductores ya con las ruedas y encoders instalados. Además existen un par de orificios más, al frente y atrás, donde se fijarán las “Ruedas locas”, que son únicamente elementos de apoyo para mantener el equilibrio del chasis y permitir el desplazamiento del robot.

Aprovechando el espacio proporcionado por esta primera placa, se hará uso de una protoboard pequeña para colocar el circuito del Puente H que controla la activación del motor así como su dirección de giro; y que se comunica además con el microcontrolador para recibir las señales que permiten lo anterior.

Sensores para detección de obstáculos: Sobre esta misma base, van instalados también los sensores ultrasónicos e infrarrojos, así como los sensores de contacto, todos distribuidos de la siguiente manera:

- Frontal: sensor ultrasónico.
- Izquierda frontal Alta: Sensor Infrarrojo
- Izquierda Frontal Baja: Sensor de contacto
- Derecha Frontal Alta: Sensor Infrarrojo
- Derecha Frontal Baja: sensor de contacto
- Trasera: Sensor ultrasónico.

La distribución sugerida aquí se debe a que se consideró conveniente para poder aprovechar mejor las señales emitidas por los sensores que medirán la proximidad de los obstáculos y de esta forma asegurar de que podrán detectarlos adecuadamente.

También, y como se mencionó en las secciones anteriores, se añaden dos sensores extras que están casi al nivel de la pista donde corre el robot, ya que pueden existir obstáculos que se encuentren fuera del campo de "visión" del robot, o por alguna otra razón, no puedan ser detectados y provoquen un choque. Cuando esto suceda, estos sensores detectarán dicha colisión y le indicarán al microcontrolador, la dirección del choque (Izquierda, Derecha o frontal si se activan ambos).

Centro de Control: En una segunda placa de MDF (o acrílico) unida la primera mediante postes, estarán colocados el micro-controlador Arduino y el circuito (montado en otra Protoboard) donde se comunicaran los sensores con el microcontrolador, así como la alimentación correspondiente de los elementos, el interruptor general del robot y la alarma que indicará el nivel de energía en el suministro eléctrico (Batería).

La placa Raspberry PI se colocará en una tercera placa de MDF, debido al tamaño que posee ya que es más grande que la de Arduino. De esta forma se podrían distribuir bien estos elementos del robot para que no queden muy amontonados y sea sencilla su manipulación.

Torre con sensores de luz: Tal y como lo indica su nombre, esta será una torre que sobresaldrá en la parte superior del robot y en la cual, estarán conectadas las fotorresistencias que detectan la luz proveniente de la meta, así como la dirección donde se encuentra.

Batería: Por último y no menos importante, será el sitio donde se deberá de colocar la batería, que estará conectada al circuito principal del robot para alimentar a todos los elementos y además estará conectada a la alarma que indicará cuando su nivel de carga sea muy bajo y requiere recargarse.

En el caso de este robot, se ancló a la contracara de la segunda placa donde están montados todos los elementos de control. En vista de que la batería es uno de los objetos más grandes y pesados que va a llevar el robot, este espacio fue el más adecuado para poder acomodar correctamente este elemento y evitar conflictos con los demás componentes del robot.

5.2. Montando los componentes

Teniendo ya claro el diseño del robot y los componentes que se van a utilizar, el siguiente paso, corresponde a simplemente ensamblarlo para poder continuar entonces con el desarrollo de sus demás funcionalidades.

A continuación se muestran las imágenes correspondientes al montaje de todos los componentes que se fueron mencionando a lo largo de este capítulo, ya sobre el chasis elegido.

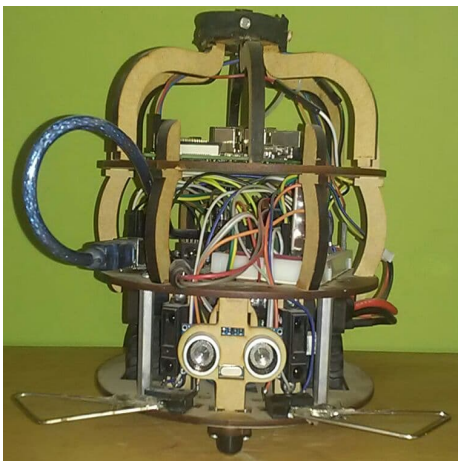


fig. 5.1 Cara Frontal King Mechanic

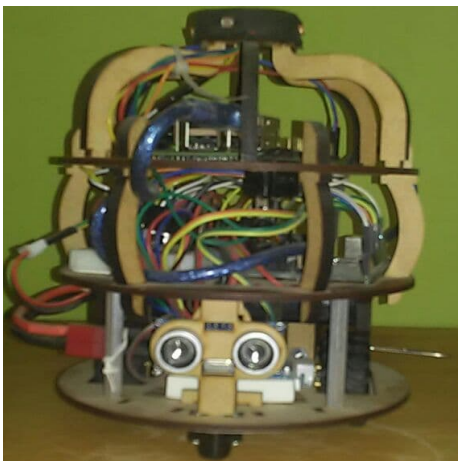


fig. 5.2 Cara Posterior King Mechanic

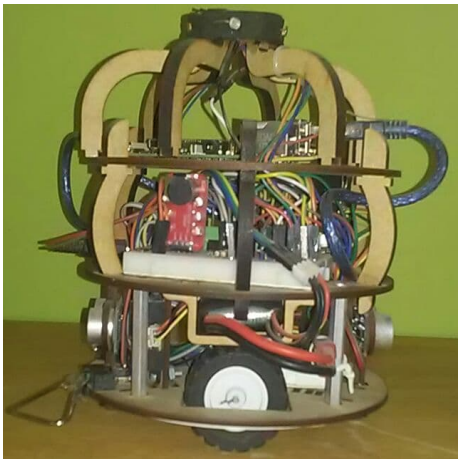


fig. 5.3 Lateral izquierdo King Mechanic

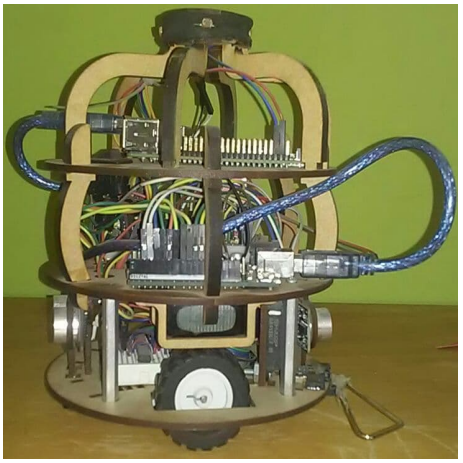


fig. 5.4 Lateral derecho King Mechanic

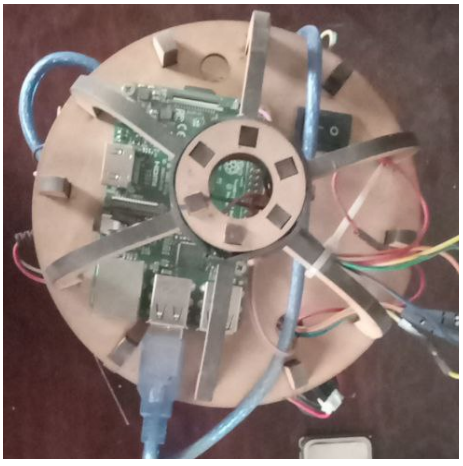


fig. 5.5 aérea King Mechanic

Capítulo 6. Cinemática del Robot Móvil

A partir de aquí se comienza a tratar de lleno el desarrollo del segundo objetivo particular que tendrá el robot, que dicta:

- El robot posee la capacidad de medir su estado interno para ajustar sus valores de modo que esto le permite mejorar el desempeño de sus funciones de navegación en el medio.

Para lograr este objetivo será necesario realizar el estudio de dos temas importantes que permitan entender el modo en que se debe ejecutar correctamente la navegación del robot. Dichos temas son, la cinemática y los sistemas de control.

En este capítulo se cubrirá el desarrollo del análisis cinemático de los movimientos del robot, así como el diseño e implementación de los algoritmos que permitirán poner en práctica todo el razonamiento generado. Además este capitulo servirá también como un importante paso previo para entender la implementación de los sistemas de control para la navegación.

6.1. Análisis cinemático del robot

Para comenzar a analizar la cinemática del robot móvil se debe de tener en cuenta las siguientes condiciones:

- Se trata de un robot de configuración diferencial, así que posee solamente dos ruedas principales que le generan el impulso para desplazarse.
- Las ruedas se encuentran unidas en el mismo eje y están separadas por una distancia rígida, es decir, constante.
- Las ruedas pueden girar en sentidos diferentes o iguales, lo cual dota al robot de la capacidad de desplazarse sobre una trayectoria lineal o curva, o girar sobre su propio eje.
- Cada una de las ruedas de impulso posee su propia velocidad lineal o angular.
- Existen dos ruedas extras ubicadas a los extremos de un eje perpendicular. Estas son ruedas giratorias, es decir, giran libremente hacia cualquier dirección y solo son de apoyo para que el robot mantenga el equilibrio.
- Por fines prácticos, se analizará el movimiento del robot bajo condiciones ideales, o sea, en las que el robot es un cuerpo rígido que posee un par de ruedas que le dan impulso y se desplaza sobre un plano bidimensional.

Iniciemos el análisis con la figura 4.1, que muestra un diagrama que representa una rueda del robot y en él se señalan sus velocidades y desplazamientos.

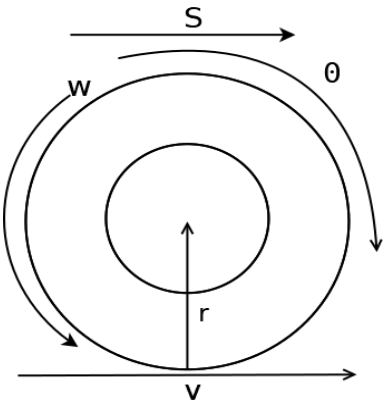


Figura 6.1 Rueda del robot

La rueda del robot es representada como una circunferencia, que posee un radio r. Cuando la rueda se encuentra en movimiento, esta genera un desplazamiento angular representado porθy un desplazamiento lineal representado por S. A su vez, la rueda posee dos velocidades: una velocidad tangencial o lineal, y una velocidad angular representadas por v yω respectivamente.

Con los datos anteriores, se pueden empezar a generar las primeras ecuaciones para el robot, tomando en cuenta que estas aplican tanto para la rueda izquierda (L) como para la rueda derecha (R).

Desplazamiento.

El desplazamiento para una rueda cualquiera $S_x$es igual al perímetro de la circunferencia de la rueda que es directamente proporcional a su posición angular o ángulo de giro (θ_x) dado, quedando:

$$S_x = (2\pi r)\left(\theta_x\right)\ldots (1)$$

Velocidad lineal y angular de la rueda.

Para obtener la velocidad lineal de la rueda hay que derivar el desplazamiento lineal descrito en la ecuación 1, con respecto al tiempo, quedando:

$$v_x = \frac{d}{dt}\left(2\pi r\theta_x\right)$$

$$v_x = 2\pi r d\frac{\theta_x}{dt}\ldots (2)$$

En la ecuación 2, $d\frac{\theta_x}{dt}$representa un cambio de posición angular respecto al tiempo, por lo que esta expresión representa directamente a la velocidad angular de la rueda (ω_x), pudiendo simplificar la expresión como sigue:

$$v_x = 2\pi r\omega_x\ldots (3)$$

Hasta este momento se tiene definido entonces que el desplazamiento lineal de cualquiera de las ruedas del robot es directamente proporcional al producto de su perímetro por su posición angular; y de igual forma la velocidad lineal de la rueda es directamente proporcional al producto de su perímetro por su velocidad angular.
Finalmente se tiene para cada rueda, las siguientes expresiones:

$S_L = (2\pi r)\left(\theta_L\right)\ldots$ (4)

$S_R = (2\pi r)\left(\theta_R\right)\ldots$ (5)

$v_L = 2\pi r\omega_L\ldots$ (6)

$v_R = 2\pi r\omega_R\ldots$ (7)

Desplazamiento y velocidad lineal del robot

En la figura 6.2, muestro un diagrama de cuerpo rígido del robot donde se muestra la forma en que actúan los desplazamientos y velocidades de las ruedas en el desplazamiento y velocidad del robot.

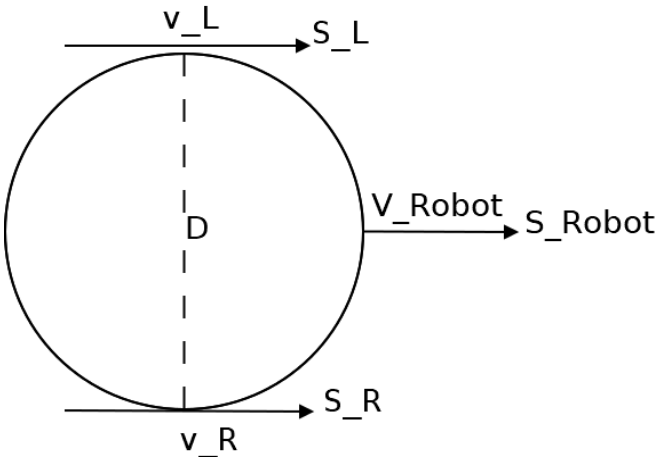


Figura 6.2. Diagrama de cuerpo rígido del robot

Según lo mostrado en el diagrama, la velocidad y desplazamiento del robot se pueden calcular fácilmente como un promedio de las velocidades y desplazamientos de cada una de sus ruedas, obteniendo entonces

$$V_{robot} = \frac{\left(v_R + V_L\right)}{2}\ldots (8)$$

$$S_{robot} = \frac{\left(s_R + s_L\right)}{2}\ldots (9)$$

Descripción general de la cinemática del robot móvil

Utilizando las ecuaciones anteriores que permiten calcular las velocidades y desplazamiento de las ruedas del robot y del robot mismo, se puede plantear un modelo general que describa todos los movimientos y velocidades de un robot móvil.

En el diagrama 6.3, se muestra un robot que avanza y gira respecto a un punto de referencia ubicado en un plano.

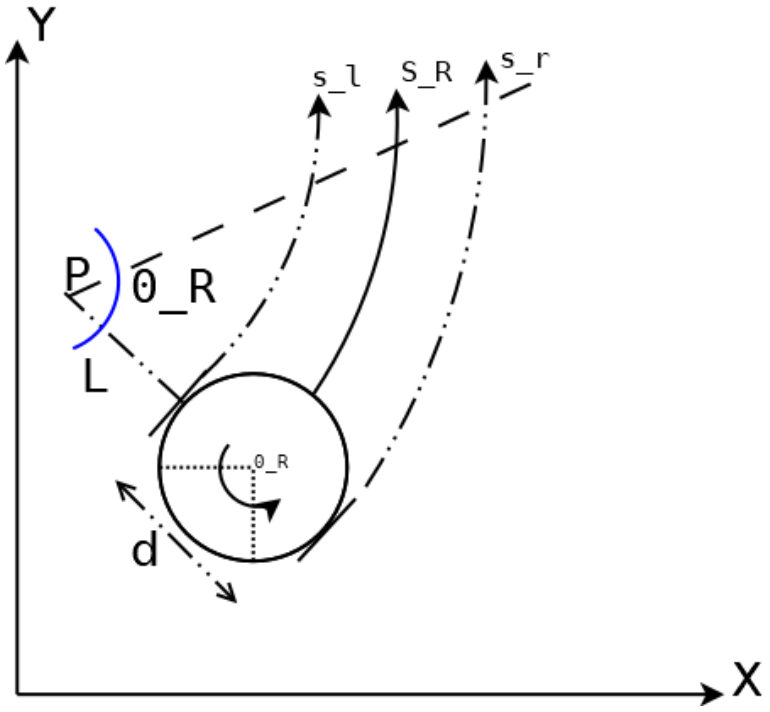


Figura 6.3. Cinemática general del robot móvil

Del diagrama anterior se tienen los siguientes datos:

- S_R → Desplazamiento angular del robot S_{Robot} sobre una trayectoria curvilínea.
- P → Punto de referencia respecto al cual gira el robot
- L → Longitud entre el punto de referencia respecto al cual gira el robot y el centro del robot.
- S_L → Desplazamiento angular en rueda izquierda S_L sobre una trayectoria curvilínea
- S_R → Desplazamiento angular en la rueda derecha S_R sobre una trayectoria curvilínea
- d → diámetro del robot o distancia entre las ruedas que ejercen la tracción.
- 0_R → Posición Angular del robot θ_{Robot}. Esta Posición angular describe el giro del robot respecto a P o respecto a su propio eje que es perpendicular al plano.

Para obtener las ecuaciones que describen la cinemática general, se toma en cuenta las siguientes relaciones:

- S_{Robot} que representa el desplazamiento angular del robot es L veces proporcional a la posición angular θ_{Robot}.

$$S_{Robot} = \theta_{Robot} * L \ldots (10)$$

- $S_L$que es el desplazamiento angular en la rueda izquierda del robot es $L - \frac{d}{2}$ proporcional a la posición angular θ_{Robot}

$$S_L = \theta_{Robot} * \left(L - \frac{d}{2}\right) \ldots (11)$$

- $S_R$que es el desplazamiento angular en la rueda derecha del robot es $L + \frac{d}{2}$ proporcional a la posición angular θ_{Robot}

$$S_R = \theta_{Robot} * \left(L + \frac{d}{2}\right) \ldots (12)$$

Se despeja L en las ecuaciones 11 y 12:

$$L = \frac{S_L}{\theta_{Robot}} + \frac{d}{2} \ldots (13) \quad L = \frac{S_R}{\theta_{Robot}} - \frac{d}{2} \ldots (14)$$

Igualando y simplificando las ecuaciones 13 y 14:

$$\frac{S_R}{\theta_{Robot}} - \frac{d}{2} = \frac{S_L}{\theta_{Robot}} + \frac{d}{2}$$

$$\frac{S_R}{\theta_{Robot}} - d = \frac{S_L}{\theta_{Robot}}$$

$$S_R - d * \theta_{Robot} = S_L$$

$$\frac{(S_R - S_L)}{d} = \theta_{Robot} \ldots (15)$$

La ecuación 15 describe que la posición angular del robot es un cociente de las diferencias de desplazamientos entre la rueda derecha y la rueda izquierda sobre la distancia que hay entre ellas.
Si se deriva la ecuación 15, se obtiene la velocidad angular del robot, como sigue:

$$d\frac{\theta_{Robot}}{dt} = \frac{\left(\frac{dS_R}{dt} - \frac{dS_L}{dt}\right)}{d}$$

Recordando que la derivada del desplazamiento de la rueda izquierda o derecha respecto al tiempo, es igual a la velocidad lineal en cada una de ellas. Por lo que la expresión quedaría como:

$$\omega_{Robot} = \frac{(v_R - v_L)}{d} \ldots (16)$$

Expresión matricial de las velocidades lineal y angular del robot

Finalmente y para concluir esta sección se representa la forma matricial de las ecuaciones de velocidad lineal y angular del robot en función de las velocidades angulares de sus ruedas, esto con el fin de mostrar la relación que hay entre ellas.

De las ecuaciones 8 y 16 obtenidas anteriormente se tiene

$$V_{robot} = \frac{\left(v_R + v_L\right)}{2} \ldots \text{ velocidad lineal del robot}$$
$$\omega_{Robot} = \frac{\left(v_R - v_L\right)}{d} \ldots \text{ velocidad angular del robot}$$

De las ecuaciones 6 y 7 obtenidas anteriormente se tiene

$$v_L = 2\pi r\omega_L \ldots \text{ Velocidad lineal de la rueda izquierda}$$
$$v_R = 2\pi r\omega_R \ldots \text{ velocidad lineal de la rueda derecha}$$

Sustituyendo las ecuaciones de v_L y v_R en V_{Robot} y ω_{Robot} se tiene

$$V_{robot} = \frac{\left(2\pi r\omega_R + 2\pi r\omega_L\right)}{2}$$
$$\omega_{Robot} = \frac{\left(2\pi r\omega_R - 2\pi r\omega_L\right)}{d}$$

Que representan las ecuaciones de velocidad lineal y velocidad angular del robot en función de las velocidades angulares de ambas ruedas, es decir, que el robot avanza o gira según a la velocidad que giran sus ruedas.

También estas ecuaciones se pueden ver como un sistema de ecuaciones lineales con dos incógnitas, por lo que se pueden expresar de forma matricial como sigue

$$\left[V_{Robot}\ \omega_{Robot}\right] = 2\pi r\left[\frac{1}{2}\ \frac{1}{2}\ \frac{1}{d}\ -\left(\frac{1}{d}\right)\right]\left[\omega_R\ \omega_L\right]$$

Considerando que A representa una matriz de constantes:

$$A = 2\pi r\left[\frac{1}{2}\ \frac{1}{2}\ \frac{1}{d}\ -\left(\frac{1}{d}\right)\right]$$

También se podría escribir la expresión como

$$\left[V_{Robot}\ \omega_{Robot}\right] = A\left[\omega_R\ \omega_L\right]$$

Ahora suponiendo el caso en el que en vez de conocer las velocidades de las ruedas, se conocen las velocidades del robot, y que de hecho será lo más común como se verá en los siguientes capítulos, simplemente se tendría que cambiar la expresión anterior de modo que las velocidades angulares de las ruedas estén en función de las velocidades lineal y angular del robot.

Esto se logra simplemente obteniendo la matriz inversa de A, y despejando a ω_R y ω_L de la expresión, como sigue.

$$-A\left[V_{Robot}\ \omega_{Robot}\right]=\left[\omega_R\ \omega_L\right]$$

quedando:

$$\left[\omega_R\ \omega_L\right]=\frac{1}{(2\pi r)}\left[1\ -\left(\frac{d}{2}\right)1\ \frac{d}{2}\right]\left[V_{Robot}\ \omega_{Robot}\right]$$

La expresión anterior va a ser de mucha utilidad a la hora de implementar la cinemática en el robot, ya que con esto simplemente se le tendría que indicar al robot a qué velocidad debe avanzar o girar para trazar una trayectoria o recorrer una distancia.

6.2. Odometría

Para poder medir los cambios de posición o distancias que ha recorrido el robot y que permitirán posteriormente obtener el cálculo de velocidades, es necesario hacer uso de la Odometría, lo cual consiste simplemente en calcular distancias o posiciones del robot en función de las lecturas del encoder que mide el giro en las ruedas del robot; en este caso el encoder magnético de efecto Hall.

Retomando las ecuaciones 4 y 5 de la sección a anterior, se tiene que la distancias S_L y S_R son una relación de proporcionalidad del perímetro de la rueda con el desplazamiento angular θ de cada una de ellas.

$S_L=(2\pi r)\left(\theta_L\right)\ldots$ Desplazamiento lineal en rueda izquierda
$S_R=(2\pi r)\left(\theta_R\right)\ldots$ Desplazamiento lineal en rueda derecha

De estas ecuaciones, el radio (r) de la rueda se puede conocer leyendo directamente las especificaciones del producto donde normalmente se incluyen sus dimensiones o simplemente midiéndolo con alguna regla. Para el caso del del desplazamiento angular se tendrá que hacer uso del encoder para adherido al motor para medir cuántas veces ha girado la rueda tomando en cuenta la siguiente relación:

$$\theta_x=\frac{(Ticksgenerados)}{(Tickspor revolución)}$$

donde:

θ_x => es el desplazamiento angular para cualquiera de las ruedas, izquierda o derecha.

$Tickgenerados$ => lectura de los pulsos que ha generado el encoder magnético

$Tickspor revolución$= cantidad de pulsos que se necesitan para determinar un giro completo en la rueda.

Quedando entonces la ecuación del desplazamiento lineal de cada rueda como:

$$S_L = (2\pi r)\left(\frac{TicksGenerados_L}{TicksPorRevolucion}\right)$$
$$S_R = (2\pi r)\left(\frac{TicksGenerados_R}{TicksPorRevolucion}\right)$$

La constante TicksPorRevolucion se puede obtener por medio de dos métodos: empírico o analítico.

Método empírico:

Consiste simplemente en conectar el encoder al microcontrolador que utiliza el robot, Arduino, y mediante el contador de pulsos o ticks visto en la sección 4.x del capítulo 4, se realiza un movimiento manual en la rueda y se visualiza en la consola serial de Arduino el número de ticks registrados cuando la rueda completo un giro.

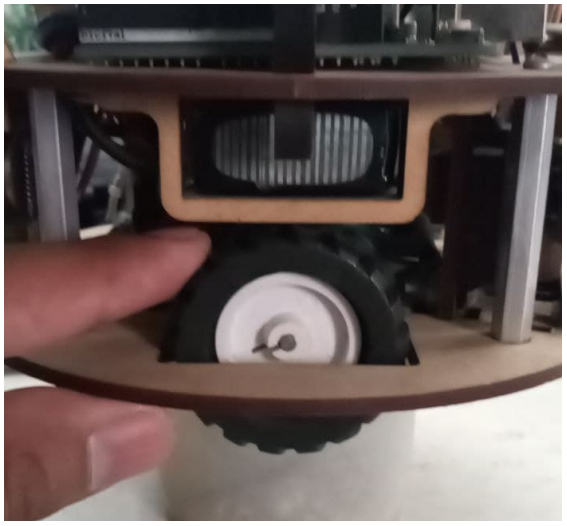


Figura 6.4. Método empírico para determinar la constante *TicksPorRevolucion*

Es importante advertir que este método puede funcionar pero no es muy preciso por lo que se pueden hacer lecturas erróneas que después afectarán directamente en los cálculos y en la ejecución de los movimientos del robot.

Método analítico:

Para aplicar este método hay que conocer los siguientes datos para poder operar:

- El giro en la rueda equivale a un giro en el eje de salida del micro reductor.
- La cantidad de giros que se producen en el eje del motor para poder generar un giro en el eje de salida del micro reductor.
- La cantidad de pulsos que se generan cada vez que el eje del motor completa una vuelta.

Repasando los datos sobre el motor con micro reductor que se está utilizando para este robot, se sabe que la salida de este último tiene una relación de 1:100, lo que significa que el eje del motor realiza 100 giros por cada giro en el eje de salida.

El número de pulsos que se generan por cada giro en el eje del motor lo proporciona el fabricante en la hoja de datos del encoder, en este caso son 12 pulsos.

Con los datos anteriores se puede resolver una revolución en el eje rueda, que como ya se dijo antes, equivale también a una revolución en el eje de salida del micro reductor, genera 1200 pulsos.

Pulsos = conteoEncoder x 100 = 12 x 100 = 1200

Pero hay que tomar en cuenta también estas otras condiciones:

- Se genera un conteo por cada pulso en alto o en bajo. Por lo que en cada ciclo hay 2 conteos.
- Los 1200 pulsos totales en el conteo corresponden a los dos canales A y B.
- Como se dijo en la sección 4.2.1 del capítulo 4, solo se tomará un solo canal de lectura, debido a las limitaciones del hardware. Así que solo serian 600 pulsos (1200/2)
- De esos 600 pulsos se vuelve a tomar solo la mitad (300), debido a que cuando el microcontrolador lee la señal, solo lee la mitad de la señal: ya sea un pulso en alto, o un pulso en bajo.

Quedando entonces:

$$TiksPorRevolucion = (12 * 100) * \left(\frac{1}{2}\right) * \left(\frac{1}{2}\right) = 1200 * \left(\frac{1}{4}\right) = 300$$

Este valor se obtuvo también aplicando el método empírico. Ya que el resultado de las lecturas oscilaba entre 296 y 304, generando un promedio aproximado de 300.

# prueba	Pulsos registrados
1	304
2	298
3	296
4	302
5	299
6	301
	Promedio = 301.16

Tabla 6.1. Muestras de medición de pulsos mediante método empírico.

Capítulo 7. Algoritmos para la cinemática del robot

7.1 Desplazamiento lineal del robot.

El algoritmo propuesto aquí, sirve para lograr que el robot se desplace una cantidad cualquiera de metros en una trayectoria “recta”. La idea es tener un programa en el microcontrolador que reciba como entrada la distancia en metros que debe de recorrer el robot, y como salida se generen las señales que activen a los motores para que las ruedas giren y se detengan exactamente cuando se haya recorrido la distancia señalada.

En la figura 6.5 se representa de forma gráfica el desplazamiento del robot desde un punto A a un punto B con una distancia D_{mts} entre ellos.

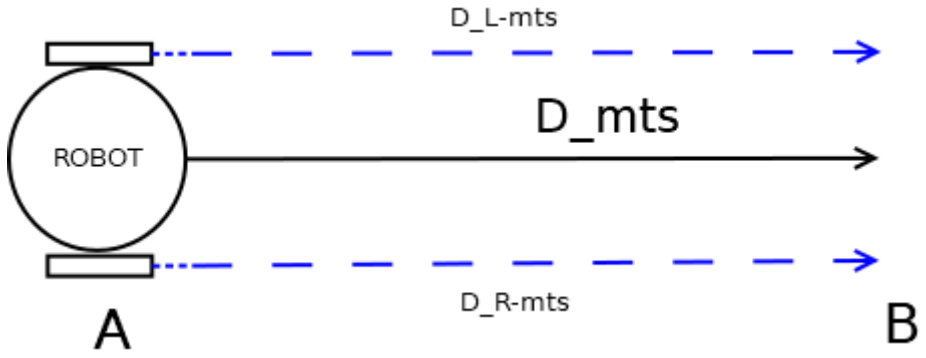


Figura 7.1. Desplazamiento del robot desde un punto a hasta un punto B

En el diagrama se puede observar que las distancias que recorren las ruedas izquierda y derecha ($D_{(L-mts)}$ y $D_{(R-mts)}$ respectivamente) son exactamente iguales a la que recorre el robot, por lo que para resolver el problema planteado al inicio de esta sección, simplemente habría que crear un programa que:

Calcule las distancias recorridas en cada rueda a partir de los pulsos generados por sus encoders y haciendo uso de las ecuaciones:

$$S_L = (2\pi r)\left(\theta_L\right)$$
$$S_R = (2\pi r)\left(\theta_R\right)$$

- Seguidamente calcule la distancia que ha recorrido el robot y la compare en todo momento con la distancia objetivo que seria el umbral, haciendo uso de la ecuación:

$$S_{robot} = \frac{\left(s_R + s_L\right)}{2}$$

- Si la distancia recorrida es igual o superior al umbral, el robot se debe de detener.

El algoritmo anterior funciona, sin embargo, al aplicarlo de esta forma se sacrificaría mucho la eficiencia del funcionamiento del robot ya que el algoritmo obligaría al microcontrolador a resolver varías tareas antes de generar una salida.

Un método mas sencillo con el que logre resolver esto, consiste en dividir en 2 partes el problema.

La primer parte consiste en crear una función que se encargue de calcular una única vez, la cantidad de pulsos que se necesitan contar en el encoder para saber que el robot ha recorrido una distancia S_x.

La idea surge a partir de la ecuación:

$$S_x = (2\pi r)\left(\frac{TicksGenerados_x}{TicksPorRevolucion}\right)$$

donde S_x y $TicksGenerados_x$ representan el desplazamiento y los pulsos en la rueda izquierda o derecha. También hay que tener en cuenta que $S_x = S_{Robot}$, como se muestra en el diagrama 7.1.

Despejando $TicksGenerados_x$ la ecuación queda:

$$TicksGenerados_x = \frac{\left(S_x * TicksPorRevolucion\right)}{(2\pi r)}$$

El Algoritmo se puede apreciar en la figura 7.2

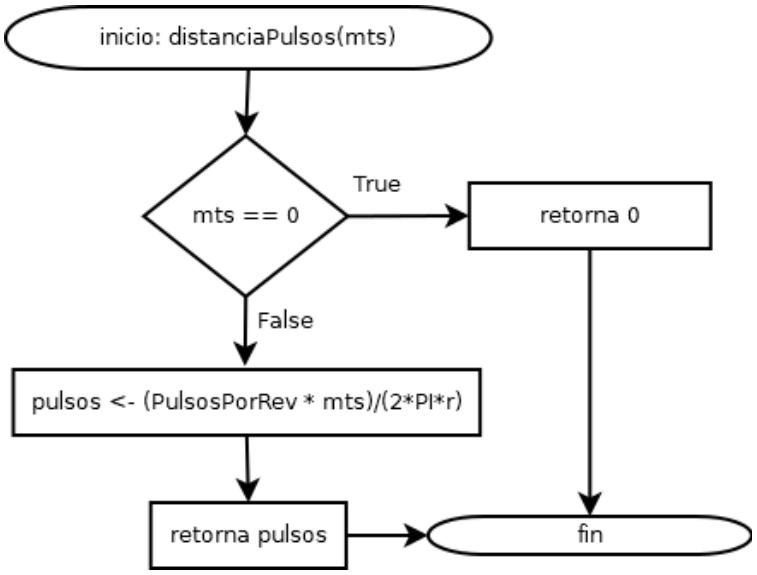


Figura 7.2. Algoritmo para convertir distancias en metros a pulsos.

La segunda parte consiste en ejecutar un bucle que enviará las señales de activación para las ruedas mientras la cantidad de pulsos generados durante el movimiento de cada rueda sea menor a la cantidad de pulsos calculados en la primera parte.

