Universidad del Azuay

Facultad de Ciencia y Tecnología

Escuela de Ingeniería Electrónica

"DISEÑO Y CONSTRUCCIÓN DE UN ROBOT PARA MAPEO Y EXPLORACIÓN DE MINAS SUBTERRÁNEAS"

Trabajo de graduación previo a la obtención
del título de Ingeniero Electrónico

Autores:
Andrés Patricio Cabrera Flor
Gabriel Alfonso Delgado Oleas

Director:
Hugo Marcelo Torres Salamea

Cuenca, Ecuador
2014

DEDICATORIA

A mis padres y mis abuelos, los que me han enseñado a superarme un día a la vez y que, a pesar de todas las dificultades, han ayudado inmensamente a que esto haya sido posible.

Andrés Cabrera.

DEDICATORIA

Este trabajo va dedicado a mis padres Carlos y Nancy por haberme apoyado incondicionalmente en el camino de mis estudios. A mi hermana Kaly por haberme dado el gran ejemplo de lucha y sacrificio. A mi hermana Ximena quien ha sido mi permanente motivación

Gabriel.

AGRADECIMIENTO

Agradezco a Dios por haberme permitido llegar más lejos de lo que hubiera imaginado. A mi gran amigo Gabriel Delgado, ya que sin él nada de esto hubiera sido posible. A Paulina, por apoyarme, comprenderme y estar conmigo en todo momento. Agradezco a la Universidad del Azuay, en especial a los distinguidos profesores de la Escuela de Ingeniería Electrónica, en especial al Dr. Hugo Torres por apoyar proyectos innovadores.

Andrés Cabrera

AGRADECIMIENTO

Agradezco a Dios quien me dio la vida y la salud, a mi amigo PATO quien me ha enseñado que la vida hay que construirla día a día, a mis amigos de la Universidad con quienes he compartido muchísimos momentos de estudio, de cansancio, de risas… A la Junta académica de Ingeniería Electrónica (Ing. Santiago Mora, Lcdo. Leopoldo Vázquez, Ing. Francisco Vázquez), al Decano de la Facultad de Ciencia y Tecnología Ing. Germán Zúñiga, quienes nos han brindado su mano amiga y su apoyo incondicional. Un agradecimiento especial al Dr. Hugo Torres quien nos ha orientado en los procesos seguidos en este proyecto, que a más de ser un profesor ha llegado a ser un gran amigo.

Gabriel.

RESUMEN

DISEÑO Y CONSTRUCCIÓN DE UN ROBOT PARA MAPEO Y EXPLORACIÓN DE MINAS SUBTERRÁNEAS

La presente tesis tiene como finalidad desarrollar un robot móvil capaz de obtener informaciń del entorno de una mina a través de un conjunto de sensores.Esta informaciń se envía a un computador mediante un enlace Wi-Fi para recrear el mapa en 2 dimensiones de la mina, y mostrar datos importantes como temperatura, humedad, gas metano y video en tiempo real.

Para el control, adquisición de datos de los sensores y comunicaciones del robot se han implementado tarjetas de microcontroladores con distintas prestaciones. Estas tarjetas se han configurado para formar un solo sistema autónomo y compacto dentro del robot.

Palabras clave: Robot, microcontrolador, Arduino, LabVIEW, mapas, minas, adquisición de datos.

Dr. Hugo Torres Salamea
Director

Ing. Francisco Vásquez Calero
Director de Escuela

UNIVERSIDAD DEL AZUAY
ESCUELA Ingeniería Electrónica

Andrés Cabrera Flor
Autor

Gabriel Delgado Oleas
Autor

ABSTRACT

DESIGN AND CONSTRUCTION OF A ROBOT FOR MAPPING AND EXPLORATION OF UNDERGROUND MINES

This thesis aimed to develop a mobile robot able to obtain information from a mine environment by a set of sensors. This information is sent to a computer using a Wi-Fi link to recreate a 2 dimensional mine map, and to show important data such as temperature, humidity, methane and video in real time.

For robot´s control logic, data sensors acquisition and data communications, microcontroller boards with different specs are implemented. These boards are configured to form a single self-contained, compact system within the robot.

Key Words: Robot, Microcontroller, Arduino, LabVIEW, Maps, Mines, Data Acquisition.

Dr. Hugo Torres Salamea
Director

Ing. Francisco Vásquez Calero
School Director

UNIVERSIDAD DEL AZUAY
ESCUELA Ingeniería Electrónica

Andrés Cabrera Flor
Author

Gabriel Delgado Oleas
Author

UNIVERSIDAD DEL AZUAY
DPTO. IDIOMAS

Translated by
Lic. Lourdes Crespo

ÍNDICE DE CONTENIDOS

CAPÍTULO 2: DISEÑO MECÁNICO Y ANÁLISIS CINEMÁTICO

CAPÍTULO 3: DISEÑO ELECTRÓNICO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

Cabrera Flor Andrés Patricio
Delgado Oleas Gabriel Alfonso
Trabajo de Graduación
Dr. Hugo Marcelo Torres Salamea
Marzo 2014

DISEÑO Y CONSTRUCCIÓN DE UN ROBOT PARA MAPEO Y EXPLORACIÓN DE MINAS SUBTERRÁNEAS

INTRODUCCIÓN

En nuestro país, la minería es una actividad que se ha realizado a pequeña y gran escala, de manera regulada y artesanal. Recientemente se han realizado varios estudios que indican un gran potencial para esta actividad, en especial para extracción de oro, cobre y plata, localizándose los proyectos principales en las provincias de Azuay, Morona Santiago y Zamora Chinchipe. La mayoría de estos proyectos utilizan un método de excavación subterráneo, es decir, la creación de túneles y caminos bajo la tierra por donde circula maquinaria y personal para extraer el material.

El gran problema provocado por la minería realizada de manera artesanal en varias provincias del país es la gran cantidad de accidentes y derrumbes que han cobrado vidas humanas. Esta actividad artesanal ha creado en varias zonas del país gran cantidad de minas abandonadas, las cuales presentan un gran riesgo para el equipo humano que ingrese a ellas y para las poblaciones cercanas.

Las minas realizadas de manera artesanal permiten la entrada de carretillas pequeñas y personas, haciendo imposible la entrada de vehículos diseñados para túneles subterráneos de minas de gran escala. Además, muchas minas poseen una gran longitud y tienen muchas ramificaciones, lo que hace limitado el uso de sondas cableadas equipadas con cámaras y sensores para realizar una exploración que abarque todos los túneles.

El presente trabajo de grado tiene como objetivo la creación de un robot prototipo capaz de movilizarse dentro de un túnel, el cual, mediante los sensores colocados dentro del mismo, realice un mapa que describa los caminos, extensión y variables ambientales dentro de la mina, el cual podrá ser manejado de manera remota o movilizarse de manera autónoma. Para ello, se realizará una investigación previa sobre modelos de locomoción, sensores y controladores robóticos de proyectos similares, que posteriormente serán aplicados al prototipo. Por último, se aplicarán algoritmos para su adecuado comportamiento y se realizarán pruebas de funcionamiento.

CAPÍTULO 1

MARCO TEÓRICO

En este capítulo se describirán los robots móviles, en especial los de tipo terrestre que usan ruedas o dispositivos similares de locomoción. Se describirán varias de las principales características de los mismos, indicando cuáles son los mejores y más comúnmente utilizados para desplazarse en caminos no convencionales. Además, se realizará una investigación de robots móviles aplicados en la minería o exploración. También se tratará brevemente acerca de la actividad minera en nuestro país, en especial sobre minas subterráneas, los riesgos que implican para el personal que ingresa a las mismas, y las ventajas de un robot explorador para brindar información detallada que brinde mayor seguridad y evite accidentes innecesarios.

1.1. Robots Móviles

Un robot móvil es una máquina automática que es capaz de moverse en cualquier ambiente para el que fue diseñado. Al contrario de los robots industriales, los cuales generalmente consisten en brazos articulados sujetos a una superficie con actuadores en sus extremos, los robots móviles tienen la capacidad de moverse sin restricciones por cualquier locación.

En los últimos años, los robots móviles han sido el tema de investigación preferido en muchos laboratorios y universidades. Esto se debe a que son excelentes plataformas de enseñanza y aplicación de teorías y técnicas, teniendo además aplicaciones en gran cantidad de campos de la industria, la milicia, e incluso el hogar.

Los robots móviles pueden ser clasificados de acuerdo a:

Medio por el que se desplazan:

- Robots terrestres: tienen llantas, orugas o extremidades para caminar.
- Robots aéreos.
- Robots acuáticos.
- Robots polares: diseñados para moverse en el hielo o en la nieve.

Dispositivos que usan para moverse:

- Robots con extremidades: bípedos, cuadrúpedos, hexápodos, etc.
- Robots con ruedas, orugas, etc.
- Robots con hélices, alas, y muchos más.

Los robots con ruedas poseen varias ventajas con respecto a otros tipos de locomoción en tierra. Usualmente son diseñados para que todas o la mayoría de las ruedas estén en contacto con el suelo, proporcionando al robot un equilibrio casi constante. Ya que el equilibrio no es un problema, el campo de investigación en robots con ruedas se concentra en problemas de tracción, maniobrabilidad y control dentro de su entorno. Estos parámetros están fuertemente influenciados por el diseño, número y disposición de las ruedas. A continuación, se describen estos parámetros.

1.1.1. Estabilidad mecánica

La estabilidad mecánica del robot se refiere a la facilidad para mantener su equilibrio tanto estático como dinámico. En el caso de un robot con una sola rueda, el equilibrio estático no sería posible, y el equilibrio dinámico sería en extremo complejo. Aunque la mínima cantidad de ruedas para estabilidad estática y dinámica es dos (conocido como modelo de robot diferencial), el diseño práctico requiere el uso de tres ruedas (mínimo dos ruedas motorizadas y una rueda pivotante), lo que simplifica el control y permite implementar más funciones dentro del robot, colocando el centro de gravedad dentro del triángulo formado por las ruedas. Así, la estabilidad puede ser mejorada añadiendo más ruedas, tomando en cuenta que cuando el número de éstas aumenta, el robot requiere un sistema de suspensión, puesto que se aumenta su estabilidad estática y se torna más complejo iniciar el movimiento especialmente en terreno irregular (1).

1.1.2. Maniobrabilidad

La maniobrabilidad se refiere a la capacidad del robot para moverse en cualquier dirección a lo largo del plano del piso sin importar su orientación con respecto al eje vertical. Los robots con mayor desarrollo de esta capacidad usan ruedas especiales (ruedas omnidireccionales), o poseen una disposición de las ruedas que permite su rotación en su propio eje para cambiar su orientación. La maniobrabilidad es equivalente al control de los grados de libertad del robot.

1.1.3. Grados de libertad

Los grados de libertad de un robot móvil con ruedas son tres: posición en eje X, en eje Y, y ángulo u orientación del vehículo. Se conoce como grados de libertad diferenciables a los grados de libertad completamente controlables por el robot. Por ejemplo, un robot en forma de bicicleta no controla su orientación de manera directa. Por otro lado, un robot omnidireccional controla todos sus grados de libertad, y puede cambiar su posición en los ejes y su orientación de manera directa.

1.1.4. Controlabilidad

Existe una relación inversa entre la controlabilidad y la maniobrabilidad. Los robots omnidireccionales requieren un control más complejo de la velocidad de las ruedas para lograr un movimiento en cualquier dirección, ya que los grados de libertad de cada rueda implican mayor cantidad de variables a controlar. Diseños con menos maniobrabilidad permiten compensar su posición y dirección con algoritmos de control más simples, obteniendo mayor precisión en su ubicación.

1.1.5. Tipos de robots móviles con ruedas

Los tipos de ruedas y su disposición son muy variados dentro de los robots móviles. Estos parámetros se eligen tomando en cuenta los criterios de estabilidad, maniobrabilidad y controlabilidad necesarios para cada tipo de aplicación y entorno de movimiento del robot. En este apartado se analizarán los principales modelos utilizados en los robots móviles.

1.1.5.1. Modelo síncrono

Es un modelo especial basado en tres ruedas controladas por dos motores. El primer motor hace girar las tres ruedas, que siempre están direccionadas en el mismo sentido, para generar movimiento lineal de avance en línea recta. El segundo motor permite el cambio de dirección de las tres ruedas para orientar el cuerpo en los giros. Este robot se considera casi omnidireccional, aunque no puede realizar giros sin antes detenerse y orientar sus ruedas. Este modelo es eficiente en medio ambientes controlados y conocidos, donde el giro en tres ruedas y la orientación son posibles gracias a la regularidad del terreno y el conocimiento del entorno (2).