El diagrama 7.3 muestra el algoritmo empleado:

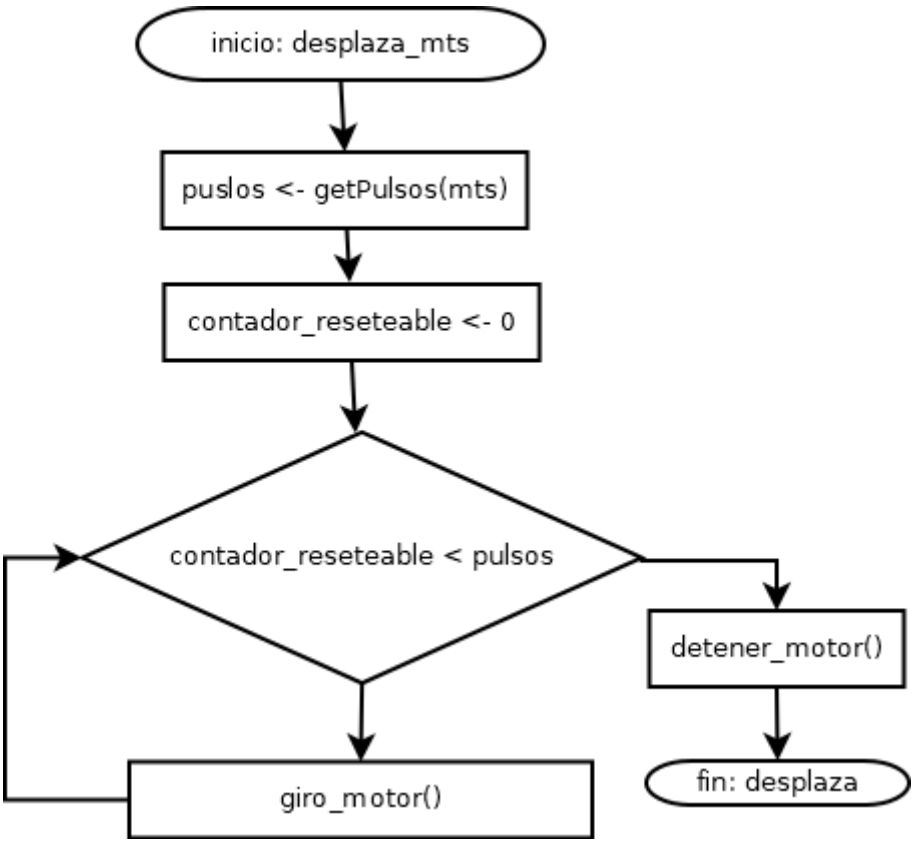


Figura 7.3. Algoritmo para ejecutar el desplazamiento del robot en una distancia X

implementación del algoritmo

A continuación muestro cómo se implementaría el algoritmo expuesto anteriormente utilizando el lenguaje C++ en Arduino.
metersToTicks: Función que se encargará de convertir una distancia medida en metros a un número de pulsos. Se utiliza el algoritmo descrito en la figura 7.2. El código es el siguiente.

```
1 int metersToTicks(double meters)
2 {
3   if (meters == 0)
4     return 0;
5
6   return (int)((PULSOSPORGIRO * meters) / (2 * PI * (RADIORUEDA)/100));
7
8 }
```

Esta función recibe un parámetro double que son los metros que se desean avanzar y retorna un valor entero que es el número de pulsos necesarios para recorrer esa distancia, como se indica en la línea 1.

La condición en la linea 4 retorna un valor por defecto cuando la distancia que se debe recorrer es de 0 metros.

En la línea 6 se aplica la ecuación generada para el algoritmo anterior. El resultado arrojado por esa ecuación es convertida (casteo) a entero, para truncar todos los valores

decimales que se generen en la operación ya que son irrelevantes para el resto del proceso y se retorna.
goForward: Función que enviará las señales de activación a los motores del robot para que se desplace por una distancia determinada por el número de pulsos que recibe como entrada. Se utiliza el algoritmo descrito en la figura 7.3. Su código es el siguiente.

```
1 void goForward(int ticks)
2 {
3   left_count_ticks = 0;
4   right_count_ticks = 0;
5
6   while (left_count_ticks <= ticks || right_count_ticks <= ticks)
7   {
8     digitalWrite(D_MRIGTH, LOW);
9     digitalWrite(D_MLEFT, HIGH);
10    analogWrite(V_MLEFT, PWM_MOTOR);
11    analogWrite(V_MRIGTH, PWM_MOTOR);
12  }
13 }
```

Esta función recibe un valor entero que es el número de pulsos necesarios para recorrer una distancia. No retorna ningún valor, ya que solo se encargara de activar el giro de las ruedas mientras se ejecuta un bucle como se muestran en las lineas del 6 al 12.

D_MRIGHT y D_MLEFT son constantes definidas que indican el numero del puerto digital a través del cual se envían las señales que indican la dirección del giro de cada rueda. V_MRIGHT, V_MLEFT, son constantes definidas que indican el puerto digital con modulación de ancho de pulso por el cual se enviaran la señales PWM_MOTOR que es otra constante que representa tanto el valor de la señal pwm (0 a 255) y la velocidad de giro de las ruedas. Por otro lado, LOW y HIGH, son constantes definidas por Arduino para referirse a los valores que puede tomar una señal digital: 1 o 0 lógico.

Finalmente, la condición que permite ejecutar las instrucciones del bucle está definida en la línea 6, donde left_count_ticks y right_count_ticks son los contadores que se incrementan al ejecutarse las interrupciones. Estos se pueden reestablecer cada vez que se manda a llamar a la función y se comparan con el número de pulsos recibidos (ticks), para saber si la rueda izquierda o la rueda derecha ya completó el recorrido de la distancia y simplemente detener su avance.

7.2 Movimiento rotacional del robot

Para este algoritmo se considera una idea similar a la anterior en donde se le podrá indicar al robot cuantos grados sexagesimales tiene que girar sobre su propio eje para que cambie de dirección su trayectoria.

En la figura 7.3, planteo la idea sobre como se lograría conseguir que el robot gire un ángulo $\alpha_F$para que su vista frontal, denotada por F, se traslade a un nuevo punto denotado por F'. Esta traslación es posible haciendo que las ruedas izquierda y derecha, cuyo centros se encuentra en los puntos L y R respectivamente, se desplazan en sentidos

opuestos[4*] sobre un par de trayectorias angulares denotadas por α_L y α_R y que además son equivalentes al ángulo α_F indicado al inicio.

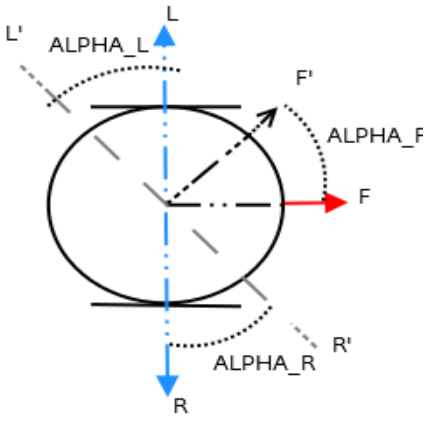


Figura 7.4. Movimiento rotacional del Robot I

Si se presta atención a los ángulos o desplazamientos angulares descritos por , α_L y α_R, se nota que se pueden manejar también como arcos de circunferencia delimitados por los puntos iniciales F, L y R y los puntos finales F', L' y R'.

La longitud de cada arco es equivalente a una porción de la circunferencia C que mide $2\pi r_{(robot)}$

Donde r_{robot} es la distancia que existe entre el centro del robot con cualquiera de sus ruedas, como se muestra en la figura 7.5.

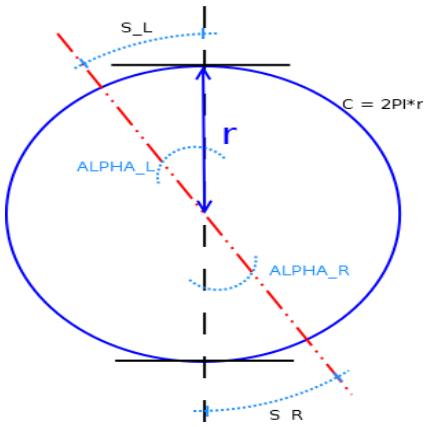


Figura 7.5. Movimiento Rotacional del robot II

[4] Recordar las características del movimiento de un robot de configuración diferencial: Si las ruedas giran a la misma velocidad en sentidos iguales, el robot avanza o retrocede en linea recta; si las ruedas giran en sentidos contrarios y a la misma velocidad, rota sobre su propio eje; Si giran en sentidos iguales y a velocidades diferentes, se desplazan sobre un arco de circunferencia.

Las longitudes de cada arco se pueden ver simplemente como distancias S_L y $S_R$que recorre cada rueda en sentidos contrarios para para lograr que el robot realice una rotación de α_F grados sobre su propio eje.

Para conocer esas distancias, se sigue un proceso similar al de convertir grados sexagesimales en radianes.
En este proceso se retoma la circunferencia $c = 2\pi r_{robot}$, descrita anteriormente.

Recordando que esa circunferencia equivale también a 360° sexagesimales, entonces se tiene que:

$$S_g = \frac{\left(2\pi r_{robot}\right)}{360} \ldots (1)$$

Donde S_g es una constante que representa la distancia que necesita recorrer cada rueda para hacer que el robot rote 1° sexagesimal.

Ahora siguiendo esta lógica, lo que se necesita hacer para obtener las distancias que debe de recorrer cada rueda para lograr que el robot gire $\alpha_F$grados es simplemente multiplicar la constante $S_g$por el número de grados sexagesimales que se desea rotar. Quedando entonces, las distancias para cada rueda representadas nuevamente como $S_x$en función de los grados $g°$ de rotación.

$$S_x(g°) = S_g * g° \ldots (2)$$

Como $S_x$es una distancia, se puede obtener ahora su equivalente a pulsos que necesita registrar el contador para saber que ya se ha recorrido. Reutilizando la expresión obtenida para el algoritmo anterior, se tiene:

$$TicksGenerados_x = \frac{\left(S_x * TicksPorRevolucion\right)}{(2\pi r)}$$

En este caso la constante $2\pi r$tiene que ver con la circunferencia de la rueda del robot.
Sustituyendo $S_x$con la expresión en (2), la ecuación quedaría entonces

$$TicksGenerados_x = \frac{\left(\left(S_g * g°\right) * TicksPorRevolucion\right)}{(2\pi r)} \ldots (3)$$

Si se agrupan todas las constantes que hay en la expresión anterior, para poner el numero de Ticks generados en función de g°, quedaría de la siguiente forma:

$$TicksGenerados_x(g°) = \frac{\left(S_g * TicksPorRevolucion\right)}{(2\pi r)}(g°) \ldots (4)$$

De esta manera, se podrían calcular fácilmente los pulsos o ticks necesarios que necesita generar cada rueda para lograr que el robot gire los grados g° deseados.
El diseño del algoritmo será igual al del caso anterior; se dividirá en dos partes el problema.

Primero: en un subprograma o función se calculará el número de ticks necesarios para cada rueda utilizando la ecuación (4). Figura 6.10-a
Segundo: mediante un bucle se enviará las señales de activación para las ruedas, pero esta vez para que giren en sentidos contrarios, mientras la cantidad de pulsos generados durante el movimiento de cada rueda sea menor a la cantidad de pulsos calculados en el paso anterior. Figura 6.10-b.

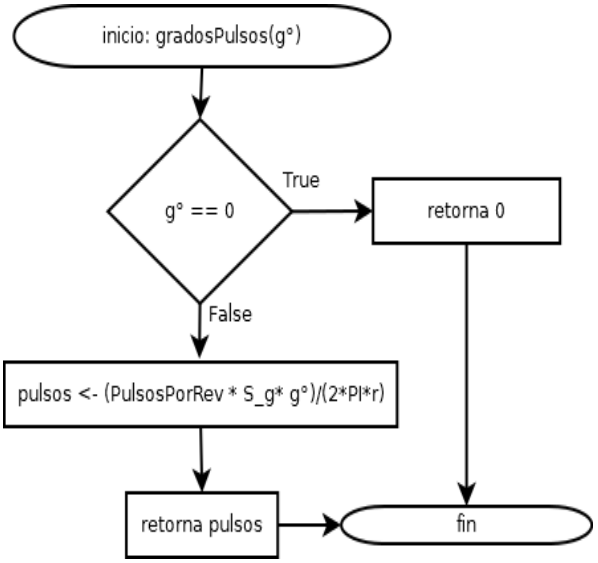


Figura 7.6-a

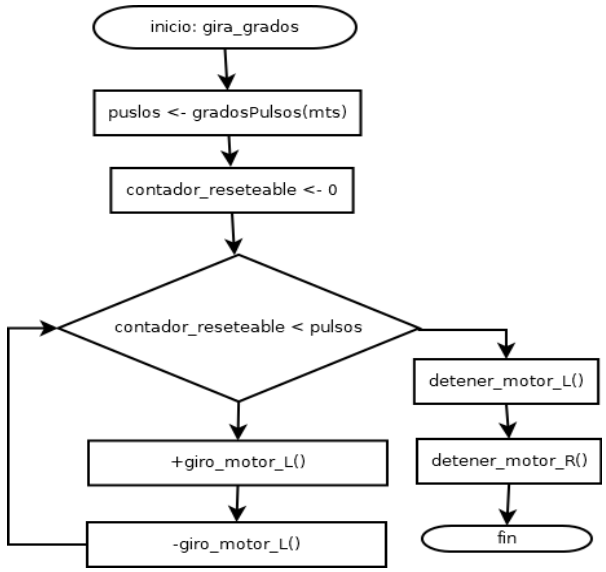


Figura 7.6-b

En mi caso, las constantes presentes en la expresión, tiene los siguientes valores:
$r_{robot} = 0.045m\ldots$ radio del robot
$r = 0.0225m\ldots$ radio de la rueda
$TicksPorRevolucion = 300$
$c = 9\frac{\pi}{100}m\ldots$ (circunferencia del robot)
$S_g = 0.785398163 * 10^{-3}$
Con estos valores, la expresión (4) se reduce a:
$TicksGenerados_x(g°) = 1.66666667(g°)$

Implementación del Algoritmo

Para la implementación del algoritmo en el lenguaje C++ de Arduino se crearán nuevamente dos funciones como en el algoritmo anterior.
degreesToTicks: se trata de una función que recibe como parámetro un tipo de dato double que representa el ángulo sexagesimal que se desea rotar al robot, y como salida devuelve un valor entero que representa el número de pulsos necesarios para lograr esta acción.

```
1 int degreesToTicks(double degrees)
2 {
3   if (degrees == 0)
4     return 0;
5
6   return (int) 1.66666 * degrees;
7
8 }
```

Esta función es esencialmente lo mismo que lo visto con la función metersToTicks(); primero se verifica de que el valor del grado recibido es 0 para simplemente retorna un valor 0 (líneas 3 y 4) y ahorrarse el cálculo. En caso contrario, devuelve la conversión a entero del resultado del producto entre la constante obtenida con la ecuación (4) como se muestra en la línea 6.

turnDegrees: Esta función se encarga de generar las señales de salida que activan los motores del robot para hacerlo girar hacia un sentido la cantidad de ángulos representados por un número de pulsos. Su primer parámetro en un valor entero que son los pulsos generados por la función anterior. Su segundo parámetro es un valor entero también pero que indica el sentido de giro. No devuelve nada al final de la ejecución.

```
1 void turnDegrees(int ticks, int turning_sense)
2 {
3   left_count_ticks = 0;
4   right_count_ticks = 0;
5
6   while (left_count_ticks <= ticks || right_count_ticks <= ticks)
7   {
8     digitalWrite(D_MRIGTH, turning_sense);
9     digitalWrite(D_MLEFT, turning_sense);
10    analogWrite(V_MLEFT, PWM_MOTOR);
11    analogWrite(V_MRIGTH, PWM_MOTOR);
12  }
13 }
```

Esta función es esencialmente igual a goForward, ya que también reinicia los contadores de los registros de pulsos censados en el encoder (lineas 3 y 4), ejecuta un bucle cuya condición de paro (linea 6) es que alguno de los registros contadores iguale o supere el número de pulsos recibidos en la función. La principal diferencia esta en el sentido de giro de cada rueda.

Para que el robot gire sobre su propio eje, ambas ruedas deben girar en sentidos contrarios. Esa dirección de giro está definida por el segundo valor entero recibido y se aplica en las líneas 8 y 9 del código anterior. La razón por la que ambas ruedas reciben exactamente el mismo valor para girar en sentidos contrarios se debe a la forma en la que está configurada su conexión. (Revisar el capítulo 4).

7.3 Obtención de Velocidades

Para realizar el cálculo de las velocidades angulares y lineales del robot será necesario analizar la siguiente gráfica.

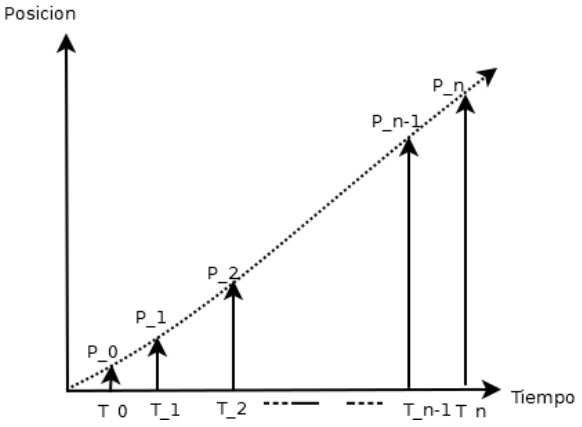


Figura 7.7. Gráfica del cambio de posiciones de la rueda respecto al tiempo.

En la gráfica de la figura 6.11 se muestran los cambios de posición angular de la rueda registrados por el encoder en cada momento del tiempo; básicamente una relación unívoca de la posición en función del tiempo.

Para obtener de una forma muy sencilla la velocidad angular de la rueda en cualquier momento del tiempo, bastaría con realizar un muestreo de la función descrita en la figura anterior como se muestra en la siguiente gráfica:

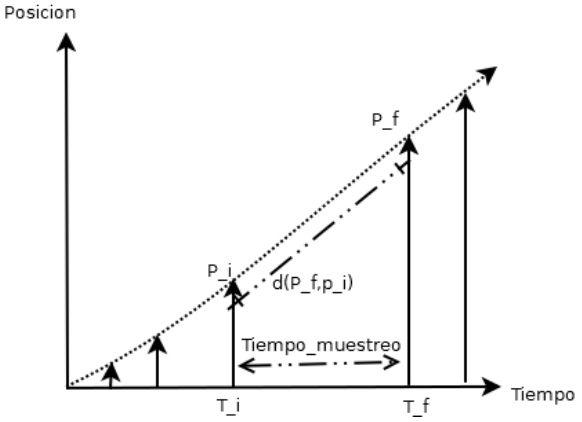


Figura 7.8. Muestreo de la función de la posición respecto al tiempo.

De la figura 6.12 se logra recuperar la siguiente información:

Un tiempo de muestreo delimitado por un tiempo inicial y un tiempo final.

Una diferencial de la posición delimitada por un punto punto inicial y un punto final, que representan las posiciones anterior y actual de la rueda respectivamente.

Recordando la ecuación básica de la velocidad se tiene:

$$v = \frac{d}{t}$$

Donde la distancia d es un cambio de posición. Para este caso, dicho cambio de posición está determinado por la diferencia de los puntos p_f y p_i.

En cuanto al tiempo t, este resulta ser igual al tiempo de muestreo, pues es en él donde se lleva a cabo el cambio de la posición angular de la rueda.

Como en este caso se está obteniendo una relación del cambio de posiciones angulares de la rueda respecto al tiempo, en vez de utilizar la literal $v$se utiliza la literal w para dejar en claro de que se trata de una velocidad angular; quedando entonces la expresión anterior como:

$$w = \frac{(p_f - p_i)}{t_{muestreo}} \ldots \text{velocidad angular}$$

Si ahora se desea obtener la velocidad lineal de la rueda, solo hay que aplicar una operación muy sencilla, la cual consiste en un producto de la velocidad angular de la rueda por su radio, quedando entonces:

$$v = w_{rueda} * r_{rueda} \ldots \text{velocidad lineal}$$

En este caso se utiliza la literal v para especificar de qué se trata de la velocidad lineal.

Descripción del algoritmo

Según el análisis realizado anteriormente, los pasos que debe de seguir el algoritmo para calcular en todo momento la velocidad de cualquiera de las ruedas del robot, serían:

1. Determinar un periodo de tiempo que será el tiempo de muestreo en el que se registrara el cambio de posición angular de la rueda.
2. Mantener un registro de modo que se pueda conocer cuál es la posición angular de la rueda al inicio del tiempo de muestreo, y cuál es la posición angular de la rueda al final del tiempo de muestreo.
3. Aplicar la fórmula de velocidad angular con los registros obtenidos y el tiempo de muestreo determinado en el primer paso.
4. Convertir la velocidad angular en lineal para obtener finalmente ambas velocidades.

El algoritmo se muestra en la siguiente figura

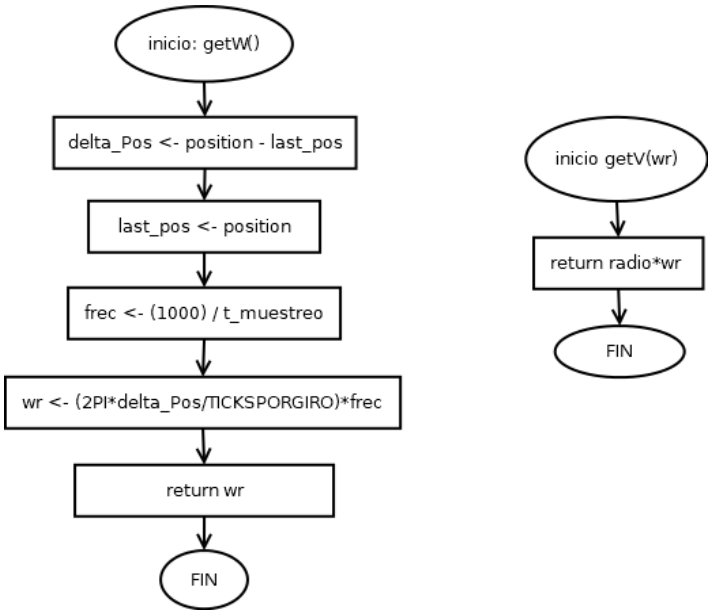


Figura 7.9 Algoritmo para obtención de velocidades

Del los pasos descritos anteriormente, fácilmente se pueden implementar y sin problemas los pasos del 2 al 4; ya que para mantener un registro de posiciones al inicio y final del muestreo, se harían uso de dos variables:

- last_position: inicializada en 0 (cero), y que se actualiza con la posición actual cada vez que se cumple el tiempo de muestreo
- current_position: es ni más ni menos que el contador que se ejecuta cada vez que se genera una interrupción debido a la lectura de la señal proveniente del encoder, y que recordemos, registra la posición angular generada por el giro de la rueda.

Para la obtención de las velocidades, simplemente se tendrían que aplicar las ecuaciones generadas en el análisis.

Sin embargo, el verdadero reto que es interés para resolver este algoritmo, es encontrar un método a través del cual se pueda determinar y controlar el tiempo de muestro con el cual se completan los cálculos de velocidad.

Se pueden pensar en 3 posibles métodos que se pueden explorar para resolver este problema. A continuación los describo.

1 Utilizar un delay de Arduino.

Quizás este sea el método que uno intentaría primero (sobre todo si se es un desarrollador “novato” en Arduino) para llegar a una solución. Este método consiste en utilizar la función delay() que viene en la biblioteca nativa del micro-controlador, y simplemente es una función que congela, o detiene el flujo de los programas en Arduino durante un tiempo señalado en milisegundos (ms), es decir, que durante una cantidad x de mili-segundos no hará nada más que solo ejecutar la última instrucción registrada.

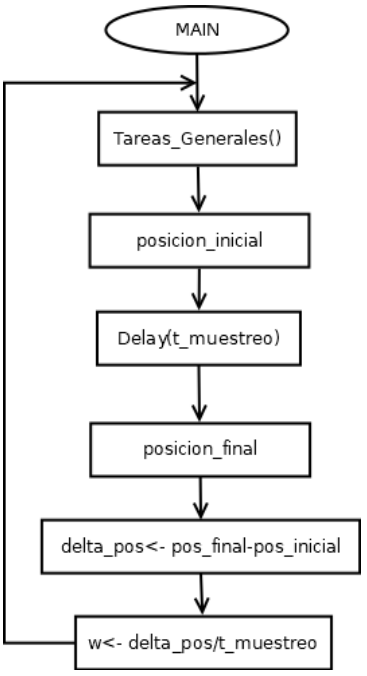


Figura 7.10 Algoritmo con delay

Imaginemos el siguiente caso de uso:

Se desea conocer cuál es la razón del cambio de posición de una de las ruedas durante un segundo, para obtener una velocidad angular medida en r/s y posteriormente una velocidad lineal en m/s. Entonces en el flujo principal de se aplica un delay(1000), para que la rueda se quede girando durante 1 segundo y obtenga la diferencia de su posición angular antes y después del delay().

Sin embargo, esto supone un grave problema para el funcionamiento general del robot, ya que durante un segundo, Arduino no hará nada más que girar una rueda e incrementar el contador de posiciones[5].

Durante ese segundo puede ocurrir que, como solamente está girando una rueda del robot, su trayectoria cambie drásticamente. Además puede que se presenten obstáculos que los

[5] Aún con el delay() aplicado, el contador sigue ejecutándose, ya que este es generado por las interrupciones por hardware que tiene Arduino. Esta instrucción no pertenece al flujo principal de Arduino, así que no la afecta el delay

sensores simplemente no van a detectar y por ende el robot no sabría nada de su entorno durante un segundo, por lo que su navegación quedaría afectada completamente.

Además el robot tiene dos ruedas, así que el delay(1000), se aplica también para la segunda rueda, por lo que se estarían sacrificando 2 segundos o 2 X milisegundos su funcionamiento. Por estas razones, el manejo de delay() como un método de solución no es útil para este caso.

La idea es que el microcontrolador permita que el robot tenga la capacidad de ejecutar múltiples tareas “al mismo tiempo”, de modo que calcule y controle su velocidad mientras se desplaza, y también verifique la ubicación de la meta y detecte los obstáculos a través del resto de sensores y comparta datos con la placa Raspberry.

Lo anterior hace pensar entonces en la necesidad de implementar una programación concurrente utilizando hilos para hacer posible la multi-tarea, le que lleva al siguiente método.

2 Programación Concurrente en Arduino.

La idea de implementar programación concurrente en este microcontrolador es que en hilos diferentes se ejecuten cada uno de los algoritmos que van a controlar los sensores de obstáculos, sensores de luz y por supuesto, el cálculo de velocidades para su posterior control. Todo sin afectar al funcionamiento del hilo principal donde tal vez se estaría ejecutando el movimiento del robot.

Si se pusiera en un hilo independiente el algoritmo para calcular las velocidades del robot, aquí sin problemas podría funcionar la estrategia con delay, ya que el flujo se congelaría únicamente en ese hilo.

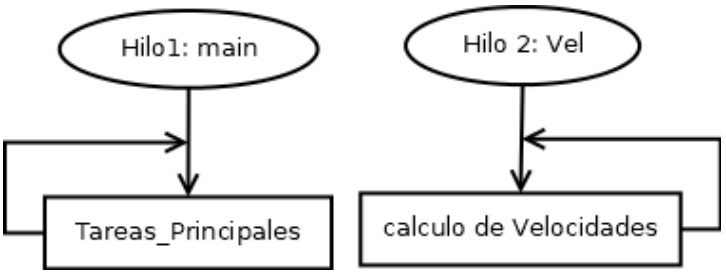


Figura 7.11 Programación concurrente en Arduino

Sin embargo, hay un serio problema por el cual tampoco sería posible utilizar este método, y eso tiene que ver directamente con la arquitectura de Arduino.

Como se recordará, Arduino UNO que es el microcontrolador elegido para este proyecto, solo posee un procesador (y en general los demás modelos), por lo que solo puede ejecutar una sola tarea al mismo tiempo, es decir, que todos los programas en Arduino se ejecutan de forma estrictamente secuencial; no hay forma de implementar hilos para programación concurrente.[6]

Ante esta situación quedaría entonces explorar otro método que permita de algún modo imitar o emular la multitarea en Arduino. Si bien, se podría hacer uso de las

[6] De hecho mientras exploraba opciones, descubrí que no existen bibliotecas oficiales para manejar Multi-Tasking en Arduino, debido a la simplicidad de su arquitectura. Las bibliotecas que existen por parte de la comunidad, solo emulan la programación concurrente.

herramientas que proporciona la enorme comunidad de Arduino para atender estas situaciones especiales, hay una alternativa mucho mas sencilla que solo hace uso de las herramientas nativas que proporciona el lenguaje de programación para Arduino, y la voy a explorar a continuación.

3 La función millis() en Arduino.

La función millis() de Arduino es una función nativa que, según su documentación oficial, solo se encarga de obtener el tiempo en milisegundos de que lleva activo el microcontrolador, es decir, desde que empezó a correr el programa.

La estrategia para utilizar millis(), es que se puede mantener un registro de tiempos similar al registro de posiciones descritos anteriormente, para que al obtener la diferencia entre los tiempos registrados, se tenga entonces un tiempo de muestreo.

Dicho lo anterior, se definen entonces las siguientes variables:

last_time: que es el ultimo tiempo registrado, inicializado en cero y actualizable.

current_time: Es el tiempo actual y devuelto por la función millis().

delta_time: La diferencia de ambos registros.

Añadiendo una instrucción condicional al flujo principal del programa y utilizando como condición que para ejecutar el calculo de velocidades, el valor obtenido en delta_time debe de ser igual al de una constante que representa el tiempo de muestreo que se desea utilizar para calcular las velocidades.

El algoritmo quedaría entonces:

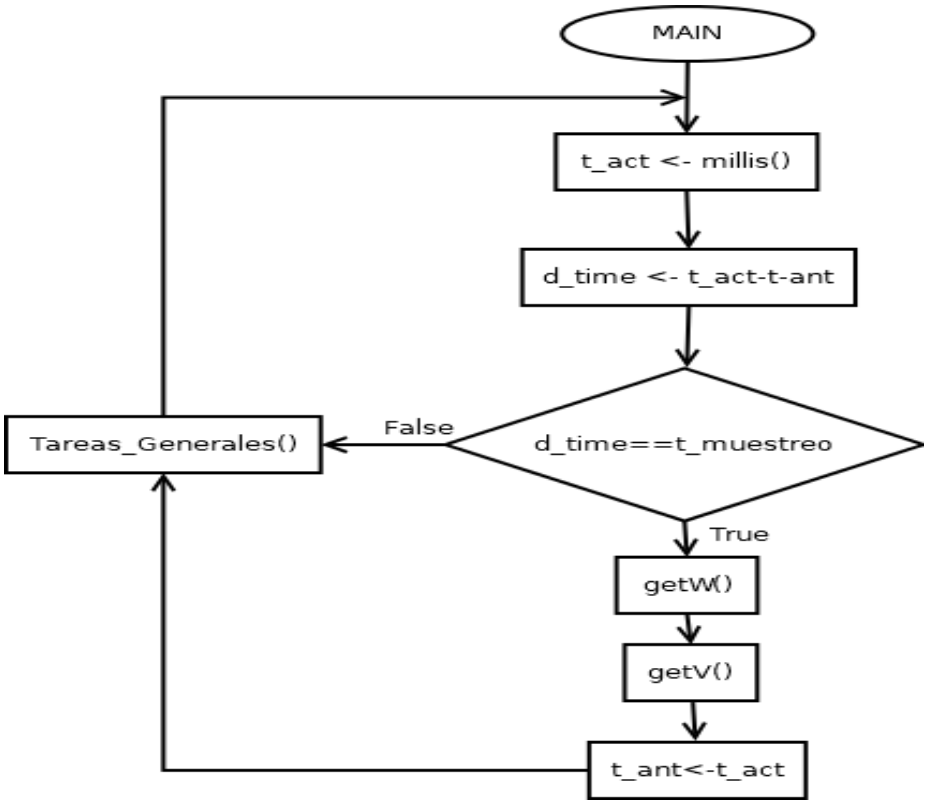


Figura 7.12 Algoritmo para el control de ejecución de subprogramas utilizando millis()

Implementación del Algoritmo

La implementación de este algoritmo sera especial debido a que se tendrá que manejar el estilo de programación de una forma diferente a la que ya se esta acostumbrado y quizás se pueda considerar un tanto caótico el código que se genera como resultado.

Sin embargo este es un “mal necesario” para poder aprovechar correctamente los beneficios de la función millis() en Arduino, ya que estos permitirán crear intervalos de tiempo en los cuales se interrumpe instantáneamente el flujo principal del programa y ejecute una cierta función específica y que además se recomienda que esté lo mejor optimizada posible para que no ocupe demasiado tiempo de ejecución.

Para poder calcular las velocidades del robot, obviamente este debe de estar en movimiento, de lo contrario, las velocidades siempre serían cero. Así que la ejecución de estos cálculos deben de estar embebida en las funciones que realizan la navegación del robot. Por ejemplo, dentro de la función goForward.

```
1 void goForward(int ticks)
2 {
3   left_count_ticks = 0;
4   right_count_ticks = 0;
5
6   while (left_count_ticks <= ticks || right_count_ticks <= ticks)
7   {
8     digitalWrite(D_MRIGTH, LOW);
9     digitalWrite(D_MLEFT, HIGH);
10    analogWrite(V_MLEFT, PWM_MOTOR);
11    analogWrite(V_MRIGTH, PWM_MOTOR);
12  }
13 }
```

En el código anterior se encuentra la estructura de la función goForward; de aquí solo será de interés el bloque while, ya que es en esta estructura de control donde se lleva a cabo el movimiento del robot.

```
6   while (left_count_ticks <= ticks || right_count_ticks <= ticks)
7   {
8     digitalWrite(D_MRIGTH, LOW);
9     digitalWrite(D_MLEFT, HIGH);
10    analogWrite(V_MLEFT, PWM_MOTOR);
11    analogWrite(V_MRIGTH, PWM_MOTOR);
12  }
```

Se definen tres nuevas variables a nivel global, que servirán para mantener los registros del tiempo utilizando por supuesto la función millis

```
1  volatile unsigned  current_time = 0;
2  volatile unsigned last_time = 0;
3  volatile unsigned delta_time = 0;
```

Las tres variables anteriores se definieron como volatile unisgned debido a que tomaran valores enteros positivos todo el tiempo, y además cambiarán su valor con demasiada frecuencia.

Ahora dentro del bucle while de la función goForward, la variable current_time tomará el valor que le devuelva la función millis(), y esta sera la primer instrucción que se ejecute en cada iteración que realice el bucle, es decir, que lo primero que se hará es obtener el tiempo actual de ejecución.

```
8  current_time = millis();
```

La instrucción que le sigue será actualizar el valor de la variable delta time para obtener una primer muestra del tiempo; esta actualización se obtiene al ejecutar una simple resta entre el valor de current_time y el valor de last_time quien en la primer iteración todavía posee el valor con el que se inicializó, o sea, cero.

```
9  delta_time = current_time - last_time;
```

A continuación se añade un bloque de ejecución condicional, en la cual se comparará el valor de delta_time con el de una constante entera que representa el tiempo de muestreo con el cual se va a calcular la velocidad. La comparación se realiza con un operador booleano, mayor o igual que (>=), y si la condición se cumple, entonces dentro de este bloque se mandan a llamar a las funciones que calculan las velocidades, en caso contrario, continua con la ejecución del bucle.

```
10  if (delta_time <= constante){
11     //funciones que calculan las velocidades
12  }
```

Una vez que se ejecuten los cálculos de velocidades, se deberá de actualizar el valor de la variable last_time, simplemente tomando el valor del current_time.

```
10  if (delta_time <= constante){
11     //funciones que calculan las velocidades
12     last_time = current_time;
13  }
```

La forma en la que se manejan las instrucciones desde la linea 8 hasta la 13, es como se logra imitar la programación concurrente en Arduino, ya que al mantener un registro de tiempos de ejecución y una obtención de tiempos de muestreo, se pueden crear interrupciones en ciertos momentos de la ejecución principal que obliguen a Arduino a ejecutar otras tareas de forma instantánea y den la sensación de que en realidad se están ejecutando de forma paralela al flujo principal del programa en Arduino.

Fuera del bloque condicional, se conservan las instrucciones que mandan las señales de activación para las ruedas del robot, esto con el propósito de generar movimiento en el robot en lo que el delta_time toma un valor mayor o igual al del tiempo de muestreo deseado.

```
13      digitalWrite(D_MRIGTH, LOW);
14      digitalWrite(D_MLEFT, HIGH);
15     analogWrite(V_MLEFT, PWM_MOTOR);
16     analogWrite(V_MRIGTH, PWM_MOTOR);
```

Implementación de los algoritmos para calcular velocidades

En cuanto a los algoritmos para cálculos de velocidad muestro a continuación la implementación realizada.

get_W_X: Se trata de la función que encarga de calcular la velocidad para una rueda x[7], aplicando el razonamiento desarrollado anteriormente. Esta función no recibe ningún parámetro, pero si devuelve un valor que es de tipo double y que representa la velocidad angular medida en $\frac{r}{s}$

```
double get_W_X(){
  //código
}
```

Como primer paso importante, se definen tres variables globales que representan los registros de las posiciones actual, anterior y la distancia recorrida por la rueda X.

```
1   volatile unsigned  x_position = 0;
2   volatile unsigned x_last_position = 0;
3   volatile unsigned x_delta_position = 0;
```

Nuevamente estas variables son de tipo volatile unsigned ya que solo almacenan valores positivos y que además cambian muy rápido.

La variable x_position se actualiza cada vez que el encoder asociado a la rueda x genera la interrupción. Las variables x_last_position y x_delta_position se actualizan en dentro de la función como se muestra a continuación:

```
1   x_delta_position = x_position – x_last_position; //obtiene la distancia recorrida
2   x_last_position = x_position;//actualiza la posicion anterior con la posicion actual
```

La variable x_delta_position, como ya se mencionó antes, representa la distancia recorrida, pero esta distancia recorrida ha sido solamente durante el periodo de tiempo establecido por el tiempo de muestreo utilizado para calcular las velocidades.

Para poder calcular cuanta distancia se recorre durante un segundo, se puede calcular primero la frecuencia con la que se repite el tiempo de muestreo durante un segundo, teniendo entonces el siguiente código.

```
3   double frecuencia = (1000) / TIEMPO_MUESTREO;
```

[7] Por meros fines prácticos estoy nombrando a la función con un sufijo ‘X’, y refiriéndome a la rueda como ‘x’ ya que todo el algoritmo que se define dentro de la función es exactamente lo mismo tanto para la rueda izquierda o derecha, salvo que en cada caso, se realizan operaciones con las lecturas de sus respectivos encoders.

Como la variable frecuencia almacenará un cociente que podría contener un valor en números reales se declara como double; el cociente es un número generado por la división de 1000 (mil), que son los milisegundos que hay en un segundo, entre la constante que es el tiempo de muestreo, que para mi caso también son milisegundos.
El siguiente paso corresponde a calcular la velocidad angular de la rueda medida durante el tiempo de muestreo proporcionado utilizando la siguiente expresión:

$$wr = \left(\frac{2*\pi*x_{position}}{PulsosPorRev}\right) * frecuencia$$

Si se le presta atención a la expresión se notará inmediatamente que se está utilizando nuevamente la ecuación con la que se obtienen distancias, solo que en este caso no se incluye la constante que representa el radio de la rueda, por lo que solo se obtendría el desplazamiento angular; además el valor resultante de dicha operación se esta multiplicando por la frecuencia, a modo de obtener el desplazamiento angular total que se da en un segundo, quedando un valor que representa la velocidad angular medida en rad/seg. El valor generado es finalmente retornado.
El código es el siguiente

```
4  return ((2 * PI * x_position) / PULSOSPORGIRO) * frecuencia;
```

Ahora en cuanto a la implementación del algoritmo para obtener la velocidad lineal de la rueda, se crea una función demasiado sencilla y se muestra a continuación:

```
1  double get_V(double wr){
2  return RADIORUEDA*wr;
3  }
```

Tal y como se muestra en el código anterior, la función recibe un parámetro de tipo double que representa la velocidad angular medida en cualquiera de las ruedas, y retorna un valor double que es la velocidad lineal obtenida al multiplicar la constante RADIORUEDA por el valor de la velocidad angular recibido.

Respecto a los registros contadores de posición.

Como complemento a los algoritmos desarrollados a lo largo de este capítulo, solamente añado un ajuste que considere necesario para el buen funcionamiento de los algoritmos.

El ajuste es simplemente tener para cada rueda, dos contadores que van a a almacenar las pociones que va adoptando las ruedas del robot durante su recorrido, ya la razón de esto es porque deseo controlar correctamente los valores registrados para que los cálculos necesarios para la medición de velocidades, así como de las distancias recorridas, se realicen correctamente.

Si presto atención al desarrollo de los algoritmos, me referí en ciertos momentos a un registro contador de pulsos, como un contador reseteable y lo utilice para medir las distancias recorridas durante el movimiento lineal y rotacional del robot, y determinar si se debía o no detener el movimiento del robot.

Por otro lado, cuando trate el tema de las velocidades el registro contador de pulsos ya no era resetable, pues este iba a estar almacenando las pociones del robot durante todo el tiempo que estuviese activo, para que también pudiese medir en todo momento las velocidades. De haber tenido un solo contador, me arriesgaba a tener cálculos incorrectos o simplemente hubiese sido complicado realizar correctamente las operaciones.

```
// ambos registros se actualizan durante la interrupcion.
1  volatile unsigned x_position = 0;
2  volatile unsigned x_position_reseteable = 0;
```

Para finalizar este capítulo, solamente mencionare algunas consideraciones sobre el porque no incluí un algoritmo que implemente el uso de la expresión que se obtuvo para el caso de un robot que avanza y gira.

$$\begin{bmatrix}\omega_R & \omega_L\end{bmatrix} = \frac{1}{(2\pi r)}\begin{bmatrix}1 & -\left(\frac{d}{2}\right) \\ 1 & \frac{d}{2}\end{bmatrix}\begin{bmatrix}V_{Robot} & \omega_{Robot}\end{bmatrix}$$

La razón es porque en este momento todavía es realmente irrelevante aplicar esta expresión. Donde va a tener más sentido su uso, es cuando se logre tener un sistema que permita asegurar que el robot se va a mover todo el tiempo a las velocidades Lineal y Angular deseadas. Así, se podría crear una función que se encargue de implementar un método numérico para resolver el sistema de ecuaciones y obtener las velocidades con las que tiene que girar las ruedas para que se cumpla el objetivo.

El desarrollo de dicho sistema, se verá en los siguientes capítulos cuando se trate sobre el control PID para la velocidad del robot.

Capítulo 8: Control del robot móvil

8.1 La necesidad del uso de un control para el robot

Con todos los algoritmos que se han diseñado e implementado en base al análisis cinemático, el robot ya posee la capacidad de ejercer movimientos que le permitan navegar por un entorno.

Sin embargo, aún se tiene la posibilidad de encontrarse con múltiples errores en la navegación, debido a que durante la ejecución de la misma, esta se vera afectada por múltiples perturbaciones en el medio que podrían provocar comportamientos indeseables al ejecutar los movimientos del robot.

Dichas perturbaciones podrían ser desniveles, deslices u objetos que impidan que las ruedas mantengan una velocidad angular estable, y debido a la naturaleza de los movimientos que posee el robot de configuración diferencial, se podrían crear trayectorias erróneas. Por ejemplo:

Caso 1:

Al intentar Avanzar en línea recta hacia el frente, si una o ambas ruedas de tracción giran a velocidades diferentes, la trayectoria no sera recta; seria mas bien una trayectoria que presenta oscilaciones dibujando una especie de zigzag.

Incluso, si una de las ruedas llegase a mantener un giro muy por debajo de la velocidad a la que se desea que avance el robot, la trayectoria recta podría terminar convirtiéndose en una curva.

Caso 2:

Si se desea que el robot avance y gire a ciertas velocidades lineal y angular a través de una trayectoria curva, y también no se mantienen constantes las velocidades de las ruedas, la trayectoria nuevamente presentaría oscilaciones o incluso generaría trayectorias completamente diferentes a la deseada.

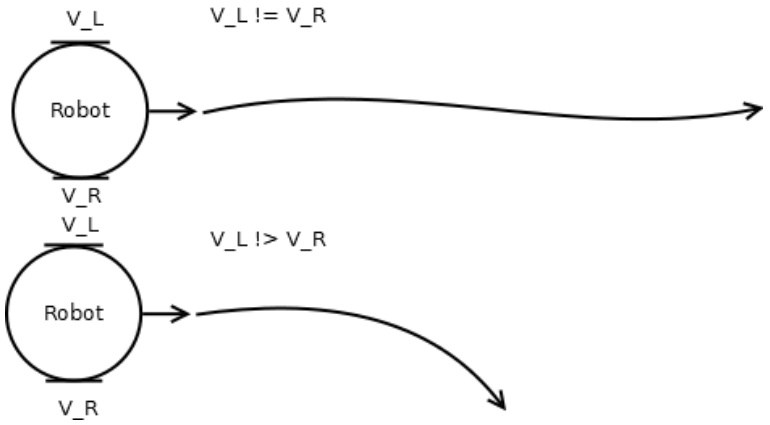


Figura 8.1. Problemas en la velocidad de las ruedas del robot

Debido a estos problemas que están presentes en la navegación del robot se vuelve necesario el pensar en un método que permita resolver tal situación; dicho método, es la implementación de un sistema control que se encargue de ajustar de forma automática las velocidades de las ruedas para que se garantice que la navegación del robot se mantenga lo más estable posible.

8.2 Control de velocidad

Para poder llevar a cabo la implementación de un sistema de control que regule de forma automática la velocidad angular de una rueda del robot, sera necesario implementar un control a lazo cerrado, es decir, un tipo de control que requiere de retroalimentación para comparar el valor de referencia a la entrada del controlador y el valor a la salida de la planta; de modo que se genere una señal de error que obligue al controlador a ajustar las señales de control para mejorar la respuesta del sistema.

El diagrama de bloques de la figura 8.2 muestra la forma en que se aplicará el control de velocidad.

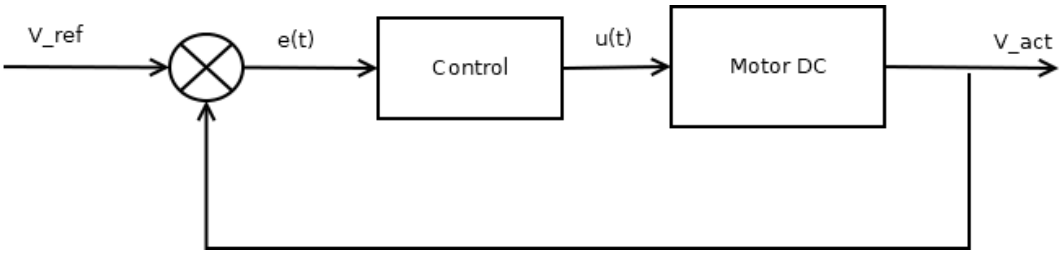


Figura 8.2. Diagrama de bloques del control a lazo cerrado para la velocidad de una rueda

Donde

V_{ref} = Velocidad angular a la que se desea que gire la rueda del robot.

$e(t)$ = Señal de error generada por la diferencia entre V_{ref} y V_{act}

Control = es el sistema de control elegido

$u(t)$ = señal de control

Motor DC = Planta sobre la que se aplicará el sistema de control

V_{act} = Velocidad angular real del motor

El diagrama anterior sugiere que el sistema de control que se desea emplear va tomar como entrada una velocidad de referencia la cual se va a comparar con la velocidad real que se está midiendo en el motor, de modo que se genere una señal de error con la cual el control debe de generar una señal tal que al aplicarse sobre el motor, este ajuste su velocidad de modo que se aproxime a la velocidad de referencia y se reduzca el error en la entrada.

Ahora es momento de cuestionarse, cuál es esa señal de control que se tiene que generar para que al aplicarse sobre el motor se ajuste su velocidad. La respuesta la da el modelo matemático del motor que se muestra a continuación.

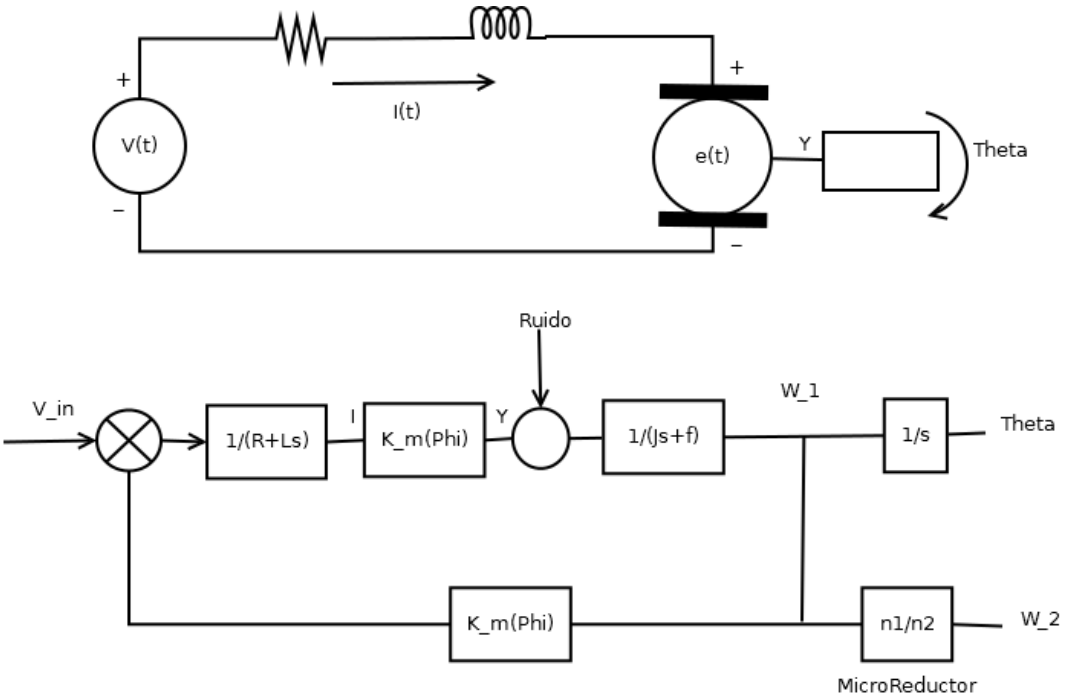


Figura 8.3. Esquema eléctrico de un motor DC y su modelo matemático representado en diagrama de bloques

En la figura 8.3 se tiene primero un esquema que representa tanto la parte eléctrica como la mecánica del motor de corriente directa (DC). En este esquema, en el apartado eléctrico, se aprecia que el motor tiene un voltaje de entrada que alimenta al sistema, una corriente que circula al interior del sistema y que es afectada por el valor de la resistencia y el inductor en la armadura del motor. De igual forma se tiene un valor E(t), que es el voltaje de fuerza electromotriz presente en el rotor y que realimenta al sistema. Finalmente en la parte mecánica, a la salida Y se tiene una carga que el motor hace girar, y donde se hacen presentes el torque $T_m(t)$, la inercia Jy la fricción F.

Mientras tanto en el diagrama de bloques de la misma figura 8.3, se representa el modelo matemático del motor junto con dos de sus diferentes salidas: la velocidad angular ω_1 y la posición angular θ, esta última se obtiene únicamente agregando una integral a la salida de la velocidad angular.

De estas salidas solo interesa ver la que representa a la velocidad angular del motor, cuya función de transferencia es:

$$G(s) = \frac{\omega(s)}{V(s)} = \frac{k_m}{(Js+f)(R+Ls)+K^2m}$$

Esta función de transferencia describe la relación que hay entre la velocidad angular (ω_1) a la salida y el voltaje que se le aplique a la entrada al motor.

En palabras más simples, también se puede entender que el motor tiene una velocidad angular muy alta o muy baja cuando se le aplica un voltaje muy alto o muy bajo respectivamente, por lo tanto, la señal de control que se tiene que generar para este sistema tiene que ser una señal de voltaje que obligue al motor a ajustar su velocidad angular, de modo que se aproxime a la referencia.

Adicional, se tiene una segunda velocidad angular ω_2 en el diagrama de bloques de la figura 8.3; está representa la velocidad angular con la que gira la rueda conectada a la salida del micro-reductor representado en el diagrama como la relación $\frac{n_1}{n_2}$, que es la función de transferencia de los engranes del micro-reductor.

En realidad la velocidad angular que se va a censar y se va a tomar en cuenta para hacer la comparación con la referencia y que se genere la señal de error que alimentará al sistema de control, será la velocidad angular ω_2. El cálculo de esta velocidad angular, ya se realizó en los capítulos anteriores cuando se trató sobre las ecuaciones de la cinemática del robot y los algoritmos para calcular las velocidades angulares y lineales de la rueda.

8.2.1 Control PID

El tipo de control elegido para regular la velocidad de las ruedas es el famoso control PID. La razón, es simplemente por las ventajas que proporciona su implementación en las que sobresalen la robustez para adaptarse a las perturbaciones que se presenten en el sistema, así como la sencillez que tiene su implementación, la cual no depende tanto de realizar un modelado matemático de la planta sobre la que se va a aplicar.

El funcionamiento del control PID se puede entender de la siguiente manera:

Consta de tres controladores, el control proporcional P, el control Integral I, y el control D.

Control P

Este tipo de control tiene como objetivo acercar el valor de la señal de salida de la planta al valor de la señal de referencia simplemente multiplicando la señal de error a su entrada por una ganancia representada por la constante K_p.

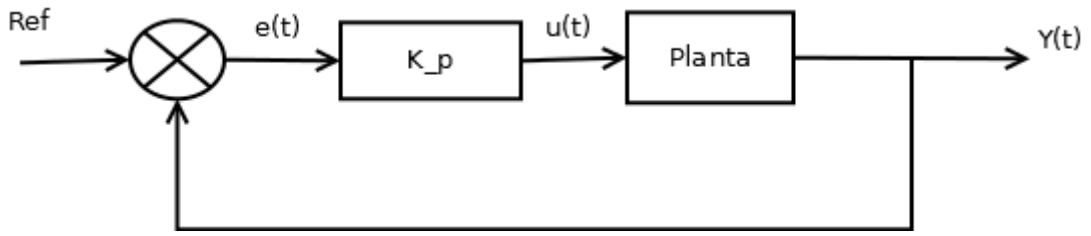


Figura 8.4. Control Proporcional

La expresión matemática del control P es la siguiente: $u(t) = K_p e(t)$

Sin embargo el controlador P presenta algunas desventajas empezando por que jamás será capaz de igualar la señal de salida con la señal de referencia, ya que en el momento en que el sistema alcance su estado estable, es decir, que el error sea 0 (cero), la señal de control también será 0 (cero) y en consecuencia el valor de salida se alejaría de la referencia generando nuevamente una señal de error aún mayor, por lo que se repetiría el proceso indeterminadamente.

Además, se requiere de que la ganancia K_p tenga un valor adecuado, ya que en caso de que este sea muy pequeño, se prolongaría demasiado el tiempo en el que la señal de salida se acerque a la referencia. Si por el contrario se tiene un valor muy grande, el tiempo

de respuesta sería demasiado corto y debido al error en el estado estable se podría provocar oscilaciones o inestabilidad de modo que llegaría a dañar al funcionamiento de la planta.

Control I

El controlador I genera su señal de control a partir de una integral definida aplicada sobre la curva de error con la intención de obtener un valor de referencia que indica el "comportamiento" el error a lo largo del tiempo; desde el tiempo inicial $t_0$cuando empieza a operar el sistema, hasta el tiempo actual t_a. Básicamente es un promedio del error.

Cuando el control se utiliza sobre un sistema analógico, la integración se hace mediante una integral definida, como lo muestra la figura 8.x

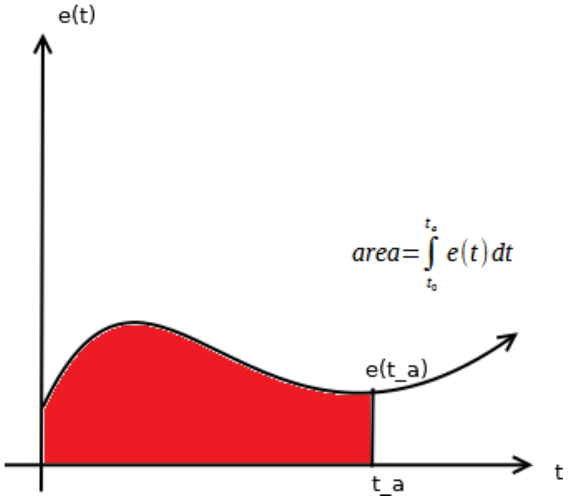


Figura 8.5. Integración de la curva de error mediante integral definida

Cuando el control se desea utilizar sobre un sistema digital, la integración del error se hace aproximandola mediante la regla trapezoidal como indica la figura 8.x.

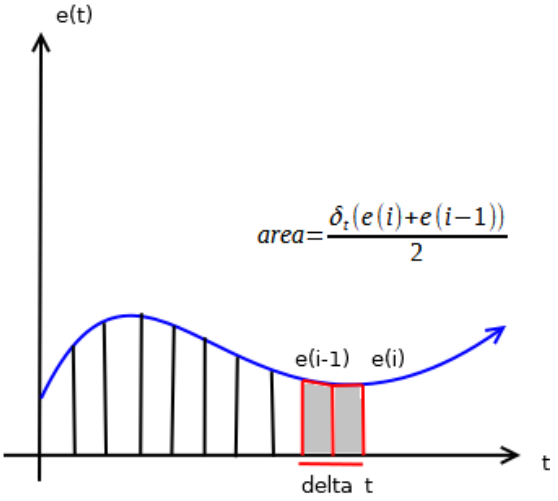


Figura 8.6. integración de la curva de error mediante regla trapezoidal

Este tipo de control tiende a actuar muy lento, por lo que para mejorar el tiempo de respuesta, se le añade una ganancia que es la constante de integración K_i, quedando la expresión matemática del control integral como sigue:

En tiempo continuo para sistemas analógicos:

$$u(t) = K_i \int_{t_0}^{t_a} e(t)dt$$

En este caso, la constante K_i se obtiene a partir de la relación entre una constante de proporcionalidad y el tiempo de integración T_i

$$K_i = \frac{K_p}{T_i}$$

En tiempo discreto para sistemas digitales:

$$u(t) = K_i \frac{\delta_t\left(e\left(t_{a-1}\right)+e\left(t_a\right)\right)}{2}$$

En este caso, la constante K_i se obtiene a partir del producto de la relación entre una constante de proporcionalidad y el tiempo de integración T_i por la diferencial de tiempo δ_i que representa a la base del trapezoide.

$$K_i = \delta_t \frac{K_p}{T_i}$$

La siguiente figura muestra el diagrama de bloques de un control Integral aplicado a una planta:

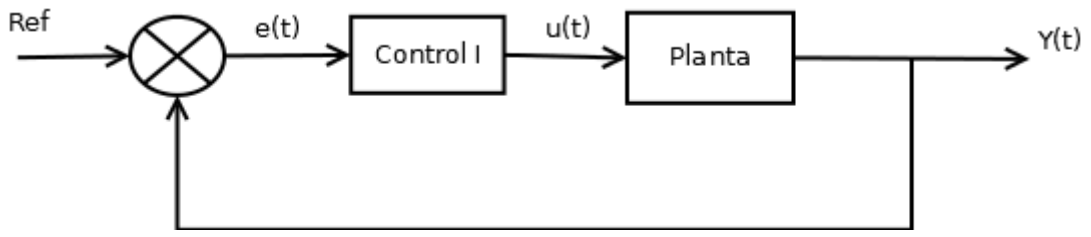


Figura 8.7. Control Integral

Control PI

Este control es el resultado de sumar las salidas del control Proporcional P y el control Integral I. La intención de esta combinación es que mediante el uso del control I se reduzca el error en estado estable del control P.

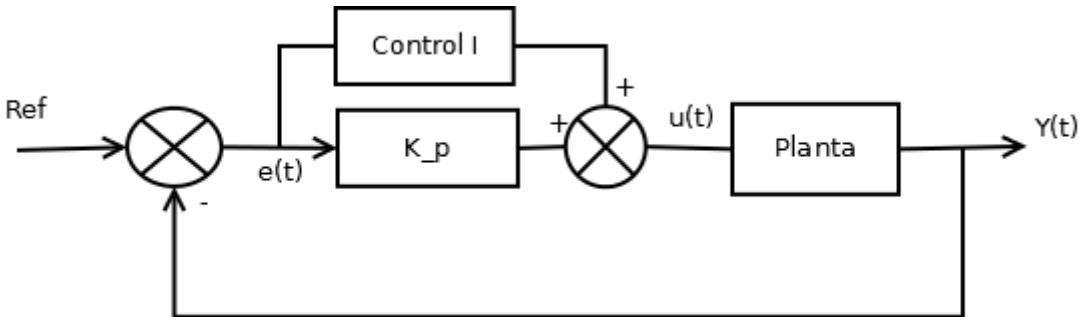


Figura 8.8. Control PI

La expresión matemática para este control es la siguiente:

$$u(t) = K_p\left[e(t) + \frac{1}{T_i}\int_{t_0}^{t_a} e(t)dt\right]$$

Debido a que este sistema de control se implementará utilizando el microcontrolador del robot para censar los datos de la velocidad y realizar los cálculos necesarios para ajustar el voltaje de alimentación del motor de cada rueda, se requiere que este control sea digital.
Por lo tanto, se discretiza la expresión anterior quedando como:

$$u_n = K_p e_n + \frac{Kp}{T_i}\delta_t \sum_{i=1}^{n} \frac{\left(e_i + e_{(i-1)}\right)}{2}$$

Para simplificar la expresión anterior se aplica recursividad, lo que da como resultado la siguiente expresión:

$$u_n = u_{(n-1)} + K_p\left(e_n - e_{(n-1)}\right) + K_i\frac{\left(e_n + e_{(n-1)}\right)}{2}$$

Controlador D

El control D viene a actuar como un complemento para la parte Proporcional e Integral, y su acción tiene como objetivo acelerar el tiempo de respuesta del control PI dotándolo con la capacidad de predicción del error a partir de, y como su nombre lo indica, la derivada de la curva de error en el tiempo actual.

Como se recordará de los temas del cálculo, la derivada es una razón de cambio puntual para una función $f(x)$, y el valor arrojado al evaluar la derivada en el punto $P(x, f(x))$representa la pendiente de una curva tangente a dicho punto.

Al conocer el dicha pendiente, se puede saber qué tan “rápido” incrementa (valor positivo) o decrementa (valor negativo) los valores de la función $f(x)$.

Aplicando esta idea sobre la curva del error, se busca saber mediante una razón de cambio puntual sobre el error actual, cuál será el comportamiento del error en un momento futuro, si incrementa o decrementa y qué tan rápido lo hace.

Sin embargo, como no se tiene una expresión matemática que represente a la curva del error, se busca realizar una aproximación de primer orden utilizando la series de Tylor, dada por la siguiente expresión.

$$e\left(t + t_d\right) = e(t) + t_d \frac{de(t)}{dt}$$

De acuerdo a la expresión mostrada anteriormente, la representación gráfica de la aproximación sería la siguiente:

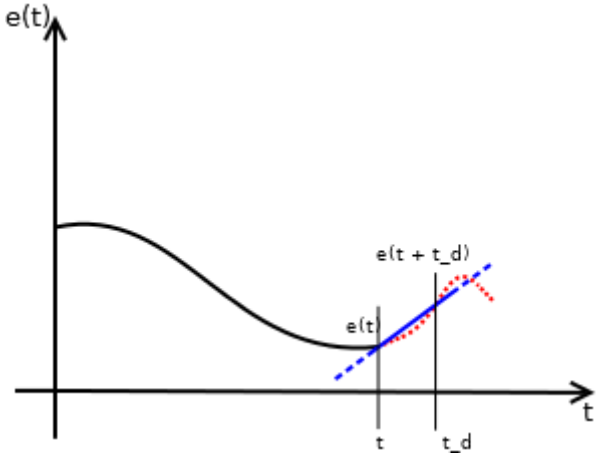


Figura 8.9. Aproximación de la curva de error mediante una recta

Donde el $t$representa al tiempo actual junto con su respectivo error $e(t)$, mientras que el tiempo t_d representa el tiempo futuro y su posible valor de error $e\left(t + t_d\right)$.

El valor de $t_d$al multiplicarse por la relación de una constante de proporcionalidad K_p con respecto a la diferencial de tiempo δ_t, va a representar una ganancia o constante derivativa $K_d$que tiene como objetivo ajustar la aproximación de la curva. Es en este momento donde también se hacen presentes los problemas de este control: Si la constante K_d es un valor muy grande, la aproximación de la curva puede ser errónea; y si por el contrario es un valor muy pequeño, la aproximación puede ser más exacta pero también podría no aportar en nada al resultado de la señal de control.

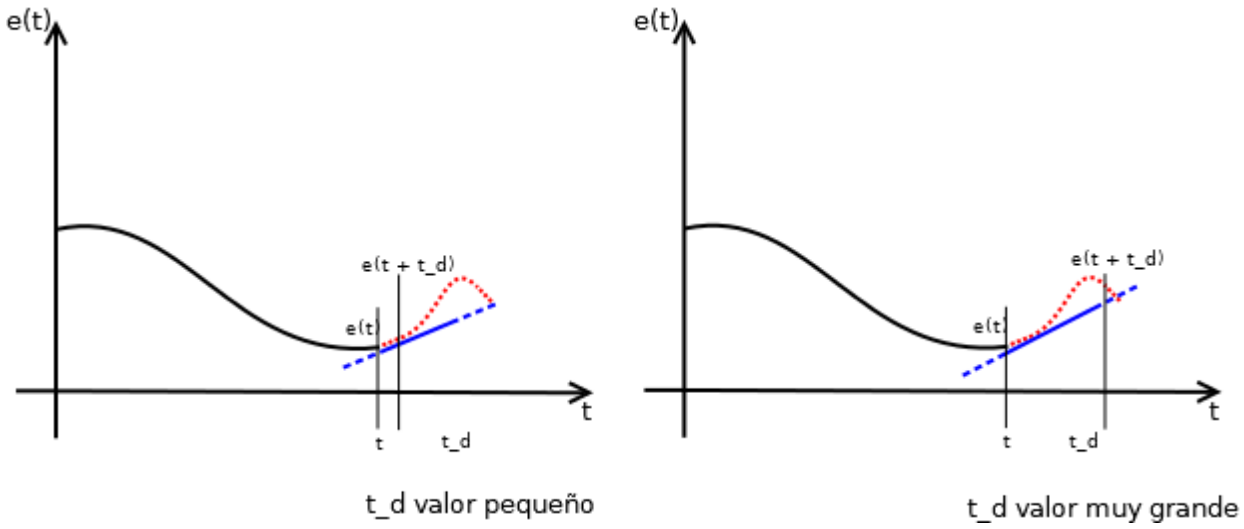


Figura 8.10. Problemas al ajustar la ganancia K_d proporcionada por el valor de t_d

Control PID

El control PID como ya se ha de suponer, se genera de sumar las señales de control de cada controlador; Proporcional, Integral y Derivativo, de modo que se dote al sistema la capacidad de ajustar los valores con los que operará una planta en base a los valores del error generado en un tiempo actual (Presente) más los generados a lo largo del tiempo (Pasado) y más los que se podrían llegar a generar (Futuro).