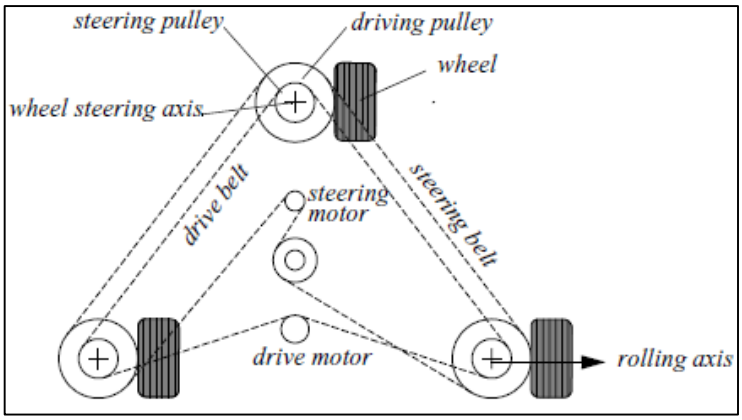


Figura 1.1 Esquema de un modelo síncrono. El robot puede moverse en cualquier dirección, pero su orientación no es controlable (1).

Figura 1.2 Robot síncrono Xenia de la Universidad de Kaiserslautern (2).

1.1.5.2. Modelo de Ackermann

Es el sistema común de los automóviles comerciales. Consiste en dos ruedas traseras con un mismo eje sin capacidad de giro, y dos ruedas delanteras que pueden girar en conjunto para cambiar la dirección del vehículo. El movimiento en línea recta es sencillo, pues consiste en bloquear la dirección delantera y dar tracción a las ruedas, es decir, la velocidad lineal está controlada de forma independiente de la velocidad angular. Sin embargo, su maniobrabilidad es limitada, puesto que para orientar el vehículo se requiere un radio mínimo de giro, y el posicionamiento exacto (por ejemplo, el parqueo de un automóvil) requiere varios movimientos hacia adelante y atrás. A pesar de esto, el modelo de dirección en las ruedas delanteras tiene una gran ventaja: su geometría le provee una estabilidad lateral especialmente en giros rápidos, característica importante en los automóviles comerciales (1).

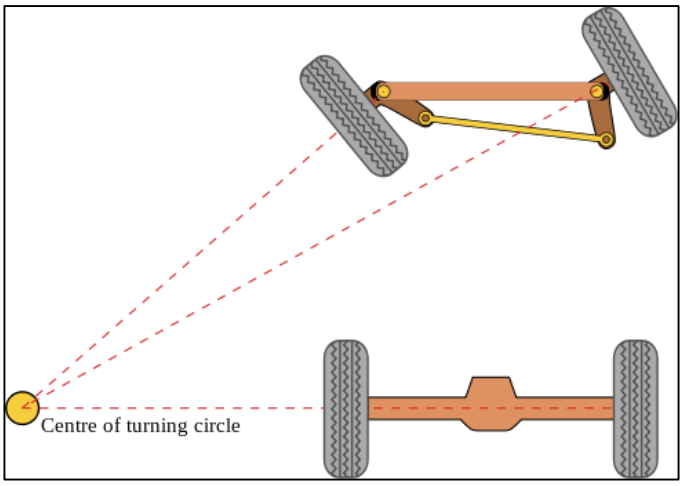


Figura 1.3 Esquema del modelo Ackermann básico.

Fuente: Wikipedia The Free Encyclopedia. *Ackermann steering geometry* [en línea]. [20 septiembre de 2013]. Disponible en web: <http://en.wikipedia.org/wiki/Ackermann_steering_geometry>.

1.1.5.3. Modelo omnidireccional

Se basan en el uso de ruedas omnidireccionales, conocidas como ruedas suecas. Los robots de este tipo usan tres o cuatro ruedas. Estos robots se consideran completamente holonómicos, es decir, que son capaces de modificar su dirección instantáneamente, sin necesidad de orientarse o rotar previamente. Sus algoritmos de control son complejos, está limitado a superficies planas y debe tomarse en cuenta los cambios de peso en el robot, puesto que su gran maniobrabilidad se basa en el rozamiento de sus ruedas especiales (1).

Figura 1.4 Robot Uranus con ruedas omnidireccionales.

Fuente: Carnegie Mellon University, School of Computer Science. “Uranus” [en línea]. Pittsburg, 1985 [20 septiembre de 2013]. Disponible en web: http://www.cs.cmu.edu/~gwp/robots/Uranus.html.

1.1.5.4. Modelo diferencial

Este diseño posee dos motores coaxiales a cada lado, los cuales están fijos a la estructura y mueven una rueda cada uno. Como se había mencionado en apartados anteriores, para una estabilidad aceptable se debe apoyar en una tercera o cuarta rueda sin tracción, las cuales pueden ser ruedas pivotantes o ruedas de tipo esfera. Este modelo es más sencillo en su diseño mecánico, ya que no incluye rotaciones en los ejes de las ruedas como el modelo de Ackermann. Sin embargo, su control es un tanto más complejo debido a la necesidad de sincronización de los motores de las ruedas (2).

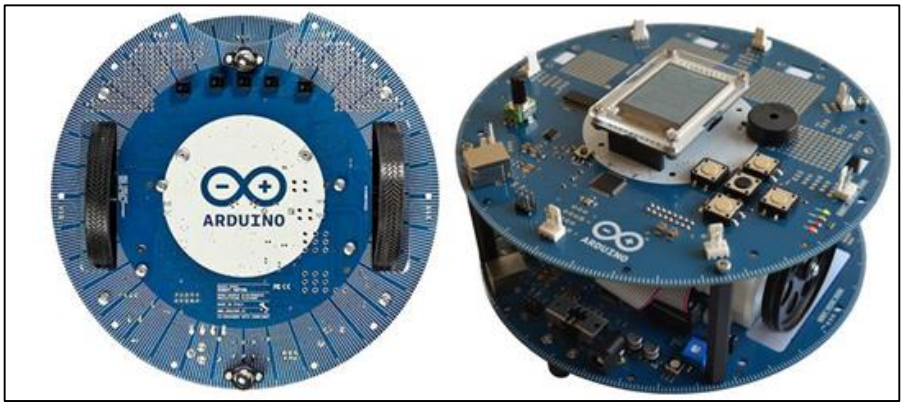


Figura 1.5 Robot Diferencial Arduino.