El diagrama de bloques de este sistema se muestra a continuación:

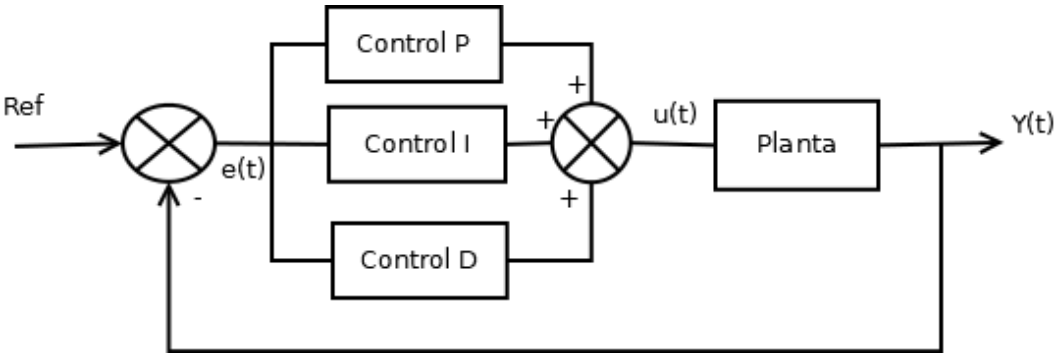


Figura 8.11. Control PID

La expresión matemática de este tipo de control en tiempo discreto para un sistema digital es el siguiente:

$$u_n = u_{(n-1)} + K_p\left(e_n - e_{(n-1)}\right) + K_i\frac{\left(e_n + e_{(n-1)}\right)}{2} + K_d\left(e_n - 2e_{n-1} + e_{n-2}\right)\text{...(x)}$$

donde $K_d = \frac{K_p}{\delta_t} t_d$

Ya con esta última expresión se podría trabajar en el algoritmo con el cual sera posible la aplicación del control PID para la velocidad del robot.

8.2.2 Implementación de las ecuaciones de control PID

La implementación del control de velocidad será sencilla. Para empezar solo hay que tener dos cosas principales en cuenta:

- La expresión del control PID en tiempo discreto obtenida en la sección anterior.
- La forma se va a aplicar la salida que genere el control para que los motores del robot tengan el comportamiento deseado. Dicha forma sera a través de la señal PWM que representa un valor de voltaje con el que se alimenta al motor DC, es decir, la señal de control es un valor para el PWM.

Algoritmo del control PID

Ahora, retomando la expresión x se puede extraer la siguiente información que será de utilidad para diseñar el algoritmo con el que podrá ser posible el control PID:

La ecuación se puede descomponer en cuatro partes independientes a partir de las sumas:

$u_n = u_{(n-1)} + K_p\left(e_n - e_{(n-1)}\right) + K_i\frac{\left(e_n + e_{(n-1)}\right)}{2} + K_d\left(e_n - 2e_{n-1} + e_{n-2}\right)$… expresión original

$u_{(n-1)}$=> representa el valor de la salida anterior generada al final del proceso.

$K_p\left(e_n - e_{(n-1)}\right)$=> representa a la parte del control proporcional

$K_i\frac{\left(e_n + e_{(n-1)}\right)}{2}$=> representa a la parte del control integral

$K_d\left(e_n - 2e_{n-1} + e_{n-2}\right)$=> representa a la parte del control diferencial.

Además se puede apreciar directamente que el valor de error utilizado para calcular cada uno de los tipos de control, así como el valor de la señal de control, son registro de errores y registros de señales de control, respectivamente, generados por el proceso a lo largo del tiempo.

$$e_n, e_{n-1}, e_{n-2}$$
$$u_n, u_{n-1}$$

Se puede considerar entonces el diseño del algoritmo de la siguiente manera:

1. Se calcula el error comparando la velocidad real de la rueda con la velocidad de referencia.
2. El error generado se guarda en un listado de registros de error inicializada en 0 y en cada iteración, el error se deberá de recorrer de posición.

3. Se aplican las operaciones correspondientes de cada control utilizando los valores almacenados en el registro de errores y sus salidas se suman.
4. Utilizando otro registro, pero ahora de señales de control e inicializado en cero, se recorren los valores para actualizar los registros del valor anterior y actual.
5. Se suma el valor que representa al registro anterior con el resultado de la suma del paso 3 y el valor generado por esta operación se almacena en el registro actual.
6. El valor del registro actual se trunca a un valor entero y este entonces representa el valor de la señal de control que se debe de aplicar en el momento actual al motor.
7. Adicionalmente, como este valor entero representa un valor para el PWM, se deberá ajustar el valor de salida de modo que se encuentre siempre dentro de un rango de 0 a 255.

Implementación del algoritmo

Para comenzar, se declaran las variables globales que se ocuparan para llevar a cabo el proceso. Se ha decidido que sean variables globales, ya que este control estará encapsulado en una función que se ejecutara cada vez que el robot cense su velocidad, para que se pueda realizar el ajuste correspondiente en ese preciso momento.

```
// Variables de referencia para el PWM
const int MAX_PWM_VAL = 255;
const int MIN_PWM_VAL = 0;
int PWM_MOTOR = 60;

//Constantes para el control PID
double _kpv = 0.215;  //contante de proporcionalidad
double _kiv = 0.5; //constante de integracion
double _kdv = 0.075;  //contante de derivacion

const double V_REF = 30.0; //referencia

//Historial de valores de control y de error
double _U[2] = {0.0, 0.0};          //registros de la señal de control
double err[3] = {0.0, 0.0, 0.0}; //registros de errores en la señal de salida
```

En el código anterior se tienen declaradas las siguientes variables:

Las constantes enteras que representan los límites superior e inferior de la señal de PWM para el motor (MAX_PWM_VAL, MIN_PWM_VAL). La variable entera que almacena el valor que debe de adoptar la señal PWM al final del proceso en cada iteración. Esta variable debe de estar inicializada con un valor cualquiera, de modo que permita hacer que el motor comience a realizar el giro a una velocidad que pueda ser censada.

Las constantes necesarias para realizar las operaciones que implican cada acción del control PID (_kpv, _kiv, kdv). Estas constantes están inicializadas con valores que para fines prácticos, se obtuvieron de forma empírica, teniendo en cuenta el efecto que pueden provocar sobre el sistema y que fueron comentados en la sección anterior.

El valor de referencia (V_REF) que es un dato de tipo double y que representa la velocidad angular en r/s al que se desea que gire la rueda todo el tiempo. Esta variable puede contener cualquier valor, siempre y cuando no sea exageradamente alto ya que debido a las limitaciones físicas que contienen los elementos del hardware del robot,

podrían generarse dificultades para que el sistema se aproxime a la referencia y ademas implicaría un sobre-esfuerzo que provoque daños irreversibles en los componentes.

Por ultimo se encuentran los historiales de los valores que van tomando tanto la señal de control, como la señal de error en cada iteración del proceso a lo largo del tiempo. Estos historiales están declarados como arreglos unidimensionales que van a almacenar 2 y 3 datos respectivamente e inicializados en ceros.

El arreglo _U[2] sirve para almacenar el valor del control actual y el anterior. El arreglo err[3] almacena el valor del error en el tiempo actual, el error en el tiempo anterior y el error en el tiempo anterior al anterior, que son los datos que se necesitan para satisfacer a la ecuación X.

Función controlPID_X()

La función controlPID_X()[8] tiene como objetivo aplicar la ecuación de control de la expresión (x) y generar la señal de control que se debe de aplicar al motor DC. Para ello deberá de recibir como entradas la velocidad angular de la rueda que ha sido medida por el encoder, así como también el valor de referencia que se desea alcanzar.

Esta función también retorna un valor entero que representa la señal de control generada y que modifica a la señal PWM que alimenta al motor DC de la rueda deseada.

```
1 const double VEL_DES = 30.0;
2
3 int controlPID_X(double v_med, double v_ref){
4   //Codigo
5}
6
```

Nuevamente se cuenta que las velocidades aquí manejadas se expresan en radianes sobre segundo (r/s) y son valores decimales, por eso se declaran como double.

La variable v_med representa a la velocidad angular con la que gira realmente la rueda del robot, mientras que v_ref representa la velocidad angular de referencia a la que se desea que gire todo el tiempo.

De acuerdo a los dos primeros pasos descritos en el algoritmo, se calcula el error comparando la referencia con el valor de la velocidad medida. Este error se debe de asignar a la primer posición (0) del arreglo err[], pero antes los valores ya almacenados se deben de recorrer un lugar, tal y como lo muestra el siguiente fragmento de código.

```
1  err[2] = errLeft[1];
2  err[1] = errLeft[0];
3  err[0] = v_ref - v_med;
```

[8] Se llama controlPID_X porque se aplica a cualquiera de los motores, izquierdo o derecho. Esta función sirve para explicar el proceso en general. Para su correcta aplicación sobre el robot como tal, solo se necesitaría repetir el proceso para cada rueda pero con sus propias variables de control en cada caso.

Ahora, teniendo ya estos valores se pueden aplicar directamente cada una de las operaciones que componen a la expresión de control, y que representan las acciones Proporcional, Integral y Derivativa.

```
1  double proporcional = _kpv * (err[0] - err[1]);
2  double integral = _kiv * ((err[0] + err[1]) / 2);
3  double diferencial = _kdv * (err[0] - (2 * err[1]) + err[2]);
```

Siguiendo los pasos 4 y 5 del algoritmo, se recorren una posición los valores almacenados en el arreglo _U[]. Teniendo ya los nuevos valores en el arreglo, ahora sí se complementan todas las operaciones que implica la ecuación de control y el resultado se almacena en la primer posición (0) del arreglo _U[].

```
1  _U[1] = _U[0];
2  _U[0] = _U[1] + (proporcional + integral + diferencial);
```

Cómo el valor almacenado en _U[0] es el valor de la señal de control actual, primero se trunca a un valor entero y se verifica mediante la estructura de decisión if, si el valor generado se encuentra dentro de los límites establecidos, y si no es así, lo redefine según sea el caso.

```
1  _U[0] = (int)_U[0];
2
3  if (_Ri[0] > MAX_PWM_VAL)
4    _Ri[0] = MAX_PWM_VAL;
5
6  if (_Ri[0] < MIN_PWM_VAL)
7    _Ri[0] = MIN_PWM_VAL;
```

Finalmente el valor de _U[0] ya generado y verificado es retornado por la función para ser aplicado sobre el motor.

```
1  return _U[0];
```

Aplicación de la función controlPID_X()

Finalmente, para integrar al funcionamiento del robot el control PID de la función desarrollada anteriormente, solamente se tiene que invocar justo después de calcular la velocidad angular de la rueda. El resultado que retorne la función debe de afectar directamente a la variable global PWM_MOTOR para que en una siguiente iteración, el motor realice su giro de acuerdo al nuevo valor que contenga el PWM.

```
1  while(/*condicion*/){
2   _sample_time = millis();
3   _delta_time = (double)_sample_time - _last_sample_time;
4
5   digitalWrite(D_M, /*DIRECTION*/);
6   analogWrite(V_M, PWM_MOTOR);
7
8   if (_delta_time >= TIEMPO_MUESTREO)
9   {
10    //Extraccion de las velocidades
11    double vel_med = get_w();
12
13    //aplicacion del control de velocidad en las ruedas
14    PWM_MOTOR = controlPID_X(vel_med, vel_ref);
15
16    _last_sample_time = _sample_time;
17  }
18 }
```

El código anterior repite prácticamente las instrucciones utilizadas para el cálculo de velocidades desarrolladas en el capitulo anterior. Lo único diferente aquí es la instrucción en la línea 14, donde se invoca a la función encargada de realizar el control PID y generar el nuevo valor para el PWM del motor que es asignado a la variable PWM_MOTOR.

Así, cuando el bucle while realiza una nueva iteración, el nuevo valor de PWM es aplicado en el motor, como lo indica la línea 6. Todo este proceso se repetirá mientras se cumpla con la condición declarada en la línea 1.

A continuación presento dos gráficos que demuestran el comportamiento de la velocidad de los motores primero sin el control aplicado (Figura 8.x)[9] y después con el control PID (figura 8.x).

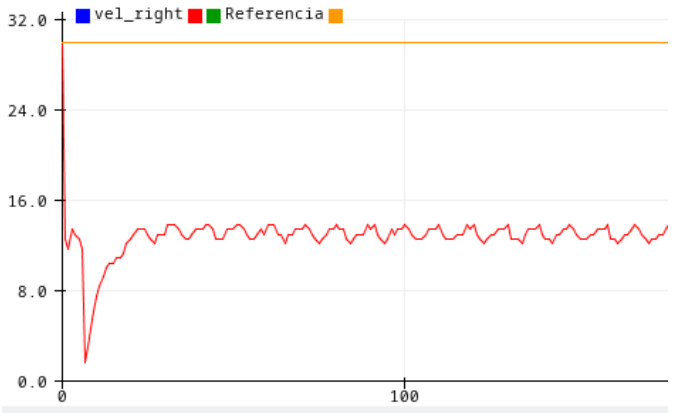


Figura 8.12. Gráfica de la velocidad en rueda derecha sin control aplicado

[9] En el gráfico se nota una curva que desciende desde un valor alto hacia un valor aproximado a cero y después vuelve a subir en las primeras porciones de tiempo; esto se debe a un comportamiento que se presenta cuando se carga el programa en arduino. Comienza la ejecución de la acción y ya se van registrando valores, y cuando se abre el serial plotter donde muestra la gráfica, se reinician los datos y se vuelve a graficar. Solamente hay que ignorar esa porción.

En la figura 8.x se muestra el comportamiento de la velocidad de la rueda derecha sin control aplicado.

En este ejercicio se definió una referencia de 30 r/s y en el grafico se esta representada por la recta que se encuentra casi hasta arriba. Para la velocidad de la rueda, se definió un valor constante para la señal de PWM que es de 60. Con este valor la velocidad angular oscila (demasiado) a un valor aproximado de 14 r/s.

Como se puede notar, la velocidad de la rueda es inestable, aun cuando la rueda no está friccionando con alguna superficie, y además no se aproxima a la velocidad deseada.

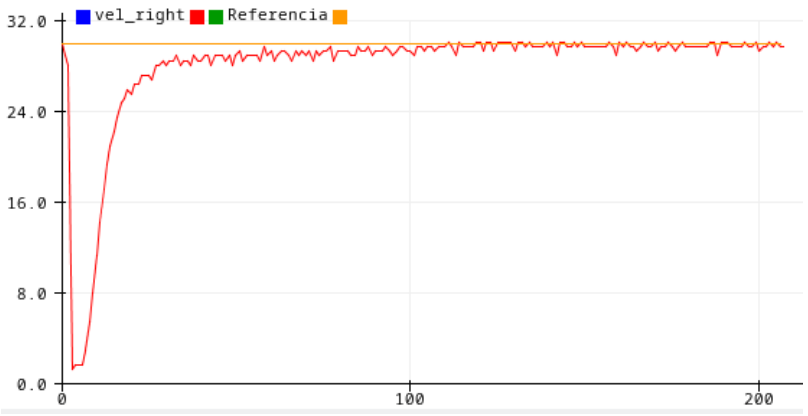


Figura 8.13. Gráfica de velocidad de la rueda derecha con el control PID aplicado.

En la gráfica de la figura 8.x[10] se muestra ahora el comportamiento del control PID aplicado. Como se puede notar, en esta ocasión, la velocidad de la rueda comienza a incrementar rápidamente hasta mantenerse con bajas oscilaciones a un valor muy parecido al de la referencia.

Para la correcta aplicación del control de velocidad en el robot, se necesitaría ahora aplicar la misma función de control pero ahora en ambas ruedas como lo muestra el siguiente diagrama de bloques.

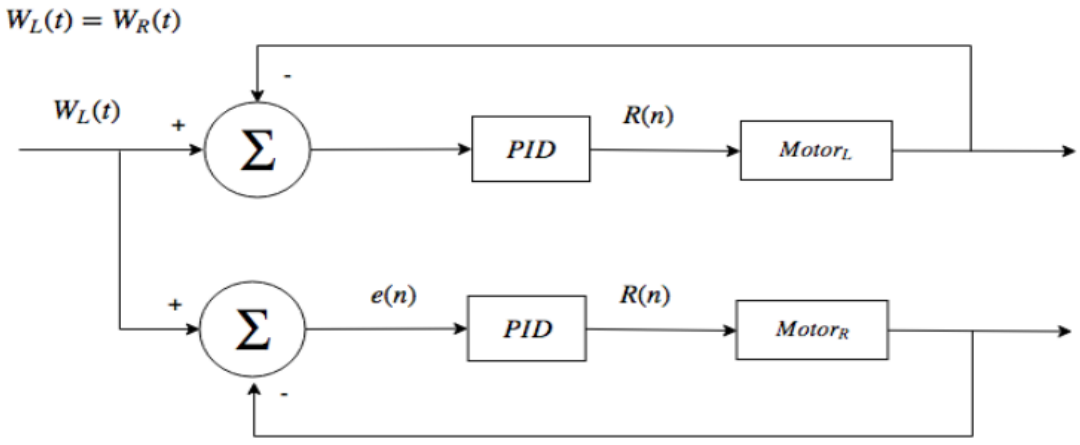


Figura 8.14. Control PID en ambas ruedas utilizando la misma velocidad de referencia

[10] En esta gráfica se presenta una situación igual a la de la grafica en la figura 8.x

El resultado de dicha implementación es el siguiente:

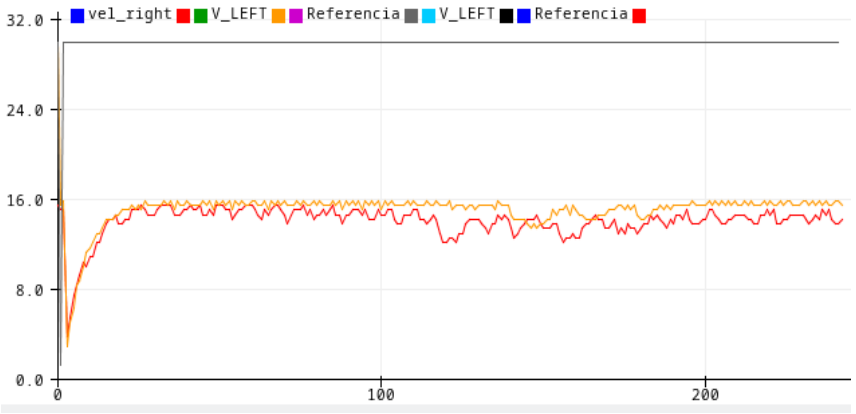


Figura 8.15. Gráfico del comportamiento de las velocidades en las ruedas sin un control aplicado

En esta nueva gráfica (8.x), se demuestra la forma en la que la velocidad de ambas ruedas difiere demasiado a pesar de que ambas tienen el mismo valor de PWM. además entre el valor 100 y 200 de la variable independiente se ve una caída abrupta del valor en una de las ruedas, lo cual representa una perturbación que provoca que la rueda se atasque y reduzca su velocidad dando paso entonces a que se presenten algunos de los problemas descritos al inicio de este capítulo.

Cuando el PID se aplica en ambas ruedas se mejora demasiado el comportamiento, pues ahora ya no solo ambas velocidades se emparejan y se ajustan al valor deseado, sino que se crea una resistencia a las perturbaciones, pues en cuanto se hacen presentes, el sistema de control actúa en automático y trata de corregir el error generado por ellas, como se aprecia en la siguiente gráfica.

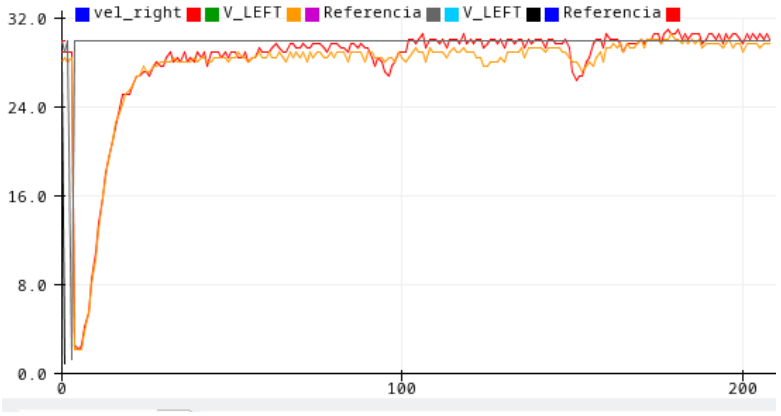


Figura 8.16. Gráfico del comportamiento de las velocidades en las ruedas con el control PID aplicado

En la figura anterior se puede observar que la velocidad de ambas ruedas se aproxima casi al mismo tiempo a la referencia. Se reduce mucho la diferencia de velocidades entre ambas ruedas, además entre los valores 100 y 200 de la variable

independiente, se añaden perturbaciones pero inmediatamente el sistema trata de ajustar los valores del PWM de modo que las velocidades se mantengan cerca de la referencia.

8.3 implementación completa del control de velocidad para la trayectoria del robot

En esta última sección del capitulo solo quisiera abordar la idea de cómo se debe de implementar completamente el control de velocidad en el robot.

Hasta el momento la programación presentada ha tenido valor de velocidad de referencia definido directamente en el código, por lo que seria imposible cambiar esa referencia a menos que se detenga el funcionamiento, se redefina el valor y se vuelva a compilar y ejecutar. Lo ideal seria que se le pueda indicar en cualquier momento al robot que velocidades angular y lineal debe de adoptar para realizar su recorrido.

Para solucionar este problema solo se requiere de hacer uso de la forma matricial de las velocidades de las ruedas vista en el capitulo 6 (cinemática del robot móvil) que se muestra a continuación:

$$\begin{bmatrix} \omega_R & \omega_L \end{bmatrix} = \frac{1}{(2\pi r)} \begin{bmatrix} 1 & -\left(\frac{d}{2}\right) \\ 1 & \frac{d}{2} \end{bmatrix} \begin{bmatrix} V_{Robot} & \omega_{Robot} \end{bmatrix}$$

Donde:

V_{Robot} es la velocidad lineal deseada

ω_{Robot} es la velocidad angular deseada

ω_R, ω_L son las velocidades angulares que debe de adoptar la rueda.

Entonces la acción de control completa sigue el siguiente diagrama de bloques:

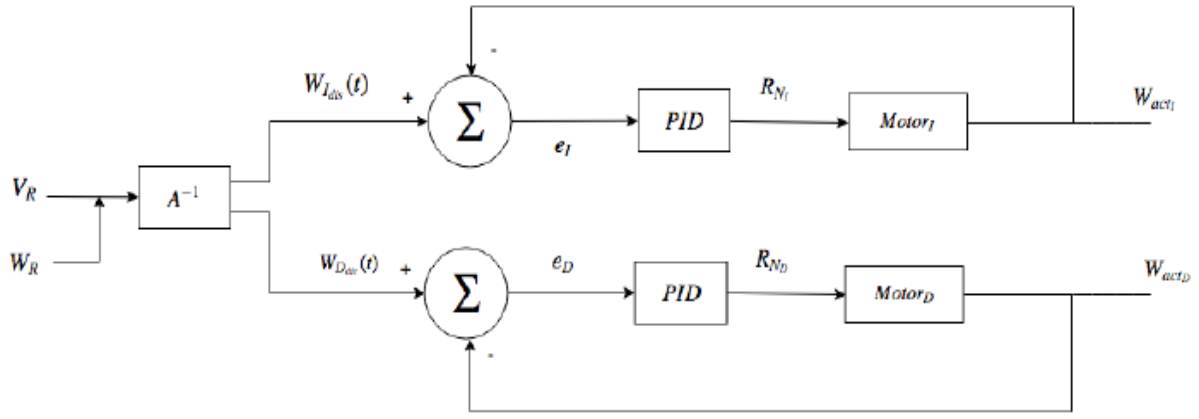


Figura 8.17. diagrama de bloques para representar la acción de control a partir de las velocidades Lineal y Angular

En este diagrama, antes del control PID para las velocidades se ejecuta una operación que recibe como entrada los valores de V_{Robot} y ω_{Robot} y devuelve como salida los

valores de las velocidades angulares de referencia que deben de adoptar cada una de las ruedas para lograr el comportamiento deseado.

Como nota adicional, se recuerda que esta operación no se recomienda realizarla dentro del programa de Arduino, pues hay que recordar que el objetivo de esta placa según lo descrito en el capítulo 3 (Componentes del robot I: Unidades de control e inteligencia), solo manejara las señales de entrada y salida de los sensores y actuadores del robot.

En el caso del control PID, como es una acción que afecta directamente a las señales de salida para los actuadores del robot, si se ejecuta directamente aquí.

La operación para el cálculo de velocidades de referencia se puede definir en un programa que ejecute directamente la placa Raspberry PI y el resultado de esta operación simplemente la envié a la placa Arduino para que se ejecute la acción.

Capítulo 9 Comportamiento del Robot

A partir de este capítulo comienzo a cubrir la cuarta sección de la tesis descrita en la introducción y que está enfocada en cumplir con el tercer objetivo particular del trabajo: El robot posee la capacidad de ejercer sus tareas de manera autónoma, es decir, que no requiera de la intervención humana para realizar todas sus actividades.

En este capítulo se llevará a cabo el análisis sobre las acciones que debe de tomar el robot al momento de navegar por un entorno lleno de obstáculos para finalmente lograr su objetivo que es alcanzar una meta señalizada por una fuente luminosa.

El comportamiento del robot se desarrollará de acuerdo a un modelo que le permita tomar decisiones sobre las acciones que debe de realizar, en base a la información que capta del medio.

9.1 Robot Seguidor de Luz y evasor de Obstáculos

Tal y como el nombre lo indica, se plantea la idea de que el robot sea capaz de realizar las siguientes actividades fundamentales:

- Identificar una fuente luminosa ubicada en cualquier dirección del medio en el que operará el robot.
- Realizar su desplazamiento por el medio de modo que consiga alcanzar la fuente luminosa
- identificar cuando se encuentre frente a posibles obstáculos que le impidan lograr su objetivo, de modo que pueda modificar su comportamiento para que evada el obstáculo y continúe con su navegación.

De acuerdo con el primer punto, el comportamiento inicial que debe de tener el robot es el de ‘saber’ identificar dónde está la fuente luminosa a la que debe de seguir. Así, este tendría que procesar la información que reciba de sus sensores encargados de detectar señales luminosas (sensores LDR), para determinar la dirección en la que se encuentra su objetivo.

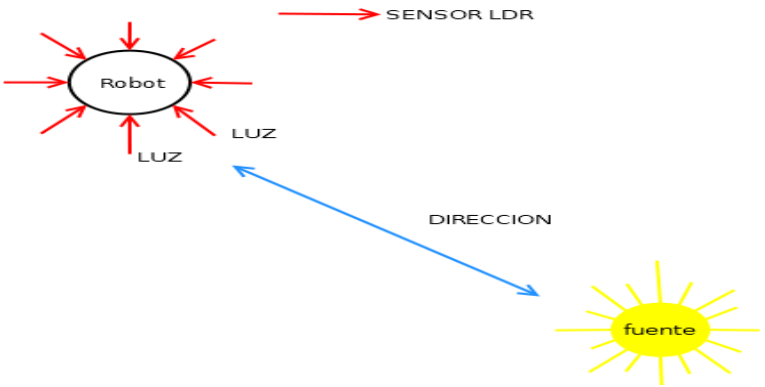


Figura 9.1. identificación de la fuente luminosa

En la figura 9.1, se representa un escenario donde el robot se encuentra estático en una posición inicial mientras captura señales de luz con sus sensores LDR representados por las flechas que apuntan hacia él. Hay una fuente de luz en

la esquina inferior derecha desde la cual provienen las señales luminosas captadas por dos de los sensores, por lo que el robot ahora puede conocer la dirección en donde está su objetivo.

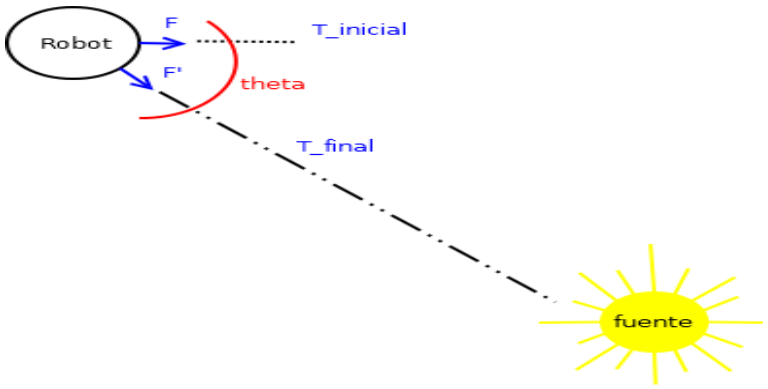


Figura 9.2. planeación de movimientos

En el segundo punto se plantea que si el robot ya ha logrado identificar el objetivo, entonces debe de proceder a realizar su navegación. Para ello deberá de 'planear' de algún modo sus movimientos para que empiece a moverse hacia la dirección calculada y avance hasta alcanzar su objetivo.

En la figura 9.2, el robot que ahora ya conoce la dirección de la fuente luminosa debe de realizar un giro de θ grados a la derecha para posicionar su punto frontal (F) en una nueva dirección (F') de modo que "vea" de frente a la fuente luminosa, y que para alcanzarla deberá de avanzar en línea recta X cantidad de metros a través de la trayectoria T_{final}.

En el tercer y último punto se tiene en cuenta que en la trayectoria sobre la que el robot avanzará para alcanzar la meta se pueden presentar obstáculos (quizá de forma aleatoria) que impidan su navegación, por lo que el robot deberá de tener la capacidad de identificarlos a través de sus sensores encargados para esa tarea (ultra-sonicos, láser, de contacto), y en base a esa información, decidir de qué forma esquivará el obstáculo para poder continuar con su navegación.

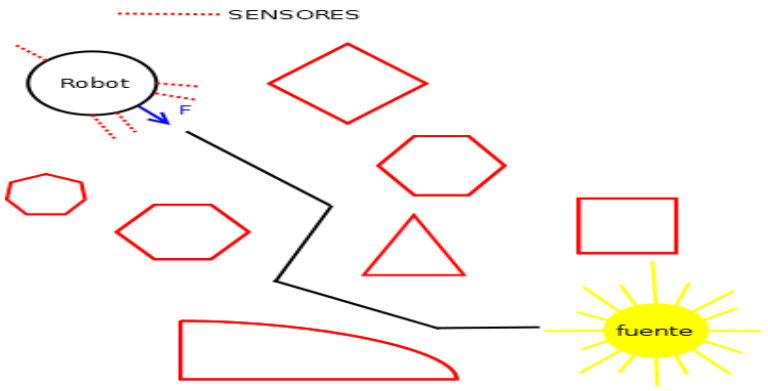


Figura 9.3. Evasión de obstáculos

En la figura 9.3 se representa el escenario completo al que se enfrentará el robot al momento de realizar sus tareas.

Una vez que el robot ya conoce la dirección de la fuente luminosa que representa la meta a alcanzar, deberá de comenzar su navegación en dirección a dicha fuente, pero censando en todo momento para conocer si se encuentran obstáculos en su camino. Si sus sensores (que están representados con la línea punteada) detectan obstáculos enfrente, atrás, a la izquierda o a la derecha, el robot deberá de decidir si avanza o retrocede X cantidad de metros, gira θ grados a la derecha o izquierda, de modo que consiga lograr una trayectoria que le permita llegar a su meta pasando a través de esos obstáculos (como la trayectoria descrita por la línea que une al robot con la fuente).

9.2 El modelo reactivo

Para diseñar el comportamiento que deberá de tener el robot al momento de enfrentarse a su entorno de trabajo de acuerdo a lo planteado anteriormente, se ha elegido seguir el modelo de comportamiento reactivo.

El modelo de comportamiento reactivo contiene una serie de ventajas que permiten diseñar de una forma más sencilla múltiples funcionamientos en el robot, de modo que le brinden mayor adaptabilidad al entorno, aún si este es dinámico o se presentan errores durante el censado.

Las características de este modelo son:

a) Basado en el comportamiento de los insectos. Este modelo trata de imitar el comportamiento natural de entidades vivas quienes realizan sus acciones de acuerdo a la información que perciben del medio.

b) No requiere de una representación del entorno ni de planeación de acciones o movimientos. A diferencia de los modelos tradicionales que se basan en métodos de búsqueda y que si requieren información actualizada y completa del entorno que van a explorar; además de las posibles acciones que pueden realizar en él.

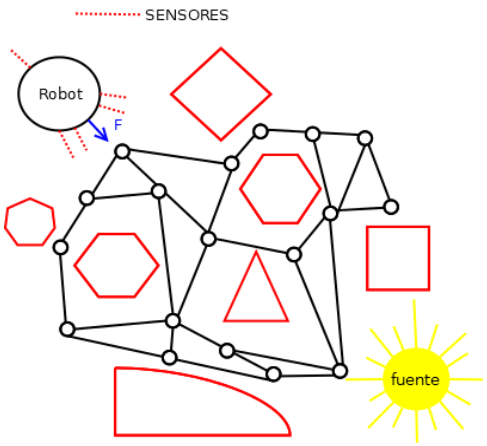


Figura 9.4 a. representación del medio mediante grafos

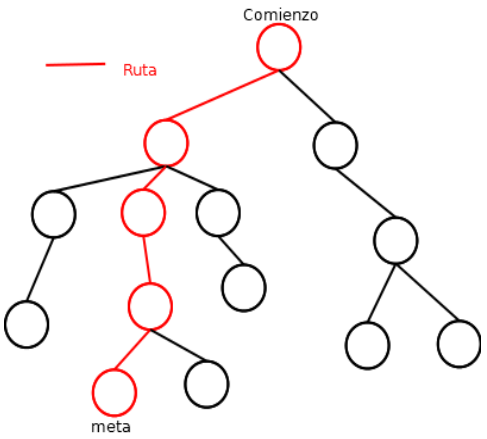


Figura 9.4 b. cálculo de rutas mediante arboles de búsqueda

c) Se adapta mejor a los entornos dinámicos y con errores en el censado.

d) Se pueden tener múltiples comportamientos funcionando en paralelo, donde cada comportamiento es independiente y se representan mediante un diagrama estímulo-respuesta (ER).

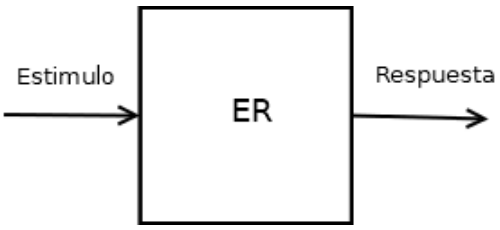


Figura 9.5. Diagrama Estímulo-Respuesta (ER)

e) Cada módulo ER recibe un estímulo de entrada proveniente de los sensores del robot y la salida generada define un comportamiento o acción que deberá ejecutar el actuador. Estas salidas usualmente se suman o se evalúan mediante un árbitro quien decide qué acción se ejecuta.

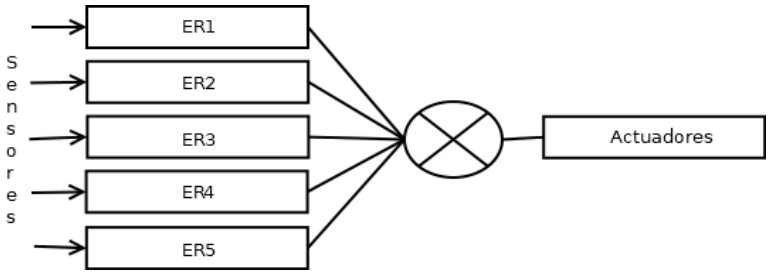


Figura 9.6 a. Suma de salidas

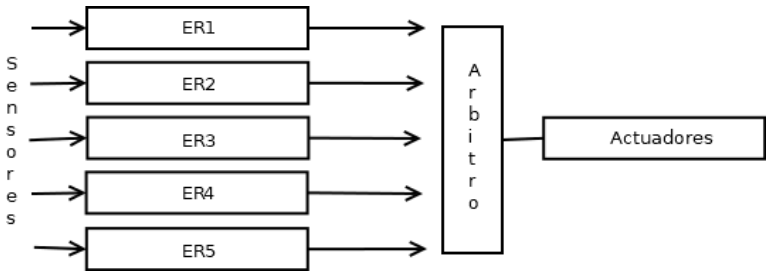


Figura 9.6 b. Arbitraje de salidas

f) Los comportamientos reactivos se suelen diseñar utilizando lógica de orden cero, máquinas de estado finitas, máquinas de estado finita aumentadas, campos potenciales o redes neuronales.

De acuerdo a la información expuesta anteriormente, se procederá a diseñar los comportamientos que deberá de tener el robot para conseguir sus objetivos. Estos comportamientos se definirán a partir de máquinas de estado finitas aumentadas, que se describen en la sección siguiente.

9.3 Descripción del comportamiento mediante máquina de estados ASM

El comportamiento reactivo del robot se puede modelar fácilmente mediante una máquina de estados finitos (FSM por sus siglas en inglés).

La FSM se puede entender simplemente como un sistema cuyo comportamiento está definido por un conjunto finito de estados que se ejecutan durante un tiempo T y que generan salidas en función de las entradas que reciben y del estado actual de ejecución.

Estos sistemas FSM pueden ejecutar algoritmos como una computadora y para el contexto del comportamiento reactivo del robot, pueden representar las acciones o comportamientos de este durante un tiempo t, hasta que recibe nuevas entradas o estímulos en su estado actual (que podrían ser señales captadas en los sensores de obstáculos) y hacer que el robot ejecute una nueva acción (respuesta) en el tiempo t+1

Por ejemplo, se piensa en la máquina de estados que define el comportamiento del robot cuando se encuentra cerca o lejos de una fuente luminosa. La siguiente máquina de estados representada mediante una carta ASM describe ese comportamiento.

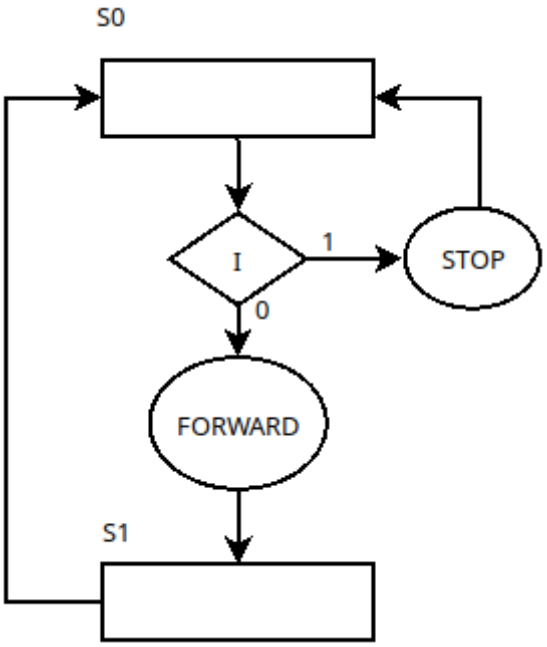


Figura 9.7. FSM que define si el robot ya está en la meta

De acuerdo a la máquina de estados, el robot tiene dos posibles estados y dos salidas condicionales. En el estado inicial S0, el robot simplemente puede estar encendido esperando algún estímulo que le permita cambiar a un nuevo estado S1.

Durante la transición del estado inicial S0 a S1, el robot recibe un estímulo a través de sus sensores de luz que le indican el nivel de intensidad luminosa (I) en su ambiente y con ese valor obtenido el robot define si se encuentra cerca o muy cerca de la fuente luminosa y asume que ya alcanzó su meta, por lo que se genera la salida condicional 1, en donde se le ordena al robot detenerse, o por el contrario, está lejos o muy lejos de dicha fuente por lo que se genera la salida condicional 0 y el robot comienza a avanzar para lograr acercarse a dicha meta.

El estado S1 para este ejemplo es genérico, pues puede representar cualquier otra acción que debe de realizar el robot durante su avance, por ejemplo, buscar la dirección de la meta, censar si hay obstáculos presentes en el camino, ejecutar movimientos para esquivar obstáculos, etc.

Cada uno de los comportamientos descritos en el párrafo anterior se pueden representar mediante máquinas de estado finitas tal y como se mostró en el ejemplo. Sin

embargo, como todos estos comportamientos pertenecen al mismo robot se puede pensar de inmediato que estas máquinas de estados deben de estar conectadas entre sí de modo que las salidas que generan unas puedan tomar el rol de entrada para otras, para que se pueda describir en una máquina de estados mucho más grande, el comportamiento general del robot.

Esta idea nos lleva a explorar un nuevo concepto que nos apoyará en el diseño del comportamiento reactivo para este robot.

Máquinas de Estado Finitas Extendidas (AFSM)

Las AFSM o máquinas de estado finitos aumentadas creadas por Rodney Brooks en los años 80's, son una evolución de las máquinas de estado finitas (descritas antes), en donde se tiene un conjunto de FSM conectadas entre sí y cuyas salidas generadas por unas funcionan como entradas para otras; las salidas de cada máquina también cuentan con supresores que las pueden bloquear y colocar otros valores generados por otras máquinas. Estas máquinas también cuentan con una entrada de reset que les permite regresar a un estado en específico general. Su representación gráfica sería como la siguiente:

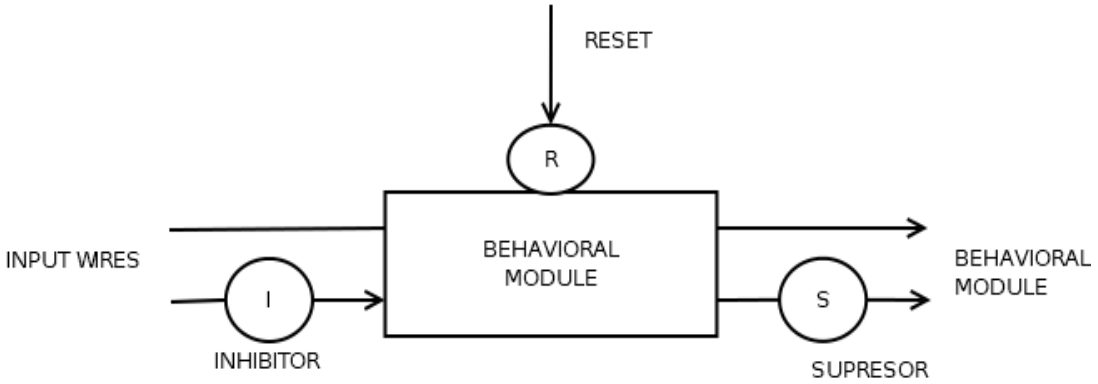


Figura 9.8. Representación de una AFSM

Los inhibidores y supresores conectados a las entradas y salidas respectivamente de la máquina de estados se pueden ver como Circuitos lógicos multiplexores, en donde dichos multiplexores eligen qué valor de entrada recibe el módulo y que valor de salida se envía.

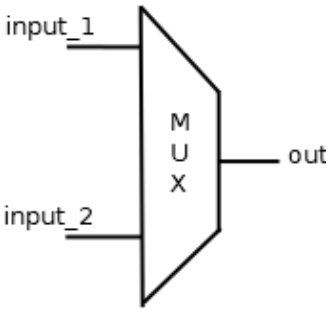


Figura 9.9. Supresores e inhibidores representados como Multiplexores

La mayor ventaja que tiene el uso de máquinas de estado finitas extendidas es que se puede generar una estructura jerárquica entre los comportamientos de modo que aquellos que tienen mayor orden en la jerarquía puedan influir sobre las entradas y salidas que tienen otros comportamientos de menor orden.

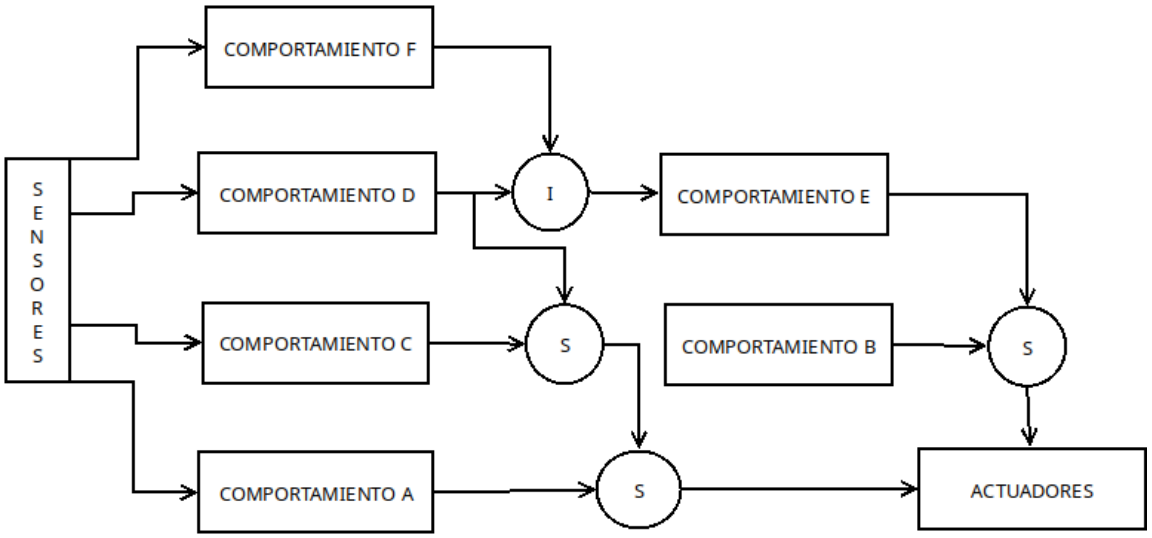


Figura 9.10. Estructura jerárquica de una AFSM

A continuación se muestra una propuesta de cómo sería el comportamiento del robot seguidor de luz y evasor de obstáculos representado mediante una AFSM.

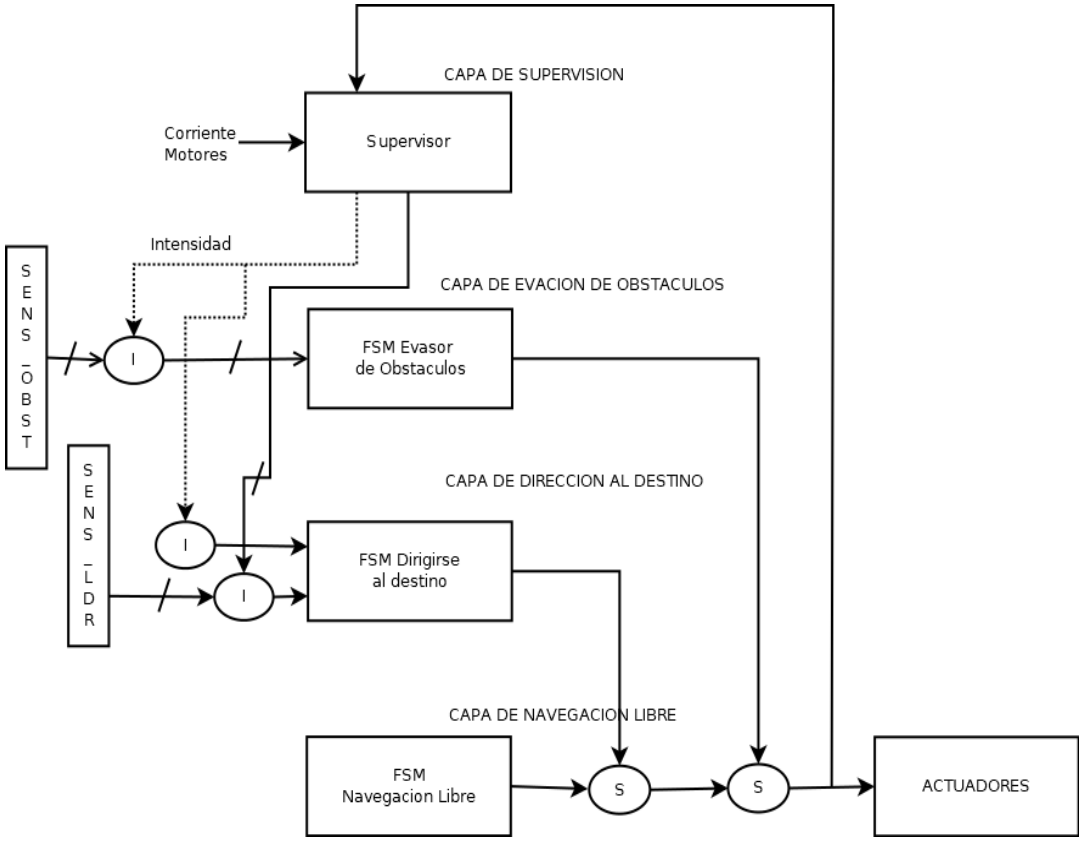


Figura 9.11. Comportamiento del robot seguidor de luz y evasor mediante AFSM

En la figura 9.x se muestra como sería en general la máquina de estados finita extendida para el robot. En este diseño se especifican 4 capas que componen a la máquina de estados: Supervisión, Evasión, Dirección, Navegacion.

Capa de supervisión

Esta primera capa contiene una máquina de estados que se encarga de comprobar dos cosas: La intensidad luminosa de la fuente y la dirección en la que se encuentra.

El primer paso que debería de realizar esta máquina de estados es comprobar si el robot ya está en la meta; esto se logra a partir de la lectura de los valores de la intensidad luminosa recolectados por los sensores LDR del robot. Si esta intensidad captada ya alcanza o supera un umbral que representa la máxima intensidad de luz que emite la fuente, entonces se concluye que el robot ya está en la meta por lo que sería un caso verdadero (1) y la salida condicional sería una señal de STOP para el robot.

Si por el contrario, la intensidad de luz es menor al umbral entonces se concluye que el robot se encuentra lejos de la meta y se genera un caso negativo (0), donde ahora se deberá de comparar los valores de los sensores LDR y aquel que posea la intensidad luminosa más alta será el que defina cuál es la posible dirección en la que se encuentra la fuente luminosa, por lo que se genera una nueva salida condicional que le especifica al robot la dirección a la que debe de dirigirse para alcanzar la meta.

La máquina de estados sería como se muestra en el siguiente diagrama.

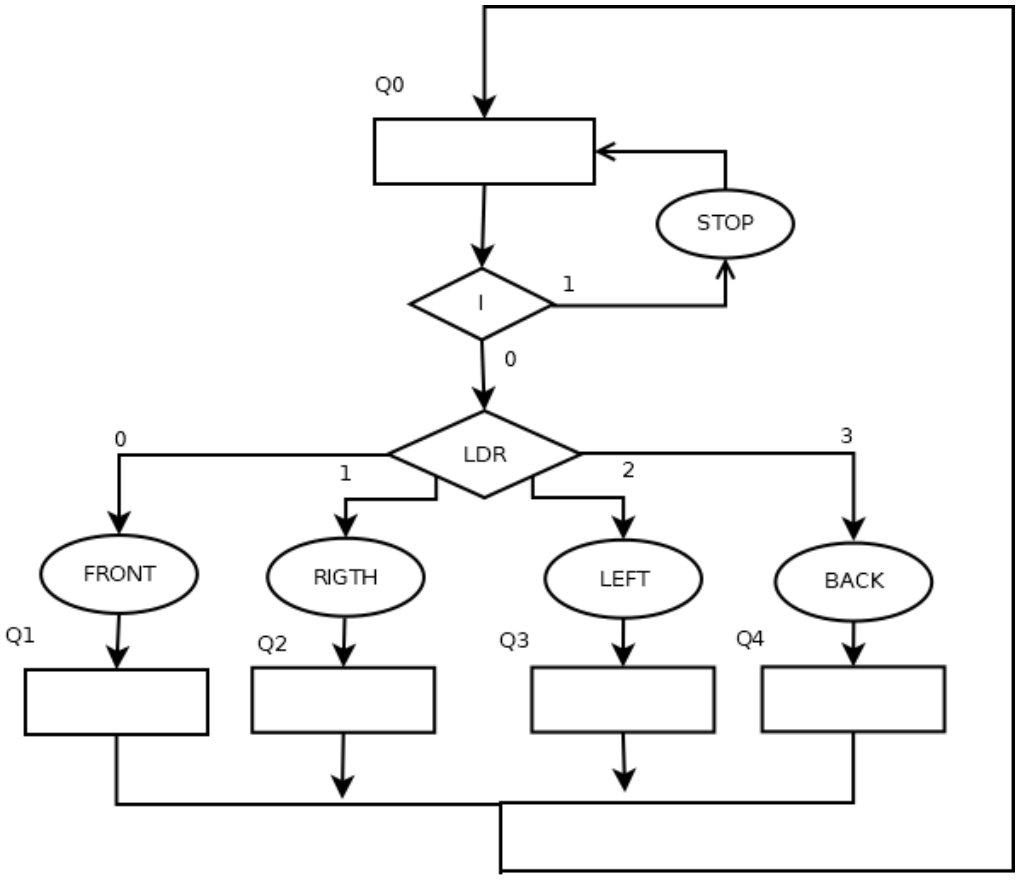


Figura 9.12. Capa de supervisión

Para esta máquina de estados el estado inicial Q0 representa, por ejemplo, el momento en que el robot enciende e inmediatamente comienza a evaluar la intensidad de luz en el ambiente para localizar la meta. Si la condición I se cumple, entonces el robot se mantiene detenido todo el tiempo, y si no es así, el robot comienza a buscar la meta y a generar las salidas condicionales.

Los estados Q1, Q2, Q3 y Q4 son representativos. Estos estados en realidad deberían de definirse en las capas siguientes a partir de las salidas condicionales de esta máquina. También es posible extender esas 4 salidas y cuatro estados a más, ya que si se considera que el robot puede buscar la fuente de luz en 8 direcciones por ejemplo, entonces también se tendrán 8 salidas y 8 estados. Al final de todo el proceso, se indica que el robot regresa a su estado inicial para volver a comprobar la ubicación de la meta y detenerse o repetir el proceso otra vez.

Capa de evasión de obstáculos.

En esta segunda capa se llevan a cabo las tareas que le permiten al robot comprobar si existen o no obstáculos en su camino una vez que ya identificó la dirección de la que proviene la luz de la fuente luminosa. Los obstáculos son detectados mediante la lectura de los valores leídos por sus sensores Ultrasonicos, Infrarrojos y de contacto.
Para entender cómo se podría diseñar la máquina de estados que describe el comportamiento del robot para detectar y evadir obstáculos, muestro a continuación los posibles casos que se podrían presentar.

CASO 1: Obstáculo al frente.

Para saber si el robot está frente a un objeto que obstruya su navegación libre, solo basta con verificar la distancia que ha detectado el sensor ultrasónico frontal como lo muestra la figura 9.x. Si esa distancia es muy corta, entonces el robot toma acciones para evadirlo. De igual manera, si este sensor falla, entonces tiene que comprobar que tanto el sensor de contacto derecho como el izquierdo arrojen una señal de HIGH para saber si ha chocado con algún objeto al frente.

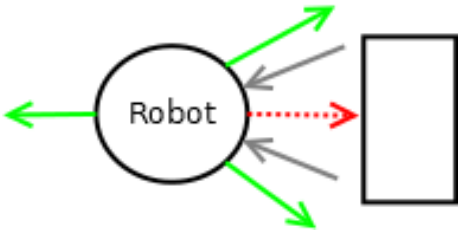


Figura 9.13-a. Obstáculo al frente

Para que el robot pueda evadir un obstáculo al frente este se detiene y comprueba si no existe un objeto detrás de él, si es así, entonces retrocede un poco, gira 90 grados a la derecha y continúa con su avance; si no, entonces solo gira 90 grados a la derecha y continúa con su avance.

CASO 2: Obstáculo a la derecha.

Para saber si el robot tiene a su derecha un objeto que obstruya su navegación libre, solo basta con verificar la distancia que ha detectado el sensor infrarrojo derecho como lo muestra la figura 9.x. Si esa distancia es muy corta, entonces el robot toma acciones para evadirlo. De igual manera, si este sensor falla, entonces tiene que comprobar que el sensor de contacto derecho arroje una señal de HIGH para saber si ha chocado con algún objeto a su derecha.

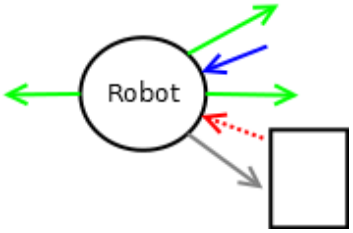


Figura 9.13-b. Obstáculo a la derecha

Para que el robot pueda evadir un obstáculo a la derecha este se detiene y comprueba si no existe un objeto detrás de él, si es así, entonces retrocede un poco, gira 45 grados a la izquierda y continúa con su avance; si no, entonces solo gira 45 grados a la izquierda y continúa con su avance.

CASO 3: Obstáculo a la izquierda.

Para saber si el robot tiene a su izquierda un objeto que obstruya su navegación libre, solo basta con verificar la distancia que ha detectado el sensor infrarrojo izquierdo como lo muestra la figura 9.x. Si esa distancia es muy corta, entonces el robot toma acciones para evadirlo. De igual manera, si este sensor falla, entonces tiene que comprobar que el sensor de contacto izquierdo arroje una señal de HIGH para saber si ha chocado con algún objeto a su izquierda

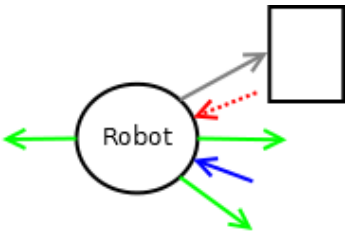


Figura 9.13-c. Obstáculo a la izquierda

Para que el robot pueda evadir un obstáculo a la izquierda este se detiene y comprueba si no existe un objeto detrás de él, si es así, entonces retrocede un poco, gira 45 grados a la derecha y continúa con su avance; si no, entonces solo gira 45 grados a la derecha y continúa con su avance.

La FSM final que describe todos estos comportamientos se muestra a continuación.

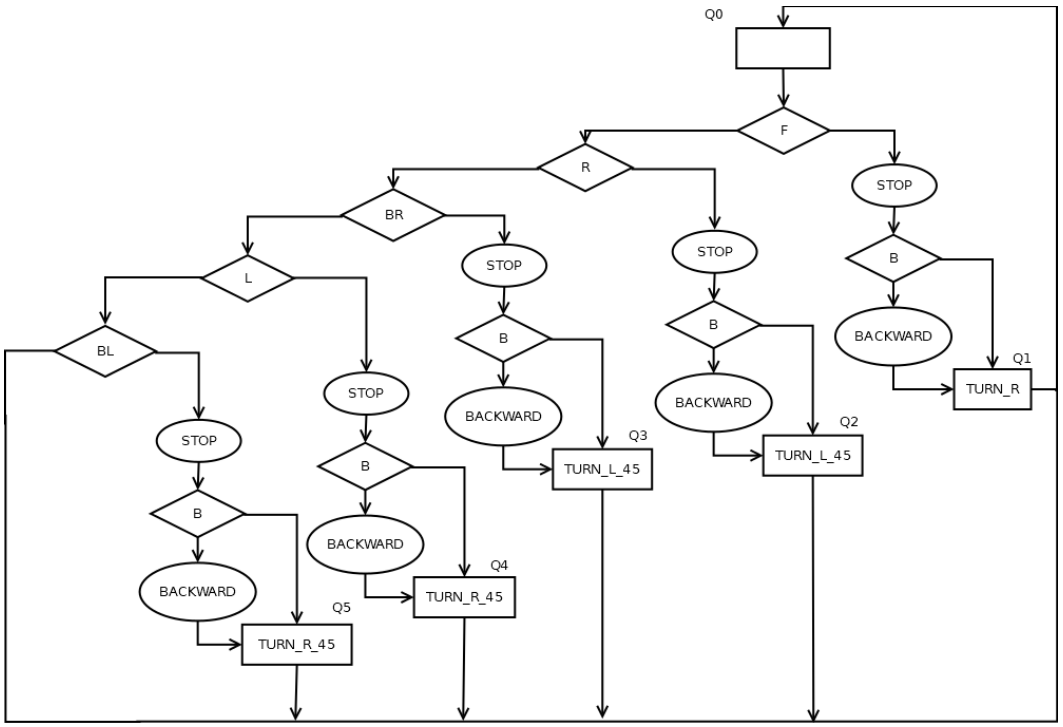


Figura 9.14. Capa de evasión de Obstáculos

Esta máquina de estados representa el comportamiento que debe de tener el robot para manejar la evasión de obstáculos en 4 diferentes direcciones. Como ya se mencionó en los 3 casos, el retroceso se maneja como una salida condicional, ya que primero se tiene que comprobar que no haya obstáculos atrás. En caso de que existan, entonces se deberá de omitir esa salida y se irá directamente al estado siguiente donde la salida es la dirección a la que el robot debe girar.

También hay que tomar en cuenta que las salidas generadas en los estados Q1, Q2, Q3, Q4 y Q5 se envían al supresor de las salidas que provienen de la máquina de estados de navegación Libre. Esta máquina de estados se ejecuta una y otra vez mientras el robot está en funcionamiento.

Capa de dirigirse al destino.

Esta capa es un complemento directo de la primera capa, la capa de supervisión. Una vez que la capa de supervisión ha detectado la ubicación de la meta y genera las salidas que indican su dirección, esta capa las toma como entrada y en base a ellas genera las salidas que le indicarán a los actuadores del robot, hacia qué dirección debe de girar este para poder dirigirse directamente hacia la meta.

La máquina de estados que representa este comportamiento se muestra a continuación:

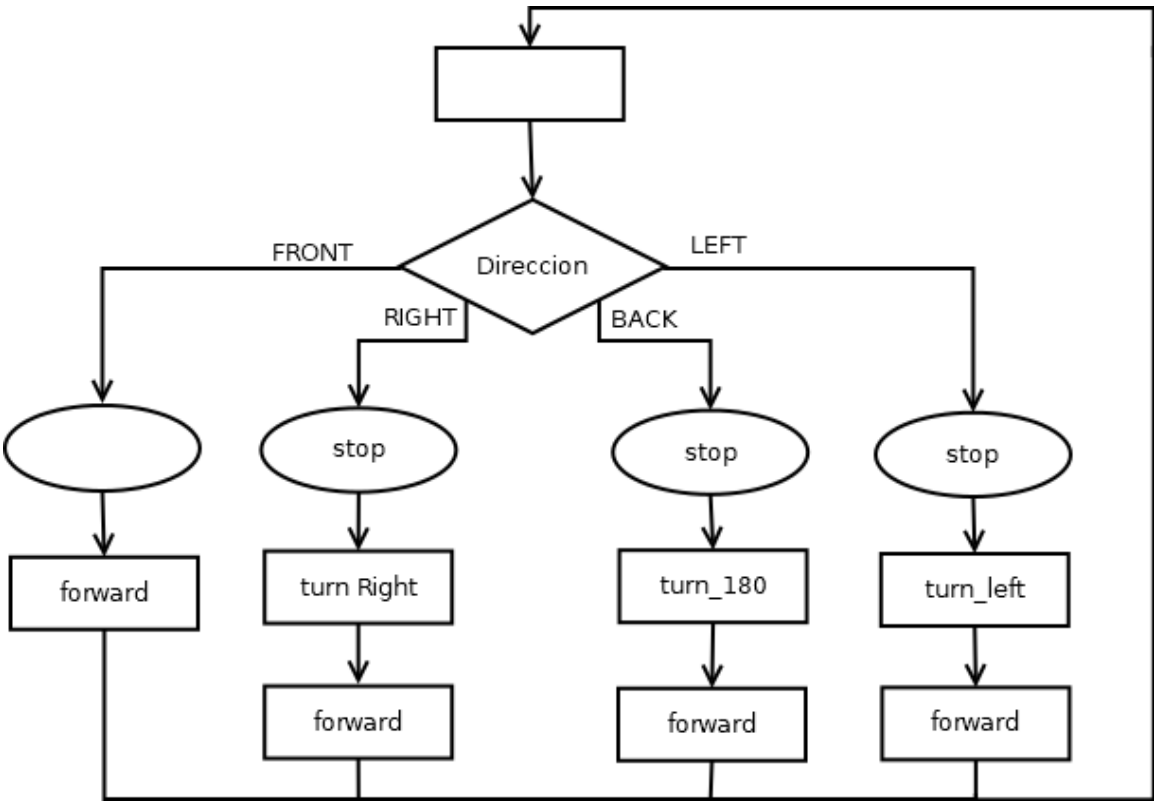


Figura 9.15. Capa de dirigirse al destino

En el estado inicial de esta máquina de estados, esta capa estaría recibiendo las salidas que generó la capa de supervisión, a continuación las evalúa y define el siguiente estado.

Si la entrada recibida es la dirección frontal, no se genera una salida condicional y el robot simplemente continúa avanzando en su estado siguiente.

Si la entrada recibida es la dirección lateral derecha, se genera una salida condicional en la que el robot se detiene y en un nuevo estado gira unos grados hacia la derecha. Finalmente continúa avanzando en su estado siguiente.

Si la entrada recibida es la dirección trasera, se genera una salida condicional en la que el robot se detiene y en un nuevo estado gira 180 grados de modo que su lado frontal “vea” hacia la meta. Finalmente continúa avanzando en su estado siguiente.

Si la entrada recibida es la dirección lateral izquierda, se genera una salida condicional en la que el robot se detiene y en un nuevo estado gira unos grados hacia la izquierda. Finalmente continúa avanzando en su estado siguiente.

Al final de cada uno de estos procesos el robot repite todo el proceso, desde la capa de supervisión, pasando por la de obstáculos y de nuevo llegando a la capa de dirigirse a la meta.

Capa de navegación libre.

Esta capa resulta ser muy sencilla, puesto que es una máquina de estados con un solo estado que no recibe ninguna entrada y su salida es siempre la misma, la de avanzar solo para hacer que el robot se mueva.

De hecho, en la capa anterior se puede notar que la salida común para todos los casos era la de avanzar. Si se separa esta salida, se tendría una máquina de estados como la siguiente.

Sin embargo, hay que recordar que la navegación libre del robot se ve afectada por dos supresores que deciden qué salida entregar: estas salidas extras corresponden a las capas de dirigirse a la meta y evasión de obstáculos.