Fuente: Arduino. *Arduino Robot* [en línea]. [20 septiembre de 2013]. Disponible en web: <http://arduino.cc/en/Main/Robot >.

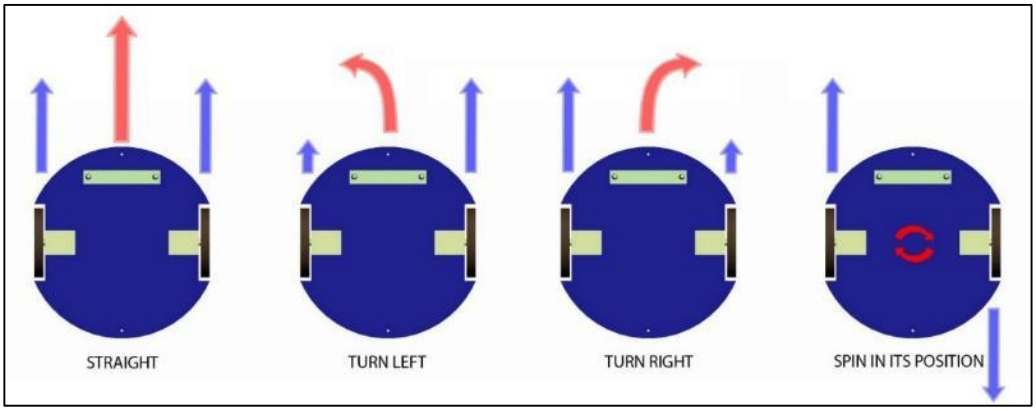


Figura 1.6 Representación de los movimientos posibles en un modelo diferencial

Fuente: I CREATOR. Transistor Based Line Following Robot-Mechanical Design [en línea]. 3 de octubre de 2010 [20 septiembre de 2013]. Disponible en web: <http://icreator.wordpress.com/category/line-following-robots/transistor-based-line-following-robot/>.

1.1.5.5. Modelo con orugas

Este modelo puede considerarse un caso especial del modelo diferencial. Se distingue por su mejor maniobrabilidad en terreno irregular, por lo cual es muy utilizado en vehículos agrícolas, tractores, excavadores y tanques. Su diseño más simple consta de dos motores independientes a cada lado, conectados a las orugas. Estas orugas son un conjunto de eslabones que aumentan la superficie de contacto de las ruedas permitiendo distribuir de una forma más equitativa el peso del vehículo. Este diseño evita que el vehículo quede atascado en superficies inestables, pudiéndose diseñar vehículos con un peso total mucho mayor a los diseñados con ruedas comunes. Sin embargo, el uso de orugas aumenta en gran cantidad la energía utilizada en los motores, y además produce gran dificultad e imprecisión en la realización de giros (1).

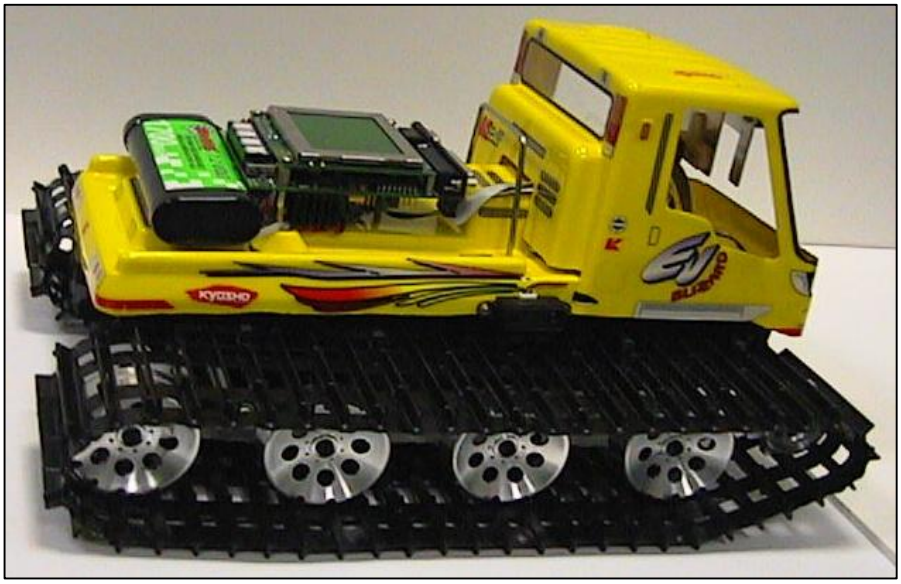


Figura 1.7 Robot EyeTrack, basado en el modelo de un tractor con orugas.

Fuente: Robotics & Automation Lab. *EyeTrack* [en línea]. [20 septiembre de 2013]. Disponible en web: < http://robotics.ee.uwa.edu.au/eyebot/doc/robots/eyetrack.html>.

1.1.6. Robots autónomos

Los robots autónomos son aquellos que pueden realizar tareas en ambientes variables sin la guía continua de un humano. Muchos tipos de robots tienen grados distintos de autonomía y pueden reflejarla de varias maneras: desde exploración espacial, hasta limpieza de pisos y tratamiento de aguas.

Algunos robots industriales modernos se denominan autónomos dentro de límites fijos de su entorno. En las más avanzadas industrias, donde existen ambientes inconstantes y se requieren manipular objetos varios, se crean condiciones especiales para sus acciones, basadas en la lectura de sensores de alta precisión. Sin embargo, cuando se habla de robots autónomos, se refiere principalmente al tipo de robots que son capaces de enfrentar cualquier obstáculo o cambio en el entorno para el cual fueron diseñados.

Tomando esto en cuenta, un robot autónomo debe tener la capacidad de:

- Obtener información de su entorno.
- Funcionar continua o periódicamente sin la intervención humana.
- Adaptarse al entorno de acuerdo a los cambios y a sus propias capacidades.

Para obtener información de su entorno, los robots están dotados de una gran variedad de sensores. Los objetivos de los sensores en un robot autónomo son la localización y navegación.

1.1.6.1. Localización

La localización implica el conocimiento de la posición y orientación en todo momento del robot. Por ejemplo, un robot de limpieza debe cubrir toda el área del piso sin repetir espacios o perderse, y para ello necesita varios sensores que le indiquen su localización en todo momento.