Dependiendo de la dirección en la que se encuentre la fuente luminosa se decide si se afecta o no la navegación libre. De igual forma, si existe un obstáculo que se debe de evadir, también se decide si se afecta o no a la salida elegida: navegación libre o dirigirse a la meta. La salida de esta capa afecta directamente a los actuadores del robot.

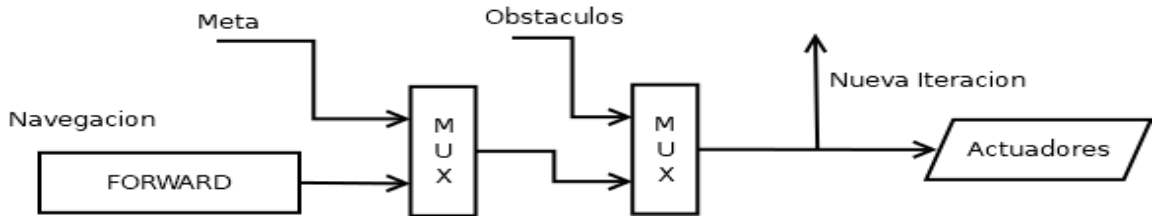


Figura 9.16. Capa de navegación libre

La salida de esta capa también se envía como entrada para la primera capa generando así una nueva iteración que repetirá todo el proceso hasta que la capa de supervisor indique que ya se ha alcanzado la meta y entonces el robot debe de mantenerse detenido pues ya ha alcanzado su objetivo.

Con lo expuesto en este capítulo, ahora se puede proceder a implementar la lógica que complementará el funcionamiento del robot para que este tenga ya un comportamiento autónomo y consiga lograr el último objetivo particular III, así como el objetivo general.

Capítulo 10. Pruebas y resultados

En este capítulo se describen las pruebas realizadas sobre el robot de modo que se garantice que el robot cumple con sus objetivos.

En primer lugar, se comprueba el correcto funcionamiento de cada uno de los algoritmos para la captación y procesamiento de las señales recibidas por los sensores y algoritmos que implementan la cinemática del robot y el control de velocidad. *En segundo lugar* se comprueba el correcto funcionamiento de las diferentes capas de la AFSM que define el comportamiento del robot. *En tercer lugar* se comprueba el funcionamiento en conjunto de todos los módulos programados para el robot de modo que se pueda verificar que se cumple con el objetivo principal de este trabajo.

A continuación se listan las pruebas realizadas.

Comprobación de algoritmos para sensores y actuadores.

- Se comprueba que los sensores LDR detectan las incidencias de luz y se define un valor máximo que será de apoyo para determinar si el robot está muy cerca de la luz o muy lejos.
- Se comprueba que los sensores ultrasónicos miden con bastante aproximación las distancias entre un objeto y el robot para que de este modo se pueda determinar si existe o no un obstáculo.
- Se comprueba que los sensores infrarrojos miden con bastante aproximación las distancias entre un objeto y el robot para que de este modo se pueda determinar si existe o no un obstáculo.
- Se comprueba que los sensores de contacto (bumpers), generan la señal que le indica al robot si ha chocado con un obstáculo o no.
- Se comprueba que el robot recorre las distancias señaladas.
- Se comprueba que el robot gira los ángulos señalados.
- Se comprueba que el control PID para la velocidad, actúa correctamente.

Comprobación de las capas del comportamiento del robot.

- Capa de supervisión:
    - Se comprueba que el robot reconoce cuando está en la meta
    - Se comprueba que el robot detecta la dirección en la que se encuentra la meta y gira hacia ella.
- Capa de detección de obstáculos
    - Se comprueba que funcionan correctamente los algoritmos para medición de distancias y detección de obstáculos.
    - Se plantean diferentes escenarios en los que el robot puede encontrar un obstáculo y se evalúa la respuesta entregada en cada caso.
- Capa de evasión de obstáculos.
    - Se comprueba que el robot genera los comportamientos esperados al encontrarse con un obstáculo
    - Se ejecutan pruebas sobre un entorno experimental y se observa si el robot se comporta de forma reactiva frente los obstáculos y busca evadirlos.

- capa de navegación libre.
    - Se comprueba que el robot realiza su navegación por el medio sin reaccionar a la presencia de obstáculos y/o incidencias de luz en cualquier dirección.

Comprobación del funcionamiento general y cumplimiento del objetivo general.

- Se ejecutan pruebas sobre un entorno experimental que contiene la meta señalada por un faro de luz y los obstáculos acomodados de forma aleatoria esperando observar que el robot navegue ejecutando sus comportamientos reactivos y consiga llegar a la meta.

10.1 Comprobación de algoritmos para sensores y actuadores.

Pruebas en sensores LDR.

Para esta prueba simplemente se conectaron los sensores LDR a los puertos analógicos tal y como se muestran en el esquema titulado “Conexión de la torre de sensores de luz” que puede encontrar en el apéndice F, o en la figura 4.x del capítulo 4. Apoyándose de una lámpara led y mediante el uso de la función de analogRead() y el monitor serial de Arduino se comprobó el correcto funcionamiento de los 4 sensores instalados.

La prueba simplemente consistió en acercar o alejar la lámpara led a cada uno de los sensores y observar los valores que se iban listando en el monitor serial.

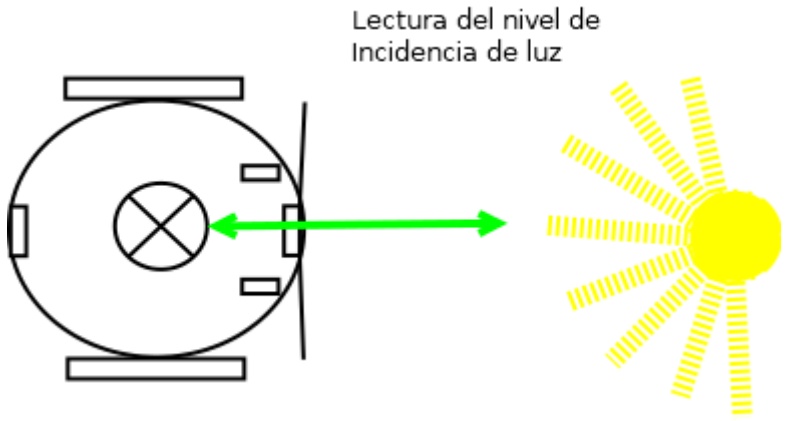


Figura 10.1. prueba en lectura de sensor LDR

Como resultado se pudo observar que los sensores funcionaban correctamente ya que con los datos registrados se veían los siguientes comportamientos:

1. Al acercar lámpara led a cada sensor, se detectaba una mayor incidencia de luz y el valor devuelto incrementaba debido a que la resistencia se reducía considerablemente y permite un mayor flujo de voltaje.

2. Al alejar la lámpara led de cada sensor, se detectaba una menor incidencia de luz y el valor devuelto decrementaba bastante debido a que la resistencia incrementaba demasiado y el flujo de voltaje se reducía casi a cero.

Estos comportamientos coinciden con el funcionamiento del LDR descrito en la sección 4.2.3 del capítulo 4.

Las pruebas se realizaron durante un minuto y en dos ambientes. Uno sin iluminación más que la de la lámpara que representaba el objetivo, y otro con iluminación extra provocada por las lámparas del cuarto donde se realizaron las pruebas.

El objetivo de la primera prueba era comprobar la intensidad de luz que debía de detectar el sensor LDR en un ambiente ideal, donde solo existe una sola fuente de luz. El segundo escenario fue solo para comprobar si el sensor sería capaz de detectar la fuente de luz objetivo e ignorar el resto de la iluminación ambiental.

Estos son los resultado:

En el ambiente ideal y sin iluminación alrededor, los valores se aprecia que los valores analógicos más bajos registrados por los sensores LDR están por debajo del 20.

```
1 LDR_F: 15 LDR_R: 13 LDR_B: 14 LDR_L: 53
2 LDR_F: 12 LDR_R: 13 LDR_B: 12 LDR_L: 14
3 LDR_F: 14 LDR_R: 13 LDR_B: 29 LDR_L: 15
4 LDR_F: 15 LDR_R: 13 LDR_B: 13 LDR_L: 54
5 LDR_F: 18 LDR_R: 15 LDR_B: 14 LDR_L: 14
6 LDR_F: 12 LDR_R: 13 LDR_B: 13 LDR_L: 13
7 LDR_F: 14 LDR_R: 13 LDR_B: 12 LDR_L: 47
```

Si se llega a presentar iluminación extra en la habitación de pruebas, como en el caso del segundo escenario, los valores de analogicos no superaban en general un valor de 90.

```
65 LDR_F: 15 LDR_R: 13 LDR_B: 13 LDR_L: 45
66 LDR_F: 79 LDR_R: 15 LDR_B: 16 LDR_L: 22
67 LDR_F: 80 LDR_R: 63 LDR_B: 20 LDR_L: 22
68 LDR_F: 78 LDR_R: 15 LDR_B: 24 LDR_L: 23
69 LDR_F: 98 LDR_R: 17 LDR_B: 18 LDR_L: 44
70 LDR_F: 80 LDR_R: 15 LDR_B: 16 LDR_L: 22
71 LDR_F: 80 LDR_R: 4 LDR_B: 21 LDR_L: 23
72 LDR_F: 83 LDR_R: 15 LDR_B: 18 LDR_L: 39
```

Por lo que se podría proponer un algoritmo para automatizar la calibración de los sensores LDR, haciendo que se lean los valores analogicos y obteniendo un valor promedio de la intensidad de luz que se percibe alrededor, y tomarlo como un límite inferior para determinar qué intensidades de luz debe de ignorar.

Para este caso, y por fines prácticos, el robot va a operar en el escenario uno, donde se contempla que en el ambiente solo existe una sola fuente de luz; por lo para determinar la ubicación de la fuente solo se va a elegir la dirección de donde provenga la mayor incidencia de luz.

Por otro lado, en ambos casos las intensidades de luz que llegaban a capturar los sensores LDR cuando había una fuente luminosa próxima al robot, superan en general los valores de 200; y cuando la fuente estaba considerablemente cerca, los valores analógicos superan el valor de 400.

```
35 LDR_F: 111 LDR_R: 312 LDR_B: 24 LDR_L: 21
36 LDR_F: 365 LDR_R: 212 LDR_B: 20 LDR_L: 10
37 LDR_F: 355 LDR_R: 231 LDR_B: 21 LDR_L: 21
38 LDR_F: 424 LDR_R: 221 LDR_B: 27 LDR_L: 22
39 LDR_F: 442 LDR_R: 206 LDR_B: 19 LDR_L: 21
```

En las lecturas anteriores, los datos sugieren que la fuente luminosa está al frente del robot, pero desplazada un poco a la derecha, por eso esos dos sensores capturan la mayor intensidad de luz.

De estas pruebas también se puede proponer un valor que sirva como límite superior para determinar cuando el robot ya haya alcanzado la meta; en este caso se escogió como límite superior el valor de 400, ya que al acercar bastante la luz al sensor, las lecturas superan ese valor.

Pruebas en sensores Ultrasónicos.

Para estas pruebas simplemente se utilizó el algoritmo descrito en la sección 4.2.2 del capítulo 4 donde se trató sobre el sensor HC-SR04.

De manera similar a las pruebas con el sensor ldr, se utilizaron objetos que se fijaban a una distancia propuesta, la cual era medida con ayuda de un flexómetro desde la base del sensor hasta el objeto; adicional a esto, se hizo uso del monitor serial de arduino para visualizar las distancias calculadas a partir de la información capturada por el sensor para corroborar los resultados.

Las medidas propuestas fueron 4 y se muestran en la siguiente tabla. Por cada distancia propuesta se realizaron 5 pruebas para corroborar el buen funcionamiento del módulo encargado de calcular las distancias con el sensor sonar.

Distancia propuesta cm	distancias medidas sensor HC.SR04	Distancias reales flexómetro	Distancia promedio
5	5.00	5.00	5.00
	5.00	5.00	
	5.00	5.00	

	5.00	5.00	
	5.00	5.00	
10	10.00	10	10.00
	10.00	10	
	10.00	10	
	10.00	10	
	10.00	10	
15	15.00	15	15.00
	15.00	15	
	15.00	15	
	15.00	15	
	15.00	15	
20	19.00	20	19.8
	19.00	20	
	20.00	20	
	21.00	20	
	20.00	20	

Tabla 10.1 Muestras de las mediciones realizadas con los sensores HC-SR04

De acuerdo a los datos de la tabla anterior, se logró comprobar que el sensor junto con el módulo encargado de calcular las distancias funcionaban bastante bien, pues los cálculos corresponden a las medidas tomadas con el flexómetro, excepto en la última distancia, donde se observó un ligero error en los datos pues a pesar de que la medida tomada con el flexómetro indicaba que el objeto se encontraba ubicado a la distancia propuesta, hubo algunas perturbaciones en la lectura del sensor que evitaron que se llegara a un cálculo preciso.

En base a los resultados obtenidos de estas pruebas, se pudo determinar un valor que representa la distancia mínima detectada por el sensor para que el robot determine si tiene un obstáculo al frente o atrás; dicha distancia es de 15 cm, pues a parte de los resultados de las pruebas realizadas, se tomó en cuenta las dimensiones del robot, de modo que se pudiese tener un espacio suficiente entre el robot y el obstáculo para que este pudiera ejecutar sus acciones de evasión sin problemas.

Pruebas en sensores infrarrojos.

Para estas pruebas se utilizó el algoritmo descrito en la sección 4.2.2 del capítulo 4 donde se trató sobre el sensor Infrarrojo.

De manera similar a las pruebas hechas con el sensor HC-SR04, se hizo uso de objetos que se aproximaban y alejaban de el sensor; solo que esta vez se utilizaron objetos de diferente color, ya que como estos sensores funcionan a partir del reflejo de una señal de

luz infrarroja, la lectura podría variar dependiendo del color y tonalidad que tuviese cada objeto.

Distancia propuesta cm	objeto: Madera de pino color claro	objeto: caja color negro	objeto: botella color púrpura obs	Medidas promedio
5	6	5	7	6
	6	5	7	6
	6	5	7	6
	6	5	8	6.3
	6	5	7	6
10	13	10	12	11.66
	13	9	12	11.33
	13	9	12	11.33
	13	10	12	11.66
	13	9	13	11.66
15	14	13	19	15.33
	14	13	19	15.33
	15	14	20	16.33
	13	13	19	15
	14	13	19	15.33
20	17	17	54	29.33
	17	17	53	29
	18	16	51	28.33
	17	17	54	29
	17	17	54	29

Tabla 10.2. Muestras de las mediciones de distancias con Sensor IR

En la tabla anterior se muestran los resultados de la prueba del sensor infrarrojo. Como el cálculo de distancias de este sensor depende de la intensidad de luz infrarroja que se refleja en las superficies y se captura por el sensor, era importante comprobar cómo se comportan los datos si el sensor trata de medir la distancia con objetos de diferente material y color; la intensidad de luz reflejada no va a ser la misma en todos los casos, pues algunos colores y materiales absorben más luz y reflejan señales más bajas y otros actúan de forma contraria.

En el caso de la madera de pino a pesar de que poseía un color más claro su superficie muestra un color opaco, por lo que se puede intuir que este material refleja con menor intensidad la luz, por lo que se afectaba la precisión del cálculo de distancias. El mejor comportamiento se da con la medida propuesta de 15 cm, pues los datos obtenidos

tras la lectura y cálculo de distancias, arrojaban valores que fallaban por 1 o 2 cm por debajo del objetivo.

En el caso de la caja de color negro, se tubo una mejor respuesta en cuanto a las primeras tres medidas propuestas, pues a pesar de tener un color muy obscuro que en teoria, no deberia de permitir reflejar bien las señales de luz; la superficie de la caja era lisa y ademas recubierta por un barniz que le permitia tener un color mas brillante, razón por la que se beneficiaba el reflejo de la señal infrarroja y en consecuencia el cálculo resulto mas preciso; por lo menos para el caso de la medida de 5 cm. Para el caso de las medidas de 10 y 15 cm el comportamiento era similar al de la madera de pino, dando resultados con un fallo de 1 a 2 cm por debajo del objetivo. Para los 20 cm, el fallo al igual que en la madera, ya aumentaba y el error era en general de 3 cm por debajo del objetivo.

En el caso de la botella de color purpura el fallo fue bastante alto. De acuerdo a las observaciones realizadas, esto se pudo deber nuevamente al material de la botella, que era plástico y tenía un poco de transparencia, lo cual afectaba al reflejo de la señal infrarroja por lo que el cálculo de la medida fallaba bastante.

Al promediar las medidas tomadas en cada prueba se observó que en general, el fallo de la medida se encontraba entre los 1.3 o 1.6 cm, esto si se incluye en el cálculo los valores tomados con la botella. Sin embargo, si solo se ponía atención a los valores que mejor se comportaron (madera y caja), el fallo era de 0.5 a 1 cm.

En la tabla los valores que mejor se aproximaban al objetivo, se daban en la medida de 15 cm, razón por la que al igual que con el sensor HC-SR04, se eligió como la distancia mínima que debe de detectar el sensor IR para determinar si hay o no un obstáculo.

Pruebas en sensores de contacto (Bumpers).

Para estos sensores las pruebas fueron muy sencillas, pues simplemente había que comprobar el cambio de estado, de cero a uno, del interruptor al pulsarlo o soltarlo. El resultado simplemente se mostraba en el monitor serial como sigue en la tabla a continuación:

Acción	Respuesta Esperada	Respuesta Entregada
Switch pulsado	HIGH	HIGH
Switch liberado	LOW	LOW

Tabla 10.3. Resultado de las pruebas del Switch Bumper

El verdadero reto para este sensor consiste en lograr que el algoritmo de detección de obstáculos logre monitorizar de forma oportuna el cambio de estado de este sensor para que se logre una interrupción en la navegación del robot y se lleven a cabo las acciones que corresponden a la evasión de obstáculos.

La resolución propuesta para este reto se muestra en la sección de pruebas para la capa de supervisión de obstáculos.

Pruebas en cinemática del robot.

Para realizar estas pruebas, se desarrollaron dos programas de apoyo: uno en arduino encargado de ejecutar las funciones encargadas de la cinemática y el control PID y uno en Python encargado de recibir los comandos de pruebas a través de un túnel SSH para que arduino las ejecutará.

El flujo de las pruebas era el siguiente:

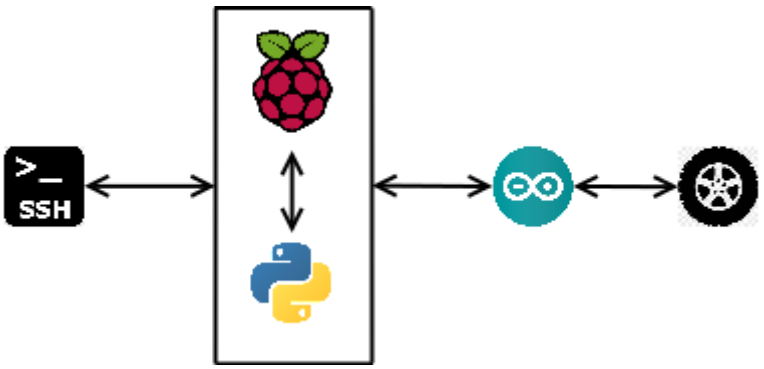


Figura 10.2. Flujo para las pruebas de la cinemática del robot.

Desde una terminal bash de Linux se conectaba a la computadora del robot por comunicación SSH y le enviaba unos comandos que capturaba el script python y a su vez le transmitía mediante la comunicación serial las instrucciones a arduino para que el robot se desplaza linealmente la cantidad de metros especificada, o girará un ángulo especificado en grados sexagecimales.

El script python y el sketch arduino usados para las pruebas las puede encontrar en el apéndice G.

Los comandos ejecutados son los siguientes:

Comando	Acción	Ejemplo
turn_L	Gira a la izquierda los grados sexagecimales especificados	turn_L 90.0
turn_R	Gira a la derecha los grados sexagecimales especificados	turn_R 90.0
advmts	Avanza los metros especificados	advmts 1.0
bckmts	retrocede los metros especificados	bckmts 1.0

Tabla 10.4 Comandos utilizados para las pruebas de la cinemática del robot

Pruebas sobre los ángulos de giro.

Para comprobar los grados que giró el robot hacia cualquiera de las direcciones, se preparó un “tapete” graduado similar al que se muestra en la siguiente figura, esto con el fin de tener una referencia con la cual se verificará que el robot si rotaba los ángulos señalados.

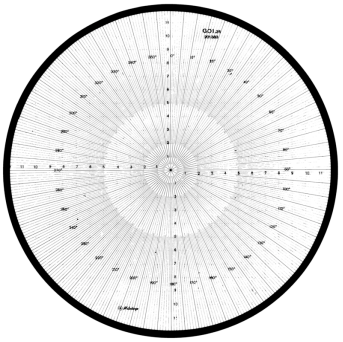


Figura 10.3. Tapete graduado para comprobar ángulos de giro [23]

Para simplificar más el proceso, se probó con los ángulos más conocidos (45, 60, 90, 135, 180, 270 y 360) y se observó si el robot acertaba la cantidad de grados deseados, si se aproximaba o definitivamente no había éxito y se requería de un ajuste en las operaciones.

Las muestras de los resultados arrojados son las siguientes.

Ángulo deseado	Ángulo obtenido	Porcentaje de Error
45	43	4.44%
60	59	1.66%
90	87	3.33%
135	140	-3.7%
180	189	5%
270	265	1.85%
360	372	3.33%

Tabla 10.5. Resultados obtenidos en pruebas sobre los ángulos de giro del robot

Durante las pruebas se pudo observar que existía un grave problema con la exactitud de los grados de giro reales del robot, pues no lograban acercarse a los valores deseados y de hecho su porcentaje de error se aproximaba al 50%, lo cual era bastante malo.

Para resolver el problema, se determinó de forma empírica una constante de ajuste que se aplicaría sobre la cantidad de pulsos calculados multiplicando el valor resultante para que el robot lograse alcanzar los grados deseados.

El valor de ajuste es de 2.5, ya que en todas las pruebas y mediciones realizadas, se noto que el giro que entregaba el robot era aproximadamente la mitad del valor deseado, y con esa constante se logró reducir considerablemente el porcentaje de error.

Pruebas en distancias recorridas

De forma similar a las pruebas realizadas en el giro del robot, se realizaron las pruebas para corroborar la exactitud de las distancias recorridas por el robot.

Se propusieron 4 distancias y se realizaron múltiples pruebas para verificar si el robot recorría correctamente las distancias indicadas. Sin embargo se presentó una situación similar a la de los grados de giro; el robot excedió las distancias, por lo que se tuvo que aplicar una nueva constante de ajuste que permitiera reducir el porcentaje de error.

La constante calculada de forma empírica fue de 1.75, y esta vez se aplicó como un divisor para la cantidad de pulsos calculados para realizar cada recorrido.

El valor de esta constante se dio debido a que se observó que el robot recorría distancias que eran aproximadamente el doble de la deseada. Con este valor se obtuvieron resultados más satisfactorios.

distancia deseada en metros	distancia real en metros	Porcentaje de Error
0.01	0.0105	-5%
0.05	0.049	2%
0.1	0.098	2%
0.15	0.145	3.33%

Tabla 10.6 Resultados obtenidos en pruebas sobre las distancias recorridas por el robot

10.2 Comprobación de las capas del comportamiento del robot.

Capa de Supervisión

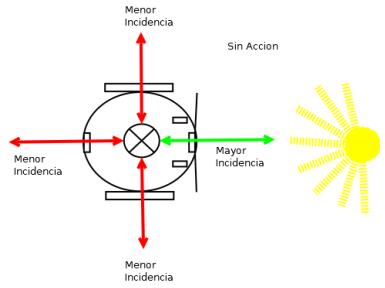
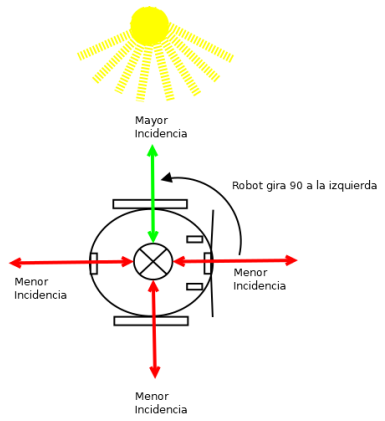
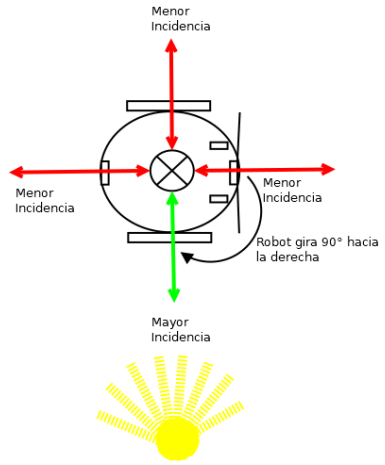
Para probar la capa de supervisión se recurrió al uso de una lámpara led, la cual tenía que apuntar hacia el robot desde cuatro direcciones diferentes para probar que el robot identificará correctamente la dirección de la meta y tomará la decisión de dirigirse hacia ella. En otro caso, se trataba de apuntar con la lámpara al robot de modo que se iluminarán todos los sensores LDR, de modo que el robot determinara si ya se encontraba o no en la meta.

. Para realizar las pruebas de la capa del supervisor en la parte donde busca y determina la dirección donde se encuentra la meta, se hizo uso del algoritmo descrito en la sección 4.2.3, el cual es simplemente un comparador de incidencias y determina cuál

lectura tomada es la que tiene mayor valor. Esta implementación en codigo arduino lo puede encontrar en el apéndice B, en la función *findGoal*().

Para las pruebas de la capa del supervisor en la parte donde se determina si el robot se encuentra o no en la meta, se desarrolló otra función sencilla que simplemente verifica que todos o al menos uno de los sensores, capture una incidencia con un valor superior al umbral definido durante las *pruebas de los sensores LDR*.

En la siguiente tabla se muestra una representación de las acciones realizadas.

Representación gráfica de la prueba	Descripción de la prueba
	Se apuntó con la fuente de luz (lámpara led) a la parte frontal del robot. Con esta prueba se espera que el robot responda que la dirección de la meta está **al frente** y no realiza ninguna acción adicional
	Se apuntó con la fuente de luz (lámpara led) al costado izquierdo del robot. Con esta prueba se espera que el robot responda que la dirección de la meta está **a la izquierda** y decida girar 90° a la izquierda.
	Se apuntó con la fuente de luz (lámpara led) al costado derecho del robot. Con esta prueba se espera que el robot responda que la dirección de la meta está **a la derecha** y decida girar 90° a la derecha.

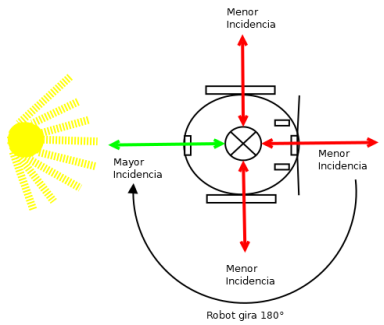
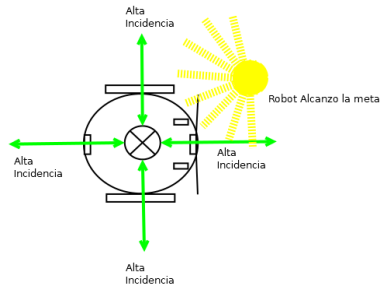
	Se apuntó con la fuente de luz (lámpara led) a la parte trasera del robot. Con esta prueba se espera que el robot responda que la dirección de la meta está **atrás** y gira 180°.
	Se apuntó con la fuente de luz (lámpara led) en una zona muy próxima al robot con la intención de que la incidencia fuera bastante alta en todos sus sensores LDR. Con esta prueba se espera que el robot responda que la fuente de luz está muy próxima, y como las incidencias superan el umbral, determine que ya está en la meta. No realiza ninguna acción extra, simplemente está detenido.

Tabla 10.7. Descripción de las pruebas a la capa de supervisión y resultados obtenidos

Capa de detección de obstáculos.

Para las pruebas de la capa de detección de obstáculos, se utilizó el algoritmo diseñado en la sección 9.3 del capítulo anterior, solo que con algunas ligeras modificaciones para ejecutar correctamente la detección de obstáculos.

A continuación se describe cómo se ejecuta el algoritmo final y que es el que ya está implementado en la función principal de arduino (revisar Apéndice E).

1. El robot busca obstáculos al frente, comprobando sus sensores en el siguiente orden:
    a. Estado de los sensores de contacto. Si los dos estaban en 1 o HIGH, el robot infiere que chocó con un obstáculo al frente.
    b. Distancias medidas por los IR: si las distancias que capturaban ambos sensores IR eran menores al umbral definido durante las pruebas, el robot infiere que tiene un obstáculo al frente.
    c. Distancia medida por el HC-SR04: Si la distancia medida por el sensor ultrasónico frontal era menor al umbral definido durante las pruebas, el robot infiere que tiene un obstáculo al frente.
2. El robot busca obstáculos a la derecha:
    a. Estado del sensor de contacto derecho. Si el estado es 1 o HIGH, el robot infiere que chocó con un obstáculo a su derecha.
    b. Distancias medidas por el IR derecho: si las distancias que capturaba el sensor IR derecho eran menores al umbral definido durante las pruebas, el robot infiere que tiene un obstáculo a su derecha.
3. El robot busca obstáculos a la izquierda.

a. Estado del sensor de contacto izquierdo. Si el estado es 1 o HIGH, el robot infiere que chocó con un obstáculo a su izquierda.
b. Distancias medidas por el IR izquierdo: si las distancias que capturaba el sensor IR izquierdo eran menores al umbral definido durante las pruebas, el robot infiere que tiene un obstáculo a su izquierda.

La razón por la que prioriza primero buscar obstáculos al frente en vez de a los lados es para reducir el riesgo de un comportamiento erróneo; si el robot detecta un choque o una distancia menor al umbral en uno solo de sus lados, reaccionara de acuerdo al comportamiento de evasión para ese lado, aun si esta lectura fue en ambos lados y en realidad se esperaba un comportamiento de evasión de obstáculos al frente.

También se decidió dar prioridad a verificar el cambio de estados de los sensores de contacto, ya que se espera que el sistema reaccione de forma casi inmediata como si se disparara una interrupción por software, cuando el robot choque.

Para comprobar que la detección de los obstáculos era correcta, se realizaron pruebas como se indican en las siguientes figuras y en el orden señalado.

Detección mediante bumpers.

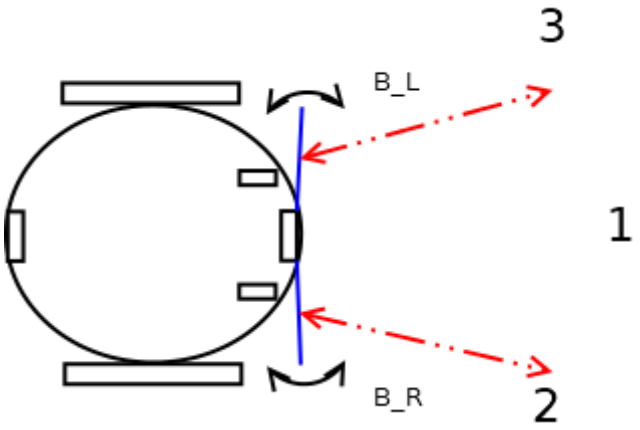


Figura 10.4. Pruebas sobre sensores de contacto.

1. Se pulsan ambos bumpers:
    - Respuesta: Obstáculo al frente
2. Se pulsa bumper derecho:
    - Respuesta: Obstáculo a la derecha.
3. Se pulsa bumper izquierdo:
    - Respuesta: Obstáculo a la izquierda.

Detección mediante IR.

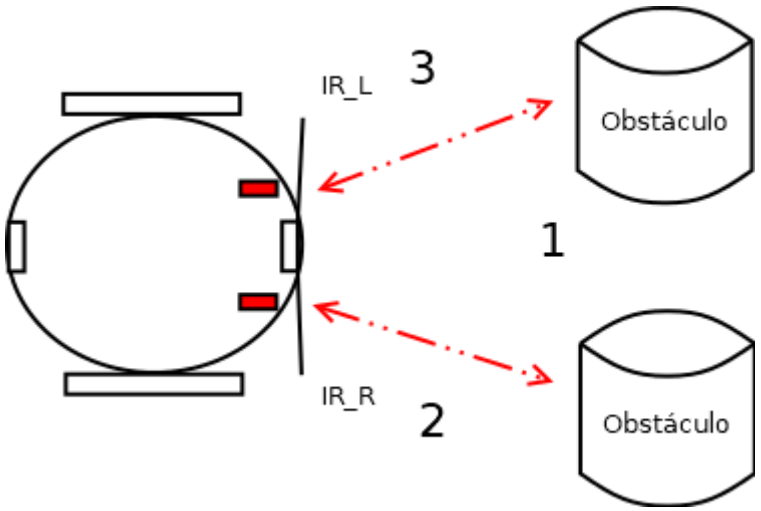


Figura 10.5. Pruebas sobre sensores IR

1. Se pone objeto frente a ambos IR:
    - Respuesta: Obstáculo al frente
2. Se pone objeto frente a IR derecho:
    - Respuesta: Obstáculo a la derecha.
3. Se pone objeto frente a IR izquierdo:
    - Respuesta: Obstáculo a la izquierda.

Detección mediante sensor HC-SR04

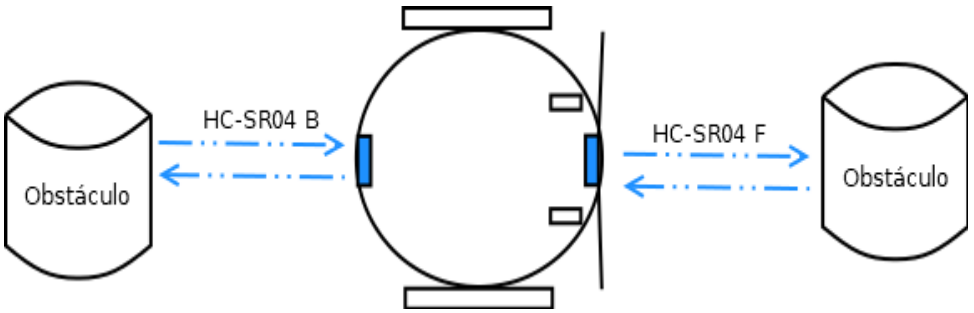


Figura 10.6. Pruebas sobre el sensor HC-SR04.

- Se pone un obstáculo frente al sensor HC-SR04 frontal:
    - Respuesta: Obstáculo al frente.

El uso del sensor HC-SR04 trasero se especifica en la fase de las pruebas.

La siguiente fase de las pruebas que se realizaron sobre esta capa son respecto al evasor de obstáculos y se ejecuta a partir de los resultados obtenidos arriba. En el algoritmo que se explicó anteriormente, se aplicó un nuevo cambió para que ahora en vez de enviar un mensaje al monitor serial de arduino, genere una bandera que es un valor entero, y que es consumida por una nueva función que se encargará de ejecutar los movimientos correspondientes para que el robot pueda evadir el obstáculo detectado.

Para esta prueba se buscó que el robot realizará las siguientes acciones:

- Que el algoritmo de detección genere la bandera correcta que indica la ubicación del obstáculo detectado.
- Que la bandera sea consumida por la función que se encarga de accionar los movimientos de evasión de acuerdo a la dirección del obstáculo.
- Que la evasión la realice de forma reactiva, es decir, el robot se mantendrá navegando libremente hasta que detecte un obstáculo; y en cuanto esto suceda, actúe de inmediato y lo evada.

En la siguiente figura se describe gráficamente un ejemplo sobre el comportamiento obtenido en el robot durante las pruebas.

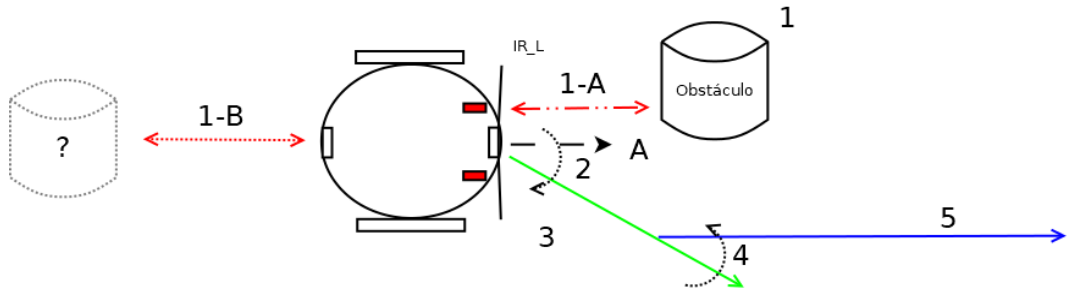


Figura 10.7. Descripción gráfica de la prueba en el evasor.

En el ejemplo mostrado en la figura 10.7 se describe lo siguiente:

1. Se colocó aleatoriamente un obstáculo sobre la trayectoria que recorre el robot para observar que este detectara correctamente la presencia del obstáculo y además determinará en donde estaba ubicado. En este caso fue un obstáculo a la izquierda.
    a. El robot detecta el obstáculo y se detiene
    b. El robot comprueba si hay un obstáculo atrás, si no lo hay retrocede unos cuantos centímetros, pero si lo hay, simplemente ejecuta el paso 2.
2. El robot gira 45° hacia una dirección contraria (derecha) a la que se encuentra el obstáculo detectado.
3. El robot comienza a desplazarse algunos centímetros hacia adelante para lograr evadir el obstáculo.
4. El robot vuelve a girar 45° grados de forma que anula el giro anterior, o sea, trata de recuperar su trayectoria original antes de la evasión.
5. el robot continúa con su recorrido.

Se repite los pasos expuestos en el ejemplo para todos los sensores, obviamente modificando las acciones que debe de tomar si ha detectado obstáculos al frente o a los costados.

Estas pruebas se aplicaron múltiples veces con el fin de rastrear problemas en la programación del robot, tanto en la máquina de estados encargada de esta capa como en la programación encargada de controlar los sensores y actuadores.

Los principales ajustes realizado para arreglar los problemas que se detectaron fueron:

- La velocidad a la que se desplaza el robot.
    - Se encontró un grave problema al hacer que el robot avanzará a una velocidad alta a que esto provocaba que el robot no reaccionara a tiempo cuando se encontraba frente a un obstáculo, por lo que terminaba chocando o atorandose con él.
- Control de lecturas erróneas en sensores IR.
    - Se llegaron a presentar problemas en las lecturas del sensor IR debido a caídas de voltaje, esto debido al alto consumo de energía cuando el robot realizaba su navegación.
    - Estas lecturas erróneas se llegaban a presentar de forma aleatoria arrojando valores negativos o valores gigantescos que no se podían evaluar correctamente provocando que el robot no detectara bien los obstáculos con sus sensores IR.
    - Para controlar esos casos, en el código encargado de manejar los datos recolectados de los sensores IR se colocó una estructura condicional que sólo se ejecutaba si en la lectura aparecen dichos valores negativos o gigantescos, forzando la respuesta entregada al detector de obstáculos para que se mantuviera dentro de un rango de valores positivos que se pudiesen evaluar.

Como resultado de las pruebas y ajustes llevados a cabo, se pudo observar el comportamiento reactivo del robot ante un obstáculo, pues en el escenario de pruebas se mantenían colocados aleatoriamente múltiples objetos que el robot detectaba y trataba de evadir.

Hasta este punto el robot no tenía una meta fija así que simplemente avanzaba y alteraba su navegación en cuanto apareciera un obstáculo.

Capa de dirigirse al destino.

Las pruebas en esta capa solo son un complemento a las pruebas realizadas en la capa de supervisión; una vez que el robot ya detectó la dirección donde se encuentra la fuente luminosa, simplemente continúa ejecutando su navegación libre, y si detectaba que ya había alcanzado la meta, se detenía.

Durante esta prueba también se buscó que el robot actuará de forma reactiva, así que se cambiaba la posición de la lámpara constantemente y se observó cómo el robot giraba nuevamente hacia donde estaba y continuaba con su navegación hasta alcanzarla. En pocas palabras: el robot seguía a la fuente de luz.

Capa de Navegación libre.

Para estas pruebas simplemente se dejó que el robot se desplazara libremente por el escenario de pruebas sin aplicar ninguna condición de detección de meta u obstáculos.

10.3 Comprobación del funcionamiento general.

Finalmente y para llevar a cabo la prueba final con la que se comprueba el cumplimento general de este trabajo, se preparó un escenario donde el robot debía de ejecutar su navegación evadiendo obstáculos fijados al azar y dirigiéndose siempre hacia una meta fija.

La pista preparada se muestra en la figura 10.8, donde el lado izquierdo de la imagen representa la línea de salida y en el lado derecho la línea de meta.

Figura 10.8. Pista de navegación libre

En el punto de salida obviamente tenía que estar colocado el robot. A lo largo del escenario se encontraban acomodados al azar múltiples objetos (bloques de madera) cuyo fin era el de impedir que el robot llegase directamente a la meta. Finalmente en el otro extremo, se encontraba fijada la fuente luminosa como se muestra en la figura 10.9.

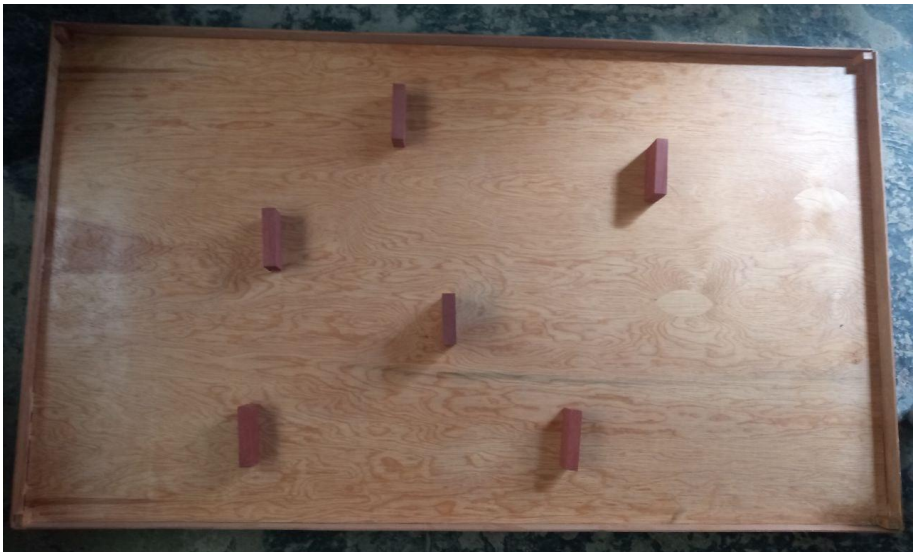


Figura 10.9. Pista de navegación con obstáculos.

Descripción de la prueba.

La prueba final consistió en poner el robot a navegar en el escenario final, ubicándolo inicialmente en cualquier punto de la línea de salida. En este punto el robot no necesariamente debía de estar “viendo” hacia el frente, pues la intención era que al momento de comenzar a operar, este sensara su medio para reconocer si se encontraba ubicado en la meta, y si no buscar en qué dirección se encontraba dicha meta y girar hacia esa ella.

Una vez que el robot ya identificaba la meta y se preparaba para dirigirse hacia ella, comenzaba su navegación, primero identificando si existía un obstáculo o no en el camino. Si no había obstáculos, este ejecuta su navegación libre simplemente para dirigirse hacia la meta; por el contrario, si hay obstáculos, ejecutaba las acciones correspondientes a la evasión dependiendo de la dirección en la que se haya detectado el obstáculo, para después repetir todos los pasos desde la identificación de la meta.

Cuando el robot logra llegar a la línea de meta e identifica que ya ha conseguido ubicarse en el objetivo, este se detiene por completo y ya no ejerce ninguna otra actividad.

En la figura 10.10 se muestra una simulación de la navegación final del robot que ya ejecuta todos las funcionalidades desarrolladas a lo largo de este trabajo. Esta figura representa a las pruebas realizadas con el robot real y los resultados obtenidos.

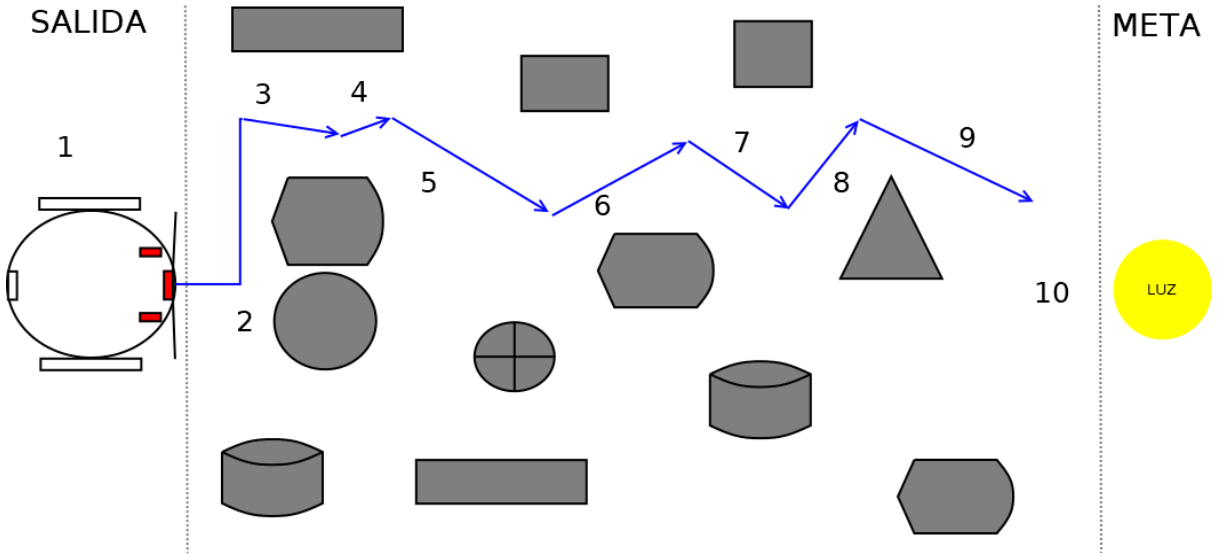


Figura 10.10 simulación del robot

A continuación se explica la navegación del robot a través del escenario de pruebas mostrado en la simulación de la figura 10.10.

1. El robot se ubica en algún punto sobre la línea de salida, en este caso, se ubica al centro y viendo hacia el frente.
2. Como primera tarea, busca la fuente de luz, que representa la meta que debe de alcanzar. Detecta que está justo al frente así que se dispone a navegar libremente hacia ella. Sin embargo identifica también que tiene un par de obstáculos al frente así que procede a ejecutar acciones de evasión.
3. El robot ha logrado evadir el primer obstáculo. Ahora procede a buscar de nuevo la fuente luminosa y la detecta hacia la derecha, por lo que gira unos grados hacia esa dirección y reanuda su avance.
4. El sensor IR derecho ha detectado un obstáculo, por lo que procede a ejecutar acciones de evasión y al finalizar vuelve a buscar la fuente luminosa.
5. Ha detectado que la fuente luminosa se encuentra nuevamente a la derecha, gira unos grados hacia esa dirección y reanuda su navegación libre para dirigirse hacia la meta.
6. El robot nuevamente ha detectado un obstáculo a la derecha, por lo que vuelve a ejecutar acciones de evasión y al finalizar vuelve a buscar la fuente luminosa.
7. Ha detectado nuevamente que la fuente luminosa se encuentra hacia su derecha, gira unos grados hacia esa dirección y vuelve a reanudar su navegación libre para dirigirse hacia ella.
8. Por última vez, vuelve a detectar un obstáculo justo al frente y procede a ejecutar acciones de evasión y al finalizar, vuelve a buscar la fuente luminosa.
9. El robot ha detectado que la fuente se encuentra hacia la derecha, gira unos grados hacia esa dirección y reanuda su navegación libre para dirigirse hacia ella.
10. El robot ha detectado que se encuentra justo en la meta y procede a detener sus acciones.

En el caso de esta simulación, tal y como se ve en la figura 10.10 no es más que una navegación en zigzag hasta llegar a la meta. Sin embargo, y por la naturaleza del comportamiento reactivo del robot, si se configura ese escenario y se coloca en una posición distinta al robot, este también va a entregar una navegación diferente al caso anterior, con más o menos pasos y todo en función de la cantidad de obstáculos que encuentre y las direcciones en las que detecte a la fuente luminosa.

Conclusión

Se consiguió construir un robot móvil que puede realizar una navegación autónoma a través de un escenario lleno de numerosos obstáculos con el fin de alcanzar una meta establecida.

Este modelo tuvo un desarrollo incremental, pues se partió desde un prototipo de robot móvil, que cuyas características iniciales consistían en un ensamblaje muy básico de varios tipos de sensores y un par de ruedas impulsadas por motores, que navegaba de forma errática y tomaba acciones muy simples al detectar un posible obstáculo o una fuente de luz.

Con el desarrollo de la primera y segunda sección de este trabajo(marco teórico, componentes y ensamblaje del robot), se fueron comprendiendo las características y funcionamiento de los componentes del robot, así como de la morfología elegida que en este caso fue la del robot de configuración diferencial.

Es estudio fue bastante importante, pues a partir de aquí se logró diseñar e implementar una enorme porción de los algoritmos encargados de manejar a los sensores involucrados y el procesamiento de los datos recolectados que posteriormente servirían para definir el comportamiento del robot, así como la generación de señales de control para que los actuadores ejecutarán los movimientos básicos del robot.

Con el desarrollo de la tercera sección (cinemática y control de velocidad) se agregó una nueva capa de complejidad al robot, y gracias al análisis cinemático que se trató en esta sección, se mejoraron los algoritmos encargados de la navegación básica y se complementaron con otros nuevos de modo que se afinaran sus movimientos haciéndolos más precisos; cosa que después permitió controlar las distancias recorridas y grados de rotación en sus respectivos desplazamientos lineales y angulares.

Adicionalmente, el análisis cinemático también permitió que fuese posible calcular las velocidades de las ruedas del robot y apoyándose de la teoría de control, se consiguió aplicar un sistema para regular automáticamente las velocidades y permitir que el robot lograra mejorar su navegación por el medio.

Ya en la cuarta sección (Comportamiento del robot) se tenía un prototipo bastante avanzado que ya cumplía con su primeros dos objetivos particulares: medir su entorno para reconocer objetivos y obstáculos, y la capacidad de medir su estado interno para tener un mejor desempeño en la navegación; ahora solo faltaba agregar la inteligencia que lo dotará de autonomía y cumplir así con el tercer objetivo particular.

Para esta sección se llevó a cabo el análisis, diseño e implementación de los algoritmos que dotaron al robot con la capacidad de actuar de forma autónoma y de rápida adaptación a los cambios espontáneos en el medio de acuerdo a lo que plantea el modelo de comportamiento reactivo. Con esto se consiguió completar al prototipo para que ahora pudiese cumplir con el objetivo general.

Finalmente en la última sección se conjuntaron en un único modelo bastante robusto y funcional, todos las características desarrolladas a lo largo del trabajo y se logró obtener un robot móvil que conseguía navegar de forma autónoma por el escenario propuesto. Aquí se pudo observar que el robot efectivamente, buscaba el objetivo y se preparaba para dirigirse hacia él; comenzaba a ejecutar su navegación y la alteraba en cuanto detectaba un obstáculo que le impidiera avanzar y tuviese que esquivarlo con movimientos bastante precisos o identificara un cambio en la posición del objetivo que tenía que alcanzar obligándolo a reconfigurar su navegación.

En conclusión: se logró cumplir con todos los objetivos propuestos para este proyecto.

Siguientes Pasos

Como ya se indicó, con la conclusión de este trabajo se consiguió obtener un modelo de robot móvil autónomo bastante robusto y funcional. Sin embargo y a consideración personal, pienso que se puede mejorar aún más este modelo, por lo que también hice un gran esfuerzo por dejar preparado el proyecto para que se pueda escalar a un modelo aún más avanzado.

A continuación listo las propuestas de mejoras que se pueden aplicar para escalar más el proyecto.

Ampliar el hardware del robot.

Esta propuesta es con el fin de hacer que el robot tenga más sensores conectados y obtenga una mejor "visión" del medio en el que está ejecutando sus actividades. El robot actual funciona bien con el número escaso de sensores que posee actualmente, sin embargo tiene demasiados puntos ciegos ya que solo cubre cuatro direcciones: Frente, Izquierda, Derecha, Atrás. Esto sería una vulnerabilidad que lo haría fallar ante un reto superior.

Calibrar la detección de fuentes luminosas.

Como el robot se probó principalmente en un ambiente controlado donde prácticamente la única fuente luminosa era la de la meta, no se puso mucha atención a este punto, más que para definir el umbral con el que se identificaría la llegada al objetivo. Si el robot se llegase a enfrentar a un medio que tiene más iluminación ambiental, tendría problemas para identificar su objetivo.

Para mejorar esto, sería bueno ajustar el algoritmo de búsqueda de meta de modo que el robot sea capaz de discriminar la luz ambiental y enfocarse únicamente en la que proviene de la fuente luminosa que representa su meta.

Implementación de ROS

En la idea inicial para este trabajo se tenía contemplado el manejo del Sistema Operativo Robótico ROS para implementar la programación del robot, pero debido a ajustes en el alcance del proyecto, se terminó descartando.

Sin embargo, se cuidó la programación del robot hecha en arduino separandola en módulos especializados para cada tarea involucrada, por lo cual, para una siguiente versión de este proyecto, se podrían traducir a **nodos de ROS** sin tanto problema y comunicarlos con las raspberry que ya tiene preparado el ambiente ROS.

Realizando este implementación también sería posible separar la programación del robot de modo que en arduino solo queden nodos especializados en controlar las señales de entrad y salida para sensores y actuadores, y en la raspberry los nodos encargados de procesar los datos recolectados y ejecutar la AFSM del robot, cosa que permitiría tener un proyecto con una arquitectura más robusta y más fácil de mantener.

Conectar al simulador del laboratorio de Bio-Robótica

Con la implementación de ROS en el proyecto, sería posible lograr que el robot se pudiese conectar y funcionar con con el simulador que posee el laboratorio de Bio-robótica, pues este está desarrollado precisamente con ROS. Aquí el reto sería entender cómo funcionan los nodos involucrados en ese simulador y lograr comunicarlos con el robot físico.

Si se logra establecer esa comunicación, el simulador podrá replicar los movimientos que hace el robot físico, y también, el robot físico podrá replicar los movimientos que ejecuta el robot de la simulación.

Apéndices

Apéndice A.

Código Arduino para definiciones globales para el robot

Archivo global.h ubicado en la ruta "*Baltazar_robot/Arduino_files/Baltazar/Modules/Global/*" del proyecto. Archivo dedicado únicamente a declarar todas las variables globales que se utilizan para definir puertos de I/O para sensores y actuadores, así como constantes y demás variables de control y configuración implicadas.

```
==========================================================================
/**
 * @file general.h
 * @author ricardo santiago (musanlori@gmail.com)
 * @version
 * @details Archivo de encabezado que declara las variables globales del proyecto
 *
 */

// definicion de puertos para sensores ultrasonicos

#define TRIGGER_F 11
#define ECHO_F 10
#define TRIGGER_B 9
#define ECHO_B 8

//definicion de puertos para sensores de contacto
#define CONTACT_R 12
#define CONTACT_L 13

//definicion de puertos para sensores LDR
#define LDR_FRONT A5
#define LDR_RIGHT A4
#define LDR_BACK A3
#define LDR_LEFT A2

//definicion de puertos para sensores IR
#define IR_R A0
#define IR_L A1

//Muestreo IR
const int sample_ir = 20;

//PINES DE LECTURA PARA EL ENCODER
int RH_ENCODER_A = 2;
int LH_ENCODER_A = 3;

// Definicion de los puertos de control de los motores
#define V_MRIGTH 5
#define D_MRIGTH 4
#define V_MLEFT 6
#define D_MLEFT 7

//CONSTANTES DE DIRECCIONES EN LOS SENSORES DE LUZ

#define LIGHT_FRONT 1
```

```
#define LIGHT_RIGHT 2
#define LIGHT_LEFT 3
#define LIGHT_BACK 4

//Constantes Cinematica del robot

#define PULSOSPORGIRO 300
#define PI 3.14159265358979323

//todo ajustar a mts
#define RADIOROBOT 4.5   //medida en cm
#define RADIORUEDA 2.1   //medida en cm

#define TIEMPO_MUESTREO 50 //tiempo de muestreo en millis
#define ADVANCEOBST 0.05   //el robot avanza x mts

//valores iniciales del pwm para giro de la rueda
int PWM_MOTOR_R = 160;
int PWM_MOTOR_L = 160;

//Valor para detener el giro de las ruedas
int M_STOP = 0;

//constantes para los valores limite de PWM
const int MAX_PWM_VAL = 255;
const int MIN_PWM_VAL = 0;

//contadores de pulsos en los encoders del motor.
//Seran de utilidad para determinar los movimientos del robot
volatile unsigned long left_count_ticks = 0;
volatile unsigned long right_count_ticks = 0;

//variables para el registro de posiciones de las ruedas del robot
//su utilidad sera para llevar a cabo el calculo de velocidades del robot
volatile unsigned long left_position = 0;
volatile unsigned long left_last_position = 0;
volatile unsigned long left_delta_position = 0;
volatile unsigned long right_position = 0;
volatile unsigned long right_last_position = 0;
volatile unsigned long right_delta_position = 0;

//variables que garantizan el tiempo de muestreo entre interrupciones
volatile unsigned _last_sample_time = 0;
volatile unsigned _sample_time = 0;
volatile unsigned _delta_time = 0;

//contantes para el control PID de velocidad
const double _kpv = 0.275;  //contante de proporcionalidad
const double _kiv = 0.25;   //constante de integracion
const double _kdv = 0.225;  //contante de derivacion

const double VEL_REF = 40.0;   //velocidad deseada en cm/s

double _Rd[2] = {0.0, 0.0};         //Rn de la llanta derecha
double errRight[3] = {0.0, 0.0, 0.0}; //error en la velocidad de la llanta derecha

int _Ri[2] = {0.0, 0.0};           //Rn de la llanta izquierda
double errLeft[3] = {0.0, 0.0, 0.0}; //error de la velocidad en la llanta izquierda
```

```
//umbrales intensidad luminosa
int highestThreshold = 325;

//manejo de estados en el detector de obstaculos

const int CHECK_FRONT = 50;
const int CHECK_RIGHT = 100;
const int CHECK_LEFT = 200;
const int CHECK_BACK = 250;

int _obts_state = CHECK_FRONT;

//Distancias minimas para definir si existe o no un obstaculo

const double UMBRAL_DIST_HCSR04 = 15;
const double UMBRAL_DIST_IRSENSOR = 18;

//Constantes para indicar el comportamiento del evasor

const int OBTS_FRONT = 1;
const int OBTS_RIGHT = 2;
const int OBTS_LEFT = 3;
const int OBTS_BACK = 4;
const int OBTS_ERROR = -1;

//estima los mts que debe moverse el robot para evadir
//un obstaculo segun sus dimensiones
//grande (detectado por ambos bumpers)
//mediano (detectado por ambos IR)
//pequeño (valor por defecto para las evasiones)

const double OBST_L = 0.15;
const double OBST_M = 0.09;
const double OBST_S = 0.01;

//avance para evadir obstaculo
const float FORWARD_MTS = 0.05f;
// distancia para retroceder
const float BACKWARD_MTS = 0.08f;
//Angulos de Giro
const float ISFRONT = 90.0f;
const float ISASIDE = 60.0f;

const float R_ANGLE = 90.0f;
const float S_ANGLE = 180.0f;

=========================================================================
```

Apéndice B.

Código Arduino para control de sensores

Archivo sensors.cpp ubicado en la ruta *“Baltazar_robot/Arduino_files/Baltazar/Modules/Sensors_modules/”* del proyecto. Dedicado a desarrollar únicamente las funciones encargadas de controlar las señales de entrada provenientes de los sensores para deteccion de obstaculos y LDR.

Depende de las variables y constantes globales definidas para el robot; por lo que para utilizar correctamente este módulo debe de haber incluido primero el módulo “global.h”.

==========

```
/**
 * @file Sensors.cpp
 * @author Ricardo Santiago Lopez (you@domain.com)
 * @version 0.1
 * @date 2022-05-18
 *
 * @details Archivo dedicado a la definicion de funciones que controlan sensores
 *
 * !! Este archivo debe de incluirse justo despues del encabezado global.h
 * Donde estan definidas varias constantes utilizadas aqui !!
 *
 */
#include <Arduino.h>

/**
 * @function: setup_sensors
 * @param: void
 * @return: void
 * @description: funcion dedicada a configurar los puertos ocupados
 *               por los sensores del robot
*/
void setup_sensors(){
  pinMode(CONTACT_R, INPUT);
  pinMode(CONTACT_L, INPUT);
  pinMode(TRIGGER_F, OUTPUT);
  pinMode(ECHO_F, INPUT);
  pinMode(TRIGGER_B, OUTPUT);
  pinMode(ECHO_B, INPUT);
}

// Detector de fuente luminosa

/**
 * @brief
 *
 * @return true
 * @return false
 * @desciption: funcion que comprueba si el robot ya se encuentra en la meta
 */
bool isTheGoal(){

  if (analogRead(LDR_FRONT) >= highestThreshold)
    return true;
```

```
  else if (analogRead(LDR_RIGHT) >= highestThreshold)
    return true;
  else if (analogRead(LDR_BACK) >= highestThreshold)
    return true;
  else if (analogRead(LDR_LEFT) >= highestThreshold)
    return true;

  return false;

}

/**
 * @function: findGoal
 * @param: void
 * @return: int
 * @description: Se encarga del sensado de la intensidad de luz en
 * alguna de las 4 direcciones definidas. Al encontrar la direccion
 * con mayor intesidad de luz retorna un numero que representa esa direccion
*/

//TODO: Se puede mejorar esta funcion para detectar mas direcciones
int findGoal()
{

  double ldr_Front = 0.0;
  double ldr_Back = 0.0;
  double ldr_Left = 0.0;
  double ldr_Rigth = 0.0;

  ldr_Front = analogRead(LDR_FRONT); //blue
  ldr_Rigth = analogRead(LDR_RIGHT); //yellow
  ldr_Back = analogRead(LDR_BACK);  //orange
  ldr_Left = analogRead(LDR_LEFT);  //greesudon

  if ((ldr_Front > ldr_Rigth) && (ldr_Front > ldr_Left) && (ldr_Front > ldr_Back))
    return LIGHT_FRONT;
  else if ((ldr_Rigth > ldr_Front) && (ldr_Rigth > ldr_Left) && (ldr_Rigth > ldr_Back))
    return LIGHT_RIGHT;
  else if ((ldr_Left > ldr_Front) && (ldr_Left > ldr_Rigth) && (ldr_Left > ldr_Back))
    return LIGHT_LEFT;
  else if ((ldr_Back > ldr_Front) && (ldr_Back > ldr_Rigth) && (ldr_Back > ldr_Left))
    return LIGHT_BACK;
  else
    return -1; //Error en la lectura;
}

// SENSADO DE DISTANCIAS

//Distancia del sensor ultrasonico
/**
 * @name: sensor_Sonar
 * @params: int: puerto Trigger, int: puerto Echo
 * @return: double: distance
 * @description: se reciben los pines del los puertos Trigger y Echo para ubicar al sensor ultrasonico
 * frontal o trasero para activarlo ,leer datos y calcular la distancia, la cual se retorna
*/

double sensor_Sonar(int Trigger, int Echo)
```

```
{
  //front
  double distance;
  long tiempo;

  //sound Sensors
  digitalWrite(Trigger, LOW); //signal to active the sensor
  delayMicroseconds(2);
  digitalWrite(Trigger, HIGH); //signal to active the sensor
  delayMicroseconds(10);
  digitalWrite(Trigger, LOW);
  tiempo = (pulseIn(Echo, HIGH));
  distance = float((tiempo / 2) / 29);

  return distance;
}

//Distancias de los sensores Intfrarrojos
/**
 * @name: read_IR
 * @param: int: muestreo
 * @param: int: pin
 * @return: double: Distancia_cm
 * @description: Se recibe un valor entero, el cual representa una taza de muestreo con la cual
 * se utilizara para implementar un filtro de media y reducir el ruido en la lectura del sensor
 * infrarrojo. El valor generado por ese filtro se utilizara para calcular las distancias en cm
 * y se retorna
*/

long read_IR(int muestreo, int pin)
{
  int suma = 0;
  for (int i = 0; i < muestreo; i++)
  {
    suma = suma + analogRead(pin);
  }
  float adc = (suma / muestreo) * 0.0048828125;
  float Distancia_cm = 13 * pow(adc, -1);

  //se asegura que siempre sean valores positivos
  //si baja a menos de cero, hay algun fallo electrico
  if (Distancia_cm < 0)
    return -1 * (Distancia_cm);

  //comodin para evitar malas lecturas
  if (Distancia_cm > 10000)
    return 10000;

  return Distancia_cm;
}
```

==========

Apéndice C

Código Arduino para control de las ruedas

El archivo odometric.cpp ubicado en la ruta "*Baltazar_robot/Arduino_files/Baltazar/Modules/Odometric_Module*" del proyecto, contiene la definición de las funciones dedicadas a la aplicación de la odometría del robot y el accionamiento de los motores para los desplazamientos lineales y rotacionales del robot.