En ambientes externos es común la utilización del Sistema de Posicionamiento Global GPS (*Global Positioning System)*, en conjunto con otros sensores. En lugares cerrados controlados es posible colocar guías de posición en paredes, piso y techo, las cuales permiten ubicar al robot dependiendo de su posición relativa a estas guías. Sin embargo, en entornos no controlados estas soluciones no se aplican y se deben implementar sensores mucho más complejos como radares, sonares y brújulas. A pesar de un buen control y adquisición de datos de varios sensores, la localización exacta del robot es difícil de lograr, por lo que se realizan muchas pruebas y se aplican métodos probabilísticos para predecir con mayor exactitud la localización del robot (3).

Un ejemplo específico de un ambiente cerrado no controlado es un túnel de mina. Este puede ser modelado como un laberinto, del cual el robot no posee un mapa o guías para ubicarse. Su objetivo es el de explorar todos los posibles caminos y derivaciones del túnel, y salir del mismo basado en la información recopilada.

1.1.6.2. Mapeo

La creación de mapas de un ambiente desconocido es una tarea mucho más complicada, puesto que no puede ser separada del proceso de localización, ya que los errores en la localización se incorporan al mapa y viceversa, creando un problema conocido como SLAM (*Simultaneous Localization And Mapping*) (1) (4).

Un aspecto importante en la creación de mapas es determinar la cantidad de información que se requiere y que se tiene capacidad de procesar. Mapas en 3 dimensiones con información sobre obstáculos y medidas (por ejemplo de un lecho marino) requieren mucho más procesamiento que esquemas en dos dimensiones de un laberinto.

Otro aspecto a considerar es el problema de correspondencia, que consiste en la dificultad de determinar si las mediciones hechas por el sensor tomadas en distintos tiempos corresponden al mismo objeto físico. Esto se relaciona a la acumulación de errores de medición en el tiempo que da como consecuencia la dificultad del robot de ubicarse en su propio mapa.

El último aspecto lo componen los cambios del entorno. Algunos cambios son mínimos como nuevas construcciones en un edificio, pero otros son recurrentes y rápidos, como el movimiento de muebles, puertas, autos de un estacionamiento, escombros, etc. Este dinamismo del ambiente puede producir errores en la creación de mapas y en la medición de los sensores.

1.1.7. Robots móviles para la minería

En la actualidad existen muchos estudios para la creación de robots móviles autónomos o semiautónomos con la capacidad de ingresar a minas subterráneas. Esto se ha justificado debido a la gran cantidad de accidentes que se producen dentro de los túneles, y que no han disminuido a pesar de todas las precauciones y tecnología que se aplican actualmente, costando muchas vidas cada año. Es así que varias universidades e incluso empresas mineras mundiales como Anglo American[1] han invertido muchos recursos económicos, tecnológicos y humanos para la creación de maquinaria que pueda reemplazar al trabajador humano, especialmente en las tareas que implican mayor riesgo. Mucha de la tecnología utilizada para la creación de estas máquinas incluso se aplica en investigaciones aeroespaciales, debido a su alta complejidad.

Figura 1.8 Robot experimental para túneles de minería (*National Robotics Engineering Center*).
Fuente: Carnegie Mellon. *Anglo American Partner on Mining Robotics* [en línea]. 9 de enero de 2013. Disponible en web: <http://www.laserfocusworld.com/articles/2013/01/carnegie-mellon-and- anglo-american-plc -sign-agreement-to-develop-mining- robots/…/1357940721553.jpg>.

[1] *Carnegie Mellon and Anglo American PLC sign agreement to develop mining robots* [en línea]. Pittsburgh: John Wallace, 11 enero 2013 [8 de marzo de 2013]. Disponible en web: <http://www.laserfocusworld.com/articles/2013/01/carnegie-mellon-and-anglo-american-plc-sign-agreement-to-develop-mining-robots.html>.

Gran cantidad de estas investigaciones se basa, principalmente, en la creación de vehículos robustos de tamaño pequeño o mediano, los cuales tienen capacidades especiales en cuanto a su autonomía y exploración (5), puesto que las comunicaciones son limitadas en ambientes subterráneos. Su principal objetivo es recolectar la mayor cantidad de información sobre la mina, o realizar actividades extractivas, evitando que esta tarea tan peligrosa sea realizada por personas.

Un ejemplo de estas investigaciones es el robot *Groundhog* de la Universidad de Carnegie Mellon. Está construido con el objetivo de ingresar a minas abandonadas de tipo subterráneo, las cuales son muy comunes en algunas zonas de Estados Unidos, y pueden provocar contaminación del agua e incendios debido al gas metano (6).

Figura 1.9 Robot *Groundhog* para mapeo de minas (6).

1.2. Minería en el Ecuador

Los proyectos mineros estratégicos en el Ecuador, impulsados durante los últimos años, se ubican en la zona sur del país, en las provincias de Azuay, Morona Santiago y Zamora Chinchipe. Su riqueza radica en los yacimientos de cobre, plata y oro, siendo cuatro de los cinco proyectos aplicados con el método de creación de ductos subterráneos.

Figura 1.10 Proyectos mineros estratégicos del Ecuador.

Fuente: *Espectro minero se abre con Kinross* [en línea]. El Comercio. Redacción Negocios, 8 de diciembre de 2011. Disponible en internet: http://www.elcomercio.com/negocios/Espectro-minero-abre-Kinross_0_604739735.html

El Informe de Gestión del Ministro de Recursos Naturales No Renovables del año 2012 indica que se ha logrado calificar un total de 370 operadores a nivel nacional correspondientes a la pequeña minería. Además, varios proyectos mineros han obtenido la autorización de reinicio de operaciones para exploraciones avanzadas, debido a que han pasado por un proceso de cumplimiento de requisitos establecidos en la Ley de Minería.

Esta ley, además, obliga a la pequeña minería a pagar regalías al Estado del 3%, suponiendo una recaudación de USD 31 millones. La minería a gran escala debe pagar una regalía mínima del 5%, representando solo para el Proyecto Mirador un anticipo de USD 40 millones (7). En el total de los proyectos, se estima una inversión inicial de USD 3600 millones, exportándose el mineral por alrededor de USD 3 700 millones, representado el 14% de las exportaciones anuales y el 5,6% del producto interno bruto PIB[2].

[2] INVEC. *Oportunidades en el sector minero a gran escala del Ecuador* [en línea]. Cuenca, Ecuador: INVEC [7 de marzo de 2013]. Páginas 2, 3, 18. Disponible en web: <http://www.invec.ec/archivos/menu_6/Oportunidades%20en%20el%20Sector%20Minero%20a%20Gran%20Escala%20del%20Ecuador.pdf>

CAPÍTULO 2

DISEÑO MECÁNICO Y ANÁLSIS CINEMÁTICO

2.1. Diseño mecánico

El presente capítulo abarcará los temas de diseño de las partes mecánicas del robot, incluyendo los materiales, formas, disposición y locomoción, todas importantes para permitir al robot movilizarse dentro de un túnel de manera eficiente.

El diseño debe ser realizado de forma que permita la adecuación de sensores de manera sencilla y segura, puesto que estos cumplen una función fundamental para obtener información.

2.1.1. Modelo de locomoción

En base a las descripciones de los modelos de locomoción planteados en el capítulo anterior, se ha decidido por un tipo de locomoción intermedia entre el modelo diferencial y el modelo con orugas. Cuando se tiene un modelo puramente diferencial se puede tener un gran control de los movimientos, puesto que las variables a monitorear son mínimas y estables; sin embargo, su capacidad de sortear terrenos difíciles se encuentra muy limitada. Por otro lado, el modelo con orugas presenta una gran habilidad para terrenos desiguales, dificultándose su control y siendo ineficiente en su consumo energético.

Un acercamiento entre estos dos modelos se puede lograr al considerar la oruga como una serie de ruedas dispuestas una después de otra, lo que permite la distribución del peso del chasis sobre más puntos (Figura 2.1.b). Es así que se aproxima a un modelo intermedio, partiendo del modelo diferencial, con una rueda en el mismo eje a cada lado (Figura 2.1.a), y añadiendo más ruedas que comparten eje a lo largo del chasis. Si se añade un par, se obtiene un modelo diferencial con cuatro ruedas (Figura 1.c), en el cual se logra una mejor estabilidad y se prescinde de las ruedas de apoyo, pero se logra muy poco en cuanto a la distribución de peso, ubicándose los puntos críticos cerca de las esquinas en la localización de las llantas. Si se añade un par más, en el centro (Figura 2.1.d), se obtiene un modelo similar al de un tanque con orugas, con puntos de apoyo en los extremos y en el centro del chasis (8).

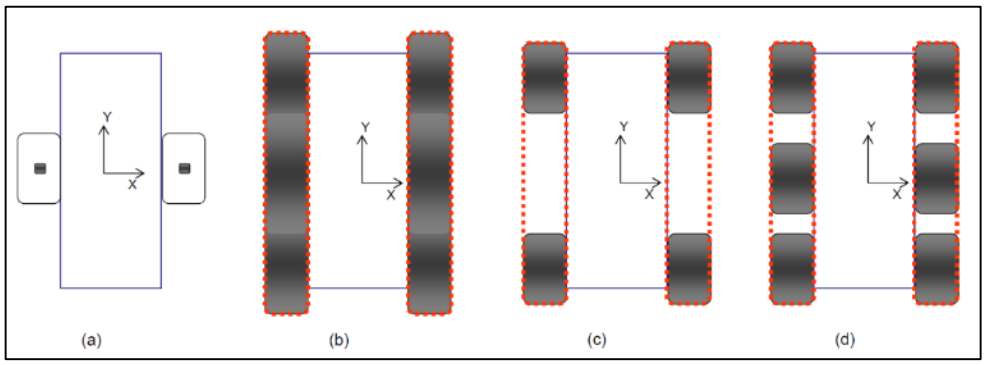


Figura 2.1 Modificación del modelo diferencial (8).

La anterior aproximación, sin embargo, no es completa. La alineación de los ejes y la distancia entre ellos pueden provocar que las llantas no tengan contacto con el suelo, generando inestabilidad y dificultad para el control y medición. Además, es muy probable que, debido a la tendencia de distribución del peso en los extremos delantero y trasero (debido al arranque y frenada), las ruedas del centro tengan poco contacto con el suelo y no apoyen a la tracción y distribución del peso como se preveía en el diseño. Para solucionar este problema estructural, es necesario que las ruedas tengan más flexibilidad para mantenerse la mayor cantidad de tiempo posible sobre el suelo. Esto se logra eliminando el eje central entre las ruedas y colocando en cada una de ellas un sistema de amortiguamiento o suspensión.

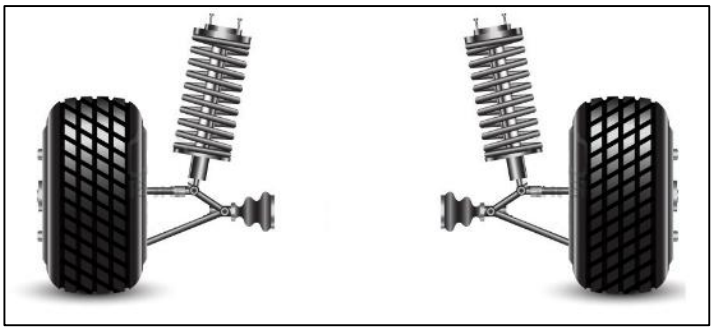


Figura 2.2 Ejemplo de suspensión independiente.

Para la construcción del presente prototipo se ha decidido utilizar una de las plataformas mecánicas comerciales que cumplan con las condiciones de diseño antes mencionadas, y que además posean flexibilidad para la modificación y adición de elementos mecánicos y electrónicos. Se utilizará una escala reducida, puesto que esto disminuirá el costo y permitirá un mejor uso y control de la energía con componentes de menor precio (motores, baterías controladores de potencia). La creación de un modelo propio implica un mayor precio debido al material, maquinado y elección de materiales, los cuales no están fácilmente disponibles en nuestro medio. Sin embargo, se presentarán como conclusiones las recomendaciones para su aplicación al final.

2.1.2. Plataformas de 6 ruedas

Con los antecedentes antes mencionados, se encuentran en el mercado las siguientes plataformas:

2.1.2.1. Aluminum A6WD2 Rover Kit de LynxMotion

Este chasis es de aluminio de 45,72 x 36,19 cm; con llantas de tipo tractor que dan una altura desde el piso de 5 cm. Sin embargo, no posee un sistema de amortiguación, y todos los ejes se encuentran alineados en la misma altura. Su peso es de 2 kg, con capacidad de carga de hasta 2,7 kg. Su cubierta es de un polímero muy resistente y además es un aislante eléctrico. Sin embargo, esta plataforma esta ideada para colocar sobre ella elementos específicos (controladores, baterías, actuadores), por lo que su adaptación a nuevos elementos es complicada.