Las funciones definidas aquí dependen de las definiciones globales contenidas en el archivo global.h, por lo que deberá de incluirlo primero. También depende las funciones para el control PID declaradas en el archivo pid_control.cpp, que ya está incluido aquí.

```
==========================================================================
#include "./control_vel/pid_control.cpp";

// FUNCIONES DE LOS ENCODERS PARA EL SENSADO DE POSICIONES Y VELOCIDADES

// funciones que unicamente realizan el conteo de pulsos que generan los encoders en los motores

/**
 * @name leftEncoderEvent
 * @param: void
 * @return void
 *
 * @details incrementa las variables globales que guardan la posicion angular de la rueda
 * left_count_ticks -> acumulador de pulsos para calcular las distancias recorridas (reseteable)
 * left_position -> acumulador de pulsos para calcular la posicion actual de la rueda (no reseteable)
 */
void leftEncoderEvent()
{
  left_count_ticks++;
  left_position++;
}

/**
 * @name rightEncoderEvent
 * @param: void
 * @return void
 *
 * @details incrementa las variables globales que guardan la posicion angular de la rueda
 * right_count_ticks -> acumulador de pulsos para calcular las distancias recorridas (reseteable)
 * right_position -> acumulador de pulsos para calcular la posicion actual de la rueda (no reseteable)
 */
void rightEncoderEvent()
{
  right_count_ticks++;
  right_position++;
}

/**
 * @function: metersToTicks
 * @param: float: meters
 * @return: int: pulsos
 * @description: Recibe la distancia en metros que debe de avanzar la rueda del robot, y
```

```
 * genera el numero de pulsos que debe de contar el encoder para avanzar la distancia deseada.
 *
 */
int mtsToTicks(float meters)
{

  int pulsos = 0;

  if (meters == 0)
    return 0;

  pulsos = (int)((PULSOSPORGIRO * meters) / (2 * PI * (RADIORUEDA) / 100));
  // Serial.println(pulsos);
  return int(pulsos/1.75);
}

/**
 * @function: degToTicks
 * @param: float: angulo
 * @return: int: _ticks
 * @description: los grados sexagecimales que debe de rotar el robot
 * y los comvierte a pulsos para que se pueda controlar desde el giro de las ruedas
 *
 */
int degToTicks(float angulo)
{
  int _ticks = (int)(0.66667 * angulo); // el valor 0.66666667 es un valor obtendio experimentalmente
  return 2.5 * _ticks;
}

/**
 * @function: linearDisp
 * @param: int ticks
 * @param: int f_dir
 * @return: void
 * @description: Toma el numero de pulsos que debe de contar en el encoder, y la bandera que le
indica
 * si el robot avanza 0, o retrocede 1
 * direccion de giro de las ruedas
 + LOW, HIGH -> avanza
 + HIGH, LOW -> retrocede
*/
void linearDisp(int ticks, int f_dir = 0)
{

  left_count_ticks = 0;
  right_count_ticks = 0;

  while (right_count_ticks <= ticks || left_count_ticks <= ticks)
  {
    _sample_time = millis();
    _delta_time = (double)_sample_time - _last_sample_time;

    digitalWrite(D_MRIGTH, f_dir);
    digitalWrite(D_MLEFT, !f_dir);
    analogWrite(V_MLEFT, PWM_MOTOR_L);
    analogWrite(V_MRIGTH, PWM_MOTOR_R);
```

```
    if (_delta_time >= TIEMPO_MUESTREO)
    {

      double vl_r = getVel_R();
      double vl_l = getVel_L();

      //aplicacion del control de velocidad en las ruedas
      PWM_MOTOR_R = PID_Right(VEL_REF, vl_r);
      PWM_MOTOR_L = PID_Left(VEL_REF, vl_l);

      //actualiza el muestre de tiempo
      _last_sample_time = _sample_time;

    }
  }
}

/**
 * @function: turnDegR
 * @param: int: pulsos
 * @return: void
  * @description: Toma los la cantidad de pulsos que representan los grados que debe rotar hacia la
derecha
 * el Robot y los aplica en ambas ruedas haciendolas girar en sentidos opuestos.
 * direccion de giro de las ruedas
 + HIGH, HIGH -> giro a la derecha
 + LOW, LOW -> giro a la Izquierda
*/
void turnDeg(int pulsos, int f_dir = HIGH)
{
  left_count_ticks = 0;
  right_count_ticks = 0;
  int tope = pulsos;

  while (right_count_ticks <= tope)
  {
    _sample_time = millis();
    _delta_time = (double)_sample_time - _last_sample_time;

    digitalWrite(D_MRIGTH, f_dir);
    digitalWrite(D_MLEFT, f_dir);
    analogWrite(V_MRIGTH, PWM_MOTOR_R);
    analogWrite(V_MLEFT, PWM_MOTOR_L);

    if (_delta_time >= TIEMPO_MUESTREO)
    {

      double vl_r = getVel_R();
      double vl_l = getVel_L();

      //aplicacion del control de velocidad en las ruedas
      PWM_MOTOR_R = PID_Right(VEL_REF, vl_r);
      PWM_MOTOR_L = PID_Left(VEL_REF, vl_l);

      //actualiza el muestre de tiempo
      _last_sample_time = _sample_time;

    }
```

```
  }
}


/**
 * @function: stop_Robot
 * @param: void
 * @return: void
 * @description: Aplica las instrucciones que permite que el robot se detenga.
 *
 */
void stop_Robot()
{
  reset_Values_PID();
  analogWrite(V_MLEFT, M_STOP);
  analogWrite(V_MRIGTH, M_STOP);
  delay(100);
}


//funcion para configuracion de los motores
void setup_motors()
{
  // configuracion de los pines de interrucion para los encoders
  attachInterrupt(digitalPinToInterrupt(LH_ENCODER_A), leftEncoderEvent, FALLING);
  attachInterrupt(digitalPinToInterrupt(RH_ENCODER_A), rightEncoderEvent, FALLING);

  // configuracion de los pines de entrada y salida
  pinMode(D_MRIGTH, OUTPUT);
  pinMode(D_MLEFT, OUTPUT);
}
```

==========

Apéndice D.

Código Arduino para aplicación del control PID

El archivo PID control ubicado en la ruta "*Baltazar_robot/Arduino_files/Baltazar/Modules/Odometric_Module/control_vel/*" del proyecto, define las funciones encargadas de monitorear las velocidades de las ruedas del robot, así como las funciones que aplican el control PID para dichas velocidades.

Las funciones desarrolladas aquí dependen de las definiciones globales en global.h, pero al estar este archivo ya incluido en odometric.cpp, basta con que el archivo global se incluya antes que este en scketch principal.

```
=============================================================================
/*********************************************************
 *  Definicion de Funciones dedicadas al control PID de  *
 *  las ruedas del Robot.                                *
 *********************************************************/

//Calculo de velocidades de las ruedas del robot
/**
 * @function: get_speed_R
 * @params: none
 * @return: double -> la velocidad medida en cm/s
 * @description: A partir de los valores sensados y guardados en los registros de posicion
 * y el tiempo de muestreo, Se calcula la velocidad de giro de las ruedas del motor medida en cm/s
*/
double getVel_R()
{

  //calcular el diferencial de posicion
  right_delta_position = right_position - right_last_position;
   right_last_position = right_position;                          //guardamos el registro de las posiciones de la rueda
  double frecuencia = (1000) / TIEMPO_MUESTREO;                   //muestreos en un segundo
  double wr = ((2 * PI * right_delta_position) / PULSOSPORGIRO) * frecuencia; //velocidad angular de la rueda r/s
  return RADIORUEDA * wr;                                         //velociad lineal de la rueda en cm/s
}

/**
 * @function: get_speed_L
 * @params: none
 * @return: double -> la velocidad medida en cm/s
 * @description: A partir de los valores sensados y guardados en los registros de posicion
 * y el tiempo de muestreo, Se calcula la velocidad de giro de las ruedas del motor medida en cm/s
*/
double getVel_L()
{

  //calcular el diferencial de posicion
  left_delta_position = left_position - left_last_position;
   left_last_position = left_position;                            //guardamos el registro de las posiciones de la rueda
  double frecuencia = (1000) / TIEMPO_MUESTREO;                   //muestreos en un segundo
```

```
  double wr = ((2 * PI * left_delta_position) / PULSOSPORGIRO) * frecuencia; //velocidad angular de la rueda r/s
  return RADIORUEDA * wr;                                   //velociad lineal de la rueda en cm/s
}

/**
 * @function: PID_Right
 * @params: vdes [velocidad deseada para el robot], vmed [velocidad medida por los encoders]
 * @description: toma de referencia la velocidad que se desea alcanzar en cm/s, y tambien recibe
 * la velodcidad medida en tiempo real; a partir de esos datos, se implementa un control de velodad PID,
 * utilizando las ecuaciones propuestas y los Coeficientes definidos en el encabezado del programa
*/
int PID_Right(double vdes, double vmed)
{
  //corrimiento de los registros de error para el motor derecho;
  errRight[2] = errRight[1];
  errRight[1] = errRight[0];
  errRight[0] = vdes - vmed;

  //calculo de las componentes del control PID
  double proporcional = _kpv * (errRight[0] - errRight[1]);
  double integral = _kiv * ((errRight[0] + errRight[1]) / 2);
  double diferencial = _kpv * (errRight[0] - (2 * errRight[1]) + errRight[2]);

  //corrimiento de los registros de valores calculados para el control PID
  _Rd[1] = _Rd[0];
  _Rd[0] = _Rd[1] + (proporcional + integral + diferencial);

  /**
    * Se aplica un tope al pwm para mantenerlo en un rango y no meter
    * datos indeseables
   */
  if (_Rd[0] > MAX_PWM_VAL)
    _Rd[0] = MAX_PWM_VAL;

  if (_Rd[0] < MIN_PWM_VAL)
    _Rd[0] = MIN_PWM_VAL;

  return _Rd[0];
}
/**
 * @function: PID_Left
 * @params: vdes [velocidad deseada para el robot], vmed [velocidad medida por los encoders]
 * @description: toma de referencia la velocidad que se desea alcanzar en cm/s, y tambien recibe
 * la velodcidad medida en tiempo real; a partir de esos datos, se implementa un control de velodad PID,
 * utilizando las ecuaciones propuestas y los Coeficientes definidos en el encabezado del programa
*/
int PID_Left(double vdes, double vmed)
{
  //corrimiento de los registros de error para el motor derecho;
  errLeft[2] = errLeft[1];
  errLeft[1] = errLeft[0];
  errLeft[0] = vdes - vmed;

  //calculo de las componentes del control PID
  double proporcional = _kpv * (errLeft[0] - errLeft[1]);
```

```
  double integral = _kiv * ((errLeft[0] + errLeft[1]) / 2);
  double diferencial = _kpv * (errLeft[0] - (2 * errLeft[1]) + errLeft[2]);

  //corrimiento de los registros de valores calculados para el control PID
  _Ri[1] = _Ri[0];
  _Ri[0] = _Ri[1] + (proporcional + integral + diferencial);

  /**
    * Se aplica un tope al pwm para mantenerlo en un rango y no meter
    * datos indeseables
   */
  if (_Ri[0] > MAX_PWM_VAL)
    _Ri[0] = MAX_PWM_VAL;

  if (_Ri[0] < MIN_PWM_VAL)
    _Ri[0] = MIN_PWM_VAL;

  return _Ri[0]; //retorna respuesta actual
}


void reset_Values_PID()
{
  left_position = 0;
  left_delta_position = 0;
  left_last_position = 0;
  right_position = 0;
  right_last_position = 0;
  right_delta_position = 0;
  _sample_time = 0;
  _last_sample_time = 0;
  _delta_time = 0;
  PWM_MOTOR_L = 60;
  PWM_MOTOR_R = 60;
  errLeft[0] = errLeft[1] = errLeft[2] = 0;
  errRight[0] = errRight[1] = errRight[2] = 0;
  _Rd[0] = _Rd[1] = 0.0;
  _Ri[0] = _Ri[1] = 0.0;
}
```

==========

Apéndice E.

Código Arduino para comportamiento del robot

El archivo Baltazar.ino ubicado en la ruta "*Baltazar_robot/Arduino_files/Baltazar/*" del proyecto, es el scketch principal y contiene la definición de las funciones que representan las diferentes capas de la AFSM que representa el comportamiento del robot.

Aquí se ejecutan las funciones principales de Arduino que son setup() y loop(), donde a su vez se incluyen y ejecutan los módulos desarrollados y mostrados en los apéndices anteriores.

```
==========================================================================
#include "Modules/Global/global.h"
#include "Modules/Sensors_modules/Sensors.cpp"
#include "Modules/Odometric_Module/Odometric.cpp"

void setup()
{
  Serial.begin(9600);
  setup_sensors();
  setup_motors();
}

/**
 * @function: turnAtGoal
 * @param: int
 * @return: void
 * @description: recibe un numero entero que representa una direccion devuelta por findGoal,
 * para indicarle al robot que rutina debe de ejecutar para asegurarse de quedar viendo
 * de frente la meta
 */
void turnAtGoal(int direccion)
{
  switch (direccion)
  {
  case LIGHT_FRONT:
    break;
  case LIGHT_RIGHT:
    turnDeg(degToTicks(R_ANGLE), HIGH); //giro a la derecha
    stop_Robot();
    break;
  case LIGHT_LEFT:
    turnDeg(degToTicks(R_ANGLE), LOW); //giro a la izquierda
    stop_Robot();
    break;
  case LIGHT_BACK:
    turnDeg(degToTicks(S_ANGLE), HIGH); //media vuelta
    stop_Robot();
    break;
  default:
    break;
  }
}
```

```
/**
 * @name evasor
 *
 * @param state: int
 * @param mts: double
 * @description: Representa la capa de evasion de obstaculos.
 * hace uso de las funciones de odometra para hacer que el robot
 * ejecute una rutina de modo que consiga evadir un obstaculo detectado
 * en la direccion definida por state.
 * a parte utiliza la variable mts para establecer cuanto se va a dezplazar
 * En funcion de que tan grande sea el obstaculo
 */

void evasor(int state, double mts)
{

  bool canBack = sensor_Sonar(TRIGGER_B, ECHO_B) <= UMBRAL_DIST_HCSR04;

  switch (state)
  {
  case OBTS_FRONT:

    stop_Robot();

    if (canBack)
      linearDisp(mtsToTicks(BACKWARD_MTS), HIGH);

    turnDeg(degToTicks(ISFRONT), HIGH);
    linearDisp(mtsToTicks(FORWARD_MTS), LOW);

    stop_Robot();

    break;
  case OBTS_RIGHT:

    stop_Robot();

    if (canBack)
      linearDisp(mtsToTicks(BACKWARD_MTS), HIGH);

    turnDeg(degToTicks(ISASIDE), LOW);
    linearDisp(mtsToTicks(FORWARD_MTS), LOW);

    stop_Robot();
    break;
  case OBTS_LEFT:

    stop_Robot();

    if (canBack)
      linearDisp(mtsToTicks(BACKWARD_MTS), HIGH);

    turnDeg(degToTicks(ISASIDE), HIGH);
    linearDisp(mtsToTicks(FORWARD_MTS), LOW);

    stop_Robot();
    break;
```

```
  default:
    stop_Robot();
    break;
  }
}

void loop()
{

  //capa de supervision
  //solo se ejecuta cuando llega a la meta
  while(isTheGoal()){
     stop_Robot();
  }

  // el robot voltea a la meta
  // capa de dirigirse a la meta
  turnAtGoal(findGoal());


  // Busca obstaculos y en caso de enontrar alguno, ejecuta el evasor
  // capa de evasion

  switch (_obts_state)
  {
  case CHECK_FRONT:

    if (digitalRead(CONTACT_R) && digitalRead(CONTACT_L))
      evasor(OBTS_FRONT, OBST_L);
        else if (read_IR(sample_ir, IR_R) <= UMBRAL_DIST_IRSENSOR && read_IR(sample_ir, IR_L) <=
UMBRAL_DIST_IRSENSOR)
      evasor(OBTS_FRONT, OBST_M);
    else if (sensor_Sonar(TRIGGER_F, ECHO_F) <= UMBRAL_DIST_HCSR04)
      evasor(OBTS_FRONT, OBST_S);

    _obts_state = CHECK_RIGHT;

    break;

  case CHECK_RIGHT:

    if (digitalRead(CONTACT_R))
      evasor(OBTS_RIGHT, OBST_S);
    else if (read_IR(sample_ir, IR_R) < UMBRAL_DIST_IRSENSOR)
      evasor(OBTS_RIGHT, OBST_S);

    _obts_state = CHECK_LEFT;

    break;

  case CHECK_LEFT:

    if (digitalRead(CONTACT_L) == 1)
      evasor(OBTS_LEFT, OBST_S);
    else if (read_IR(sample_ir, IR_L) < UMBRAL_DIST_IRSENSOR)
      evasor(OBTS_LEFT, OBST_S);

    _obts_state = CHECK_FRONT;
```

```
    break;

  default:
    _obts_state = CHECK_FRONT;
    break;
  }

  linearDisp(mtsToTicks(0.01), LOW); //naveacion libre

}
```

==========

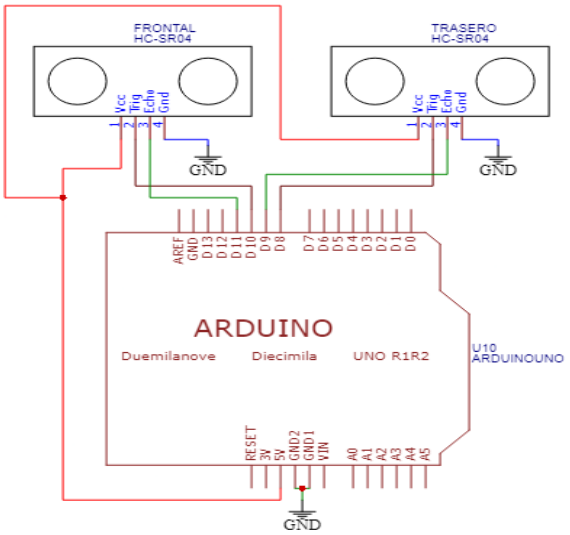
Apéndice F

Esquemas de conexión del Robot Baltazar

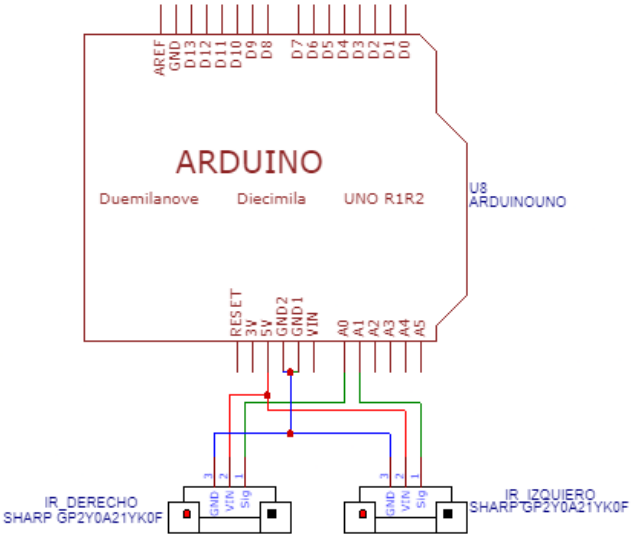
A continuación se muestran los esquemas de conexiones eléctricas diseñados para este proyecto.

Conexión de Sensores.

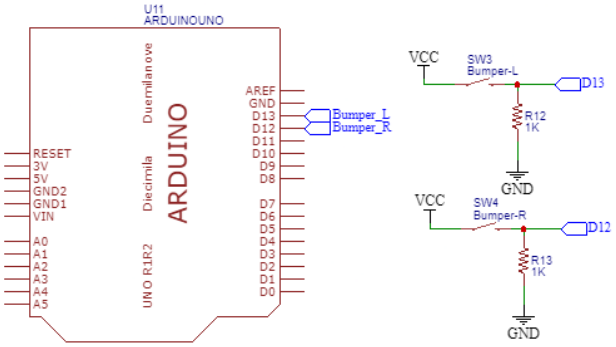
HC-SR04



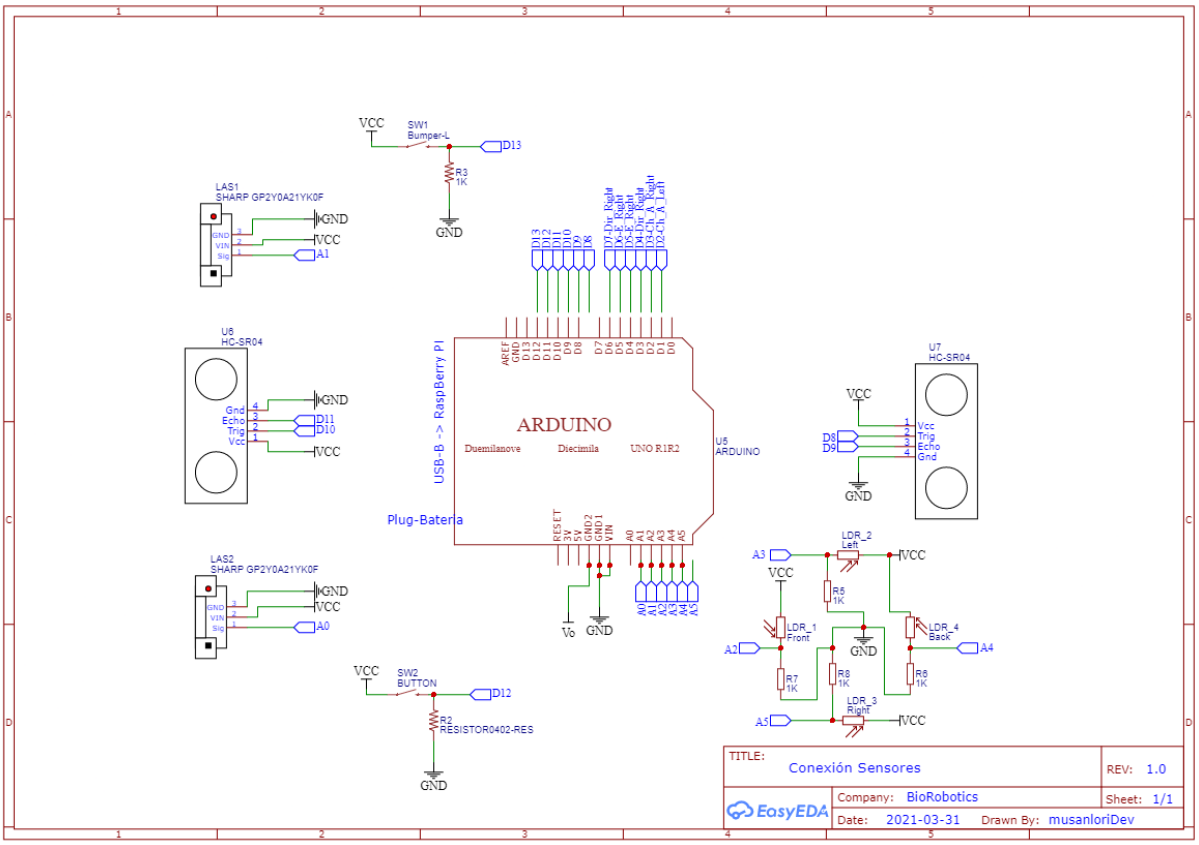
Sensores Infrarrojos



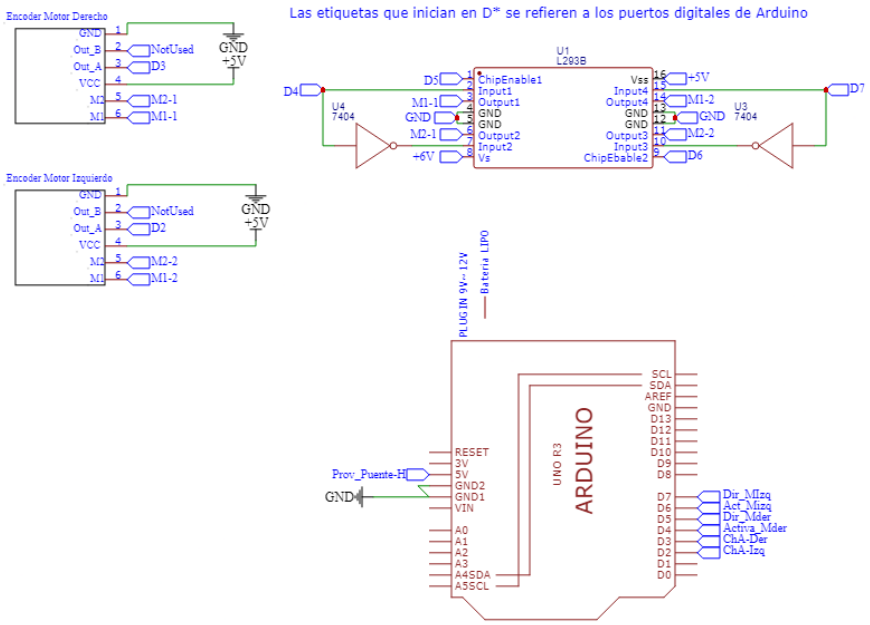
Sensores de Contacto



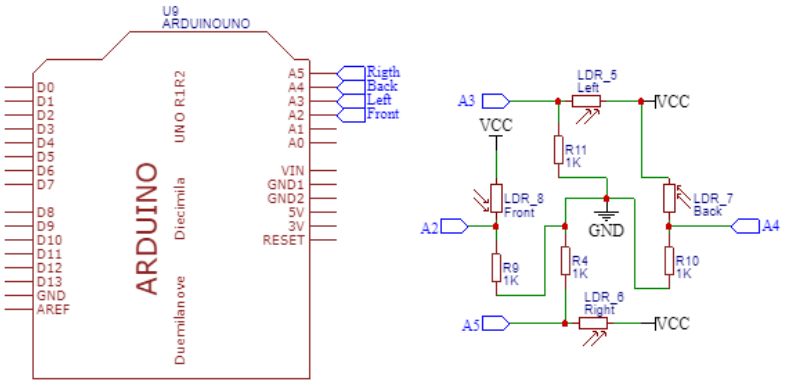
Integración de los sensores al robot



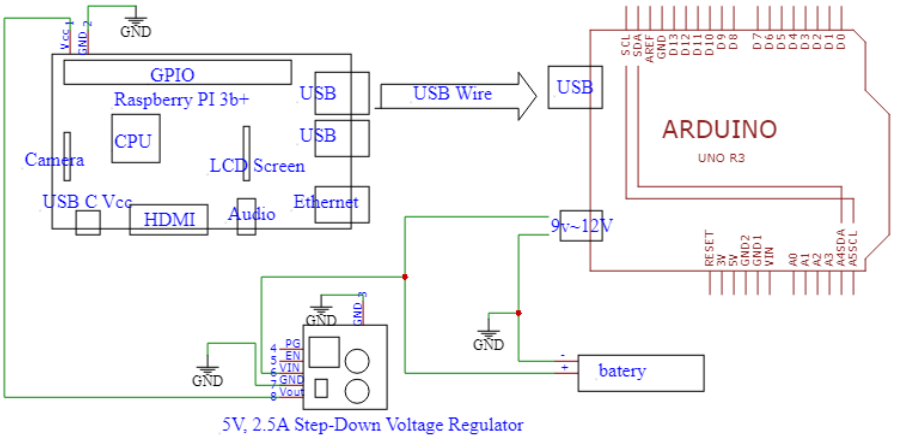
Conexión de los encoders, puente H y arduino



Conexión de la torre de sensores de luz



Comunicación Arduino Raspberry



Bibliografía y Referencias

1. Barrientos A, Peñin L, Balaguer C & Aracil R. (1997). *Fundamentos de Robótica*. Aravaca, Madrid: McGraw Hill.

2. Ollero Batorune A. (2001). *Robótica, manipuladores y robots móviles*. Barcelona, España: Marcombo.

3. Kumar Saha S. (2010). *Introducción a la robótica*. México, DF: McGraw Hill.

4. Mihelj M, Bajd T, Ude A, Lenarcic J, Stanovnik A, Munih M, Rejc J & Slajpah S. (2010). *Robotics*. Second Edition. Cham, Switzerland: Springer.

5. Jones, Joseph L., Seiger, Bruce A. & Flynn, Anita M.. (1998). *Mobile Robots: Inspiration to Implementation*, Second Edition. : A K Peters/CRC Press.

6. Zavala G. (2007). Robótica-1a ed.. *User Express*, 29, pp 16-92.

7. Arduino. Arduino documentation https://www.arduino.cc/reference/es/ , 2021

8. Torrente O.. (2013). *Arduino. Curso práctico de formación.* México, DF: Alfaomega.

9. Raspberry PI. (2021). *Raspberry Pi Documentation*. 2021, de Raspberry PI Sitio web: https://www.raspberrypi.com/documentation/computers/getting-started.html (Setting up your Raspberry Pi)

10. Raspberry PI. (2021). *Raspberry Pi Documentation.* 2021, de Raspberry PI Sitio web: https://www.raspberrypi.com/documentation/computers/getting-started.html (Installing the Operating System)

11. Raspberry PI. (2021). *Raspberry Pi Documentation*. 2021, de Raspberry PI Sitio web: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#raspberry-pi-3-model-b-2 (specifications)

12. ROS. (2021). *ROS Tutorials*. 2021, de ROS.org Sitio web: *http://wiki.ros.org/ROS/Tutorials*

13. Pololu. Pololu Documentation https://www.pololu.com/product/1101 (microreductor)

14. Issac. *HC-SR04: todo sobre el sensor de ultrasonidos*. Sitio web: https://www.hwlibre.com/hc-sr04/

15. Llamas, Luis. *¿CÓMO FUNCIONA EL SHARP GP2Y0A02YK0F?* junio de 2016, Sitio web: https://www.luisllamas.es/arduino-sharp-gp2y0a02yk0f/

16. Sharp. *gp2y0a21yk datasheet*. Enlace: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf

17. Naylamp Mechatronics SAC. TUTORIAL SENSOR DE DISTANCIA SHARP. Sitio web: https://naylampmechatronics.com/blog/55_tutorial-sensor-de-distancia-sharp.html

18. Savage, Jesus. (2020). *Lección 2: Comportamientos reactivos* [Diapositiva de PowerPoint]. enlace web. https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/robots_moviles/2020_2/lecciones/robots_moviles_2020_2_leccion_2.pdf

19. Savage, Jesus (2020). *Lección 13: Cinemática de un robot Móvil* [Diapositiva de PowerPoint]. enlace web: https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/robots_moviles/2020_2/lecciones/Leccion13_robots_moviles_2020_2.pdf

20. Savage, Jesus. (2020). *Lección 14: Control de un robot Móvil*.[Diapositiva de PowerPoint]. Enlace web. https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/robots_moviles/2020_2/lecciones/Leccion14_robots_moviles_2020_2.pdf

21. TutosIngenieria. (2019). *Carro ARDUINO y la necesidad del modelo matemático (Vídeo #2)*. [Archivo de Vídeo]. https://www.youtube.com/watch?v=XWApJxz9laM&list=PLiJv_3SD9kXBv5ecDxioEtX-Iu5HPcyhS&index=2

22. TutosIngenieria. (2019). *MODELO matemático de CARRO diferencial con ARDUINO (Vídeo #3)*. [Archivo de Vídeo]. https://www.youtube.com/watch?v=XWApJxz9laM&list=PLiJv_3SD9kXBv5ecDxioEtX-Iu5HPcyhS&index=3

23. TutosIngenieria. (2019). *Odometría y encoder, medir POSICIÓN de robot con ARDUINO (Vídeo #5)*. [Archivo de Vídeo]. https://www.youtube.com/watch?v=XWApJxz9laM&list=PLiJv_3SD9kXBv5ecDxioEtX-Iu5HPcyhS&index=5

24. TutosIngenieria. (2019). *Realimentación de un SISTEMA de control, ROBOT con ARDUINO (Vídeo #6)*. [Archivo de Vídeo]. https://www.youtube.com/watch?v=XWApJxz9laM&list=PLiJv_3SD9kXBv5ecDxioEtX-Iu5HPcyhS&index=6

25. Castaño, Sergio. *Control PID – Acción de Control Proporcional*. Sitio web: https://controlautomaticoeducacion.com/control-realimentado/control-pid-accion-proporcional/

26. Castaño, Sergio. *Acción de Control Integral – Control PID*. Sitio web: https://controlautomaticoeducacion.com/control-realimentado/accion-de-control-integral-control-pid/

Referencias de Imágenes

1. Figura extraída de: Ollero Batorune A. (2001). *Robótica, manipuladores y robots móviles*. página 17
2. Figura extraída de: Ollero Batorune A. (2001). *Robótica, manipuladores y robots móviles*. página 28
3. Figura extraída de: Ollero Batorune A. (2001). *Robótica, manipuladores y robots móviles*. página 30
4. Figura extraída de: https://i.ytimg.com/vi/pl7-gIvl1o8/hqdefault.jpg
5. Figura extraída de: https://i.blogs.es/d540f2/spotmini1/1366_2000.jpg
6. Figura extraída de:: https://store.arduino.cc/usa/arduino-uno-rev3
7. Figura extraída de:: https://aprendiendoarduino.wordpress.com
8. Figura extraída de: https://components101.com/microcontrollers/atmega328p-pinout-features-datasheet
9. Figura extraída de: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus
10. Figura extraída de: https://sandorobotics.com/producto/2858/
11. Figura extraída de: Pololu, robotics and electronics https://www.pololu.com/product/1101
12. Figura extraída de: https://grupoelectrostore.com/wp-content/uploads/2020/09/0J1131.1200-1.jpg
13. Figura extraída de: Pololu Robotics and Electronics, https://www.pololu.com/product/2598
14. Figura extraída de: Pololu Robotics and Electronics. https://www.pololu.com/product/2598
15. Figura extraída de: https://www.luisllamas.es/salidas-analogicas-pwm-en-arduino/
16. Figura extraída de: https://www.hwlibre.com/wp-content/uploads/2019/08/hc-sr04.jpg.webp
17. Figura extraída de: https://www.luisllamas.es/wp-content/uploads/2016/06/arduino-sharp-GP2Y0A02YK0F1-esquema.png
18. Figura extraída de: https://www.luisllamas.es/wp-content/uploads/2016/06/arduino-sharp-GP2Y0A02YK0F1-funcionamiento.png
19. Figura extraída de: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf
20. Figura extraída de: https://www.semiconductorforu.com/wp-content/uploads/2017/08/ldr.jpg
21. Figura extraída de: https://sandorobotics.com/producto/prt-11856/
22. Figura extraída de: https://sandorobotics.com/producto/67002/
23. Figura extraída de: https://shop.mitutoyo.eu/pim/upload/mitutoyoData/image/bigweb/201381_a_tif.png