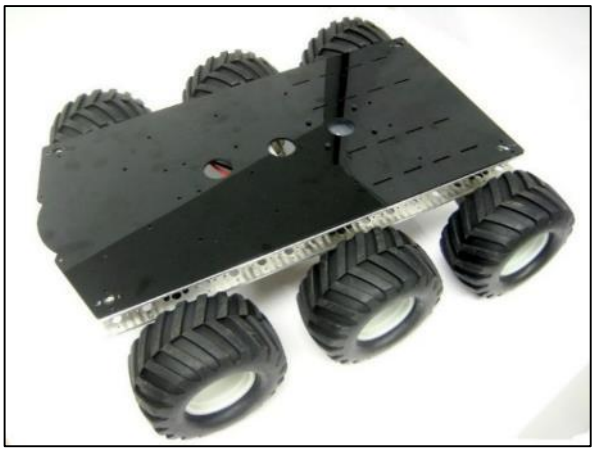


Figura 2.3 Rover A6WD2 de LynxMotion.

Fuente: Lynxmotion. *About the A6WD2 v1 Robot* [en línea]. [20 noviembre de 2013]. Disponible en web: <http://www.lynxmotion.com/c-163-a6wd2-no-electronics.aspx.>

2.1.2.2. 6WD All Terrain Robot Platform de SuperDroid

Este chasis es mucho más robusto y de grandes dimensiones: 52,2 x 48,57 cm. Sus llantas permiten una distancia desde el piso de 5,4 cm y son neumáticas, lo que mejora su adaptabilidad al terreno, sin embargo, no reemplazan a una suspensión (no incluida). Tiene un peso 15 kg, con capacidad de carga de hasta 25 kg. Está hecho de aluminio y los motores requieren de baterías de 24 voltios. Esto aumenta las características de los dispositivos de control de motores, los que tienen que soportar mayor voltaje y corriente, incurriendo en mayor costo.

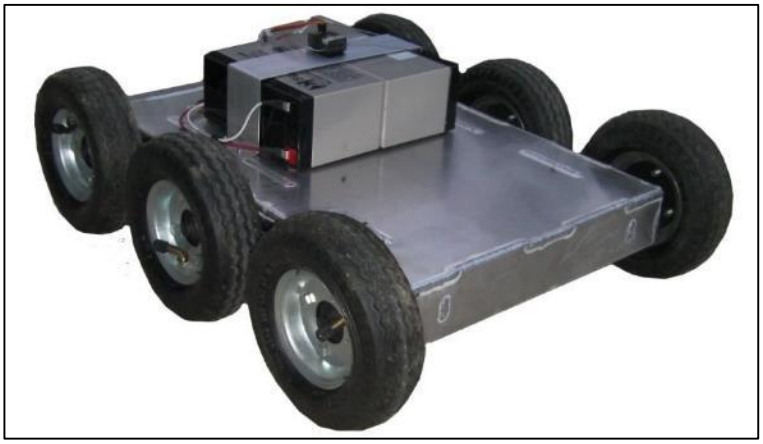


Figura 2.4 Plataforma 6WD de SuperDroid.

Fuente: SuperDroid Robots. *6WD All Terrain Robot Platform with 42mm motors* [en línea]. [21 noviembre de 2013]. Disponible en web: <http://www.superdroidrobots.com/shop/item.aspx/6wd-all-terrain-robot-platform-with-42mm-motors/1476/>.

2.1.2.3. Odyssey 6 Wheel Drive de Orión Robotics

Esta plataforma robótica posee amortiguación independiente en cada llanta basada en un pequeño muelle. Sus dimensiones, peso y motores no se encuentran especificadas, y su plataforma principal posee agujeros para la colocación de elementos específicos (brazo robótico).

Figura 2.5 Robot Odyssey 6 Wheel Drive de Orión Robotics.
Fuente: Orion Robotics. *Odyssey 6 Wheel Drive Robot* [en línea]. [21 noviembre de 2013]. Disponible en web: <http://www.orionrobotics.com/Odyssey-6-Wheel-Drive-Robot_p_250.html>.

2.1.2.4. DAGU Wild Thumper 6WD Robot Chassis Set de Dagu

Plataforma robótica con ruedas de goma de 12 cm de diámetro. Su estructura está hecha de aluminio anodizado de 2 mm de grosor, en el que se ha realizado una matriz de agujeros estándar de 4 mm. Cada rueda posee un sistema de amortiguación compartido con su rueda contraria, basado en un resorte que permite el movimiento de las ruedas sobre los obstáculos. La distancia desde el piso es de 4,6 cm, con distancias entre los ejes de 15 cm. Sus dimensiones totales son de 29,7 x 42,8 cm. Se especifica que el robot puede incluir motores de 160 rpm a 6 VDC con engranajes reductores de metal con relaciones de 34:1, 75:1, e incluso mayores relaciones (99:1 hasta 499:1), las cuales no están recomendadas por el fabricante, debido que la propia fuerza de los engranajes es suficiente para destruir el eje del motor. Su peso es de 2,7 kg, y tiene una capacidad de carga de hasta 5 kg.

Figura 2.6 Chasis DAGU Wild Thumper 6WD.
Fuente: Pololu. *Dagu Wild Thumper 6WD All-Terrain Chassis, Black, 75:1* [en línea]. [21 noviembre de 2013]. Disponible en web: <http://www.pololu.com/product/1563>.

Debido a la facilidad de adaptación de elementos electrónicos, sensores y baterías, al cumplimiento con las características principales, y a su información detallada en la web, además de la oferta de repuestos de sus llantas y la variedad de motores, se ha elegido la plataforma DAGU Wild Thumper 6WD con motores de relación 75:1, similar a la que se muestra en la Figura 2.6. El robot se puede desplazar a una velocidad máxima de 3 km/h.

2.1.3. Ensamblaje del chasis

El chasis robótico Wild Thumper 6WD se obtiene en piezas, tal como se muestra en la Figura 2.7. Los motores con sus respectivas cajas reductoras se encuentran atornillados al chasis mediante los contenedores de la amortiguación (Figura 2.8). Cada motor posee un par de cables soldados, los cuales convergen en un conector de nylon incluido en el centro de la estructura principal. El proceso de ensamblaje requiere la colocación de las ruedas, las cuales poseen un tornillo para asegurarse al eje de la caja reductora de los motores, la colocación de la tapa protectora superior y el ajuste de cualquier pieza en donde sea necesario.

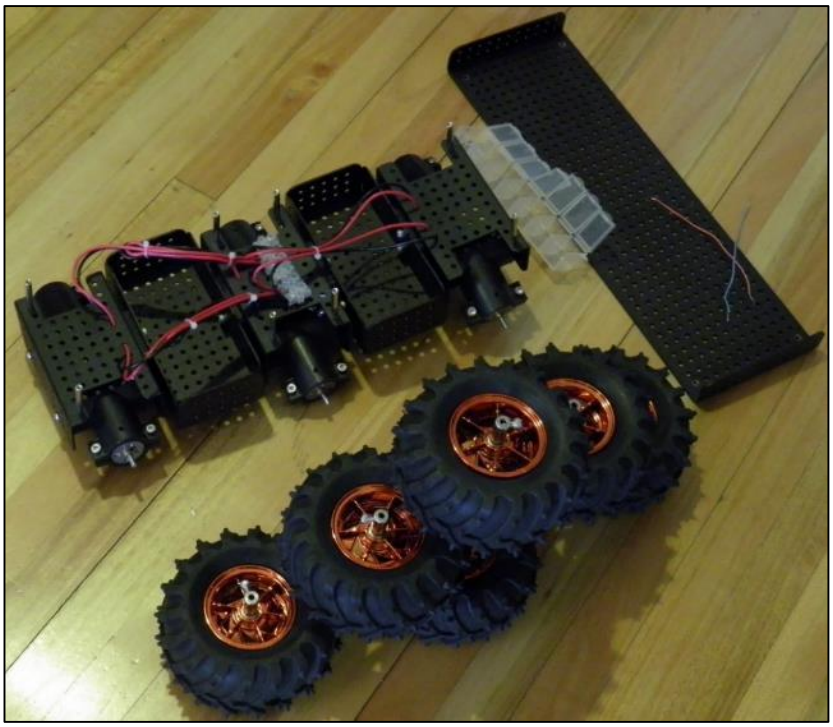


Figura 2.7 Piezas incluidas en el kit Wild Thumper 6WD.

Es importante notar que los contenedores no permiten la colocación de un sistema de *encoders* o codificadores rotativos que permitirían la medición de la velocidad de las ruedas, y por lo tanto, del espacio recorrido por el robot. Sin embargo, este sistema no es indicado en robots para terrenos irregulares, puesto que las ruedas no siempre giran en conjunto y se producen muchos deslizamientos, generando falsas mediciones y errores en el posicionamiento del robot. En el anexo 1 se muestra el manual de ensamblaje y dimensiones del chasis.

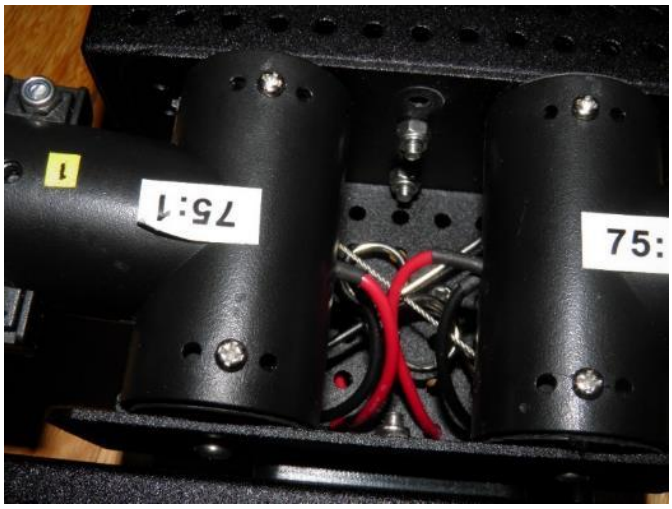


Figura 2.8 Contenedores de los motores y sistema de amortiguación.

2.2. Análisis cinemático

El análisis cinemático de un sistema mecánico es básico para entender su comportamiento. En el caso de los robots móviles, esta tarea es fundamental para el diseño del hardware y software para su control. Para ello se definen las ecuaciones cinemáticas y de estado del robot en todo momento, basadas en las variables de control que se pueden modificar. Aunque las variables dinámicas, como el peso del robot, la fuerza de los motores, los coeficientes de rozamiento, etc., jueguen un papel importante en cuanto a las limitaciones en el movimiento del robot, estas pueden considerarse aproximadamente constantes en un análisis previo. En la cinemática se analiza la velocidad, movimientos y posiciones sin tomar en cuenta las fuerzas que lo producen.

2.2.1. Modelo cinemático simplificado

Con el objetivo de simplificar el modelo y facilitar su posterior aplicación en software se tomaran en cuenta varias condiciones:

- Se considera al robot como un modelo diferencial, es decir las tres ruedas de cada lado se consideran una, y se les aplica la misma variable de control, por lo tanto poseen la misma velocidad.

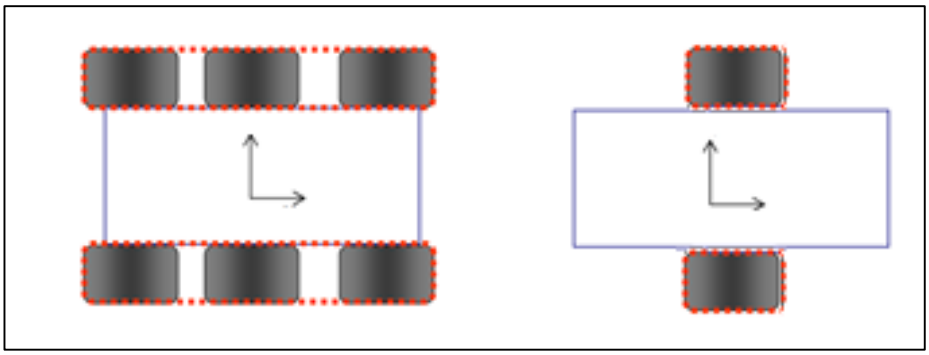


Figura 2.9 Simplificación del modelo de 6 ruedas a un modelo diferencial (dos ruedas).

- Las ruedas no presentan deslizamiento lateral, es decir, solo se trasladan en dirección perpendicular a su eje.

- El punto P (x, y) es la posición actual del centro del robot en el marco general.

- Cada eje de cada conjunto de ruedas se encuentra a una distancia $0{,}5l$ del punto P.

- En el marco de referencia propio del robot (X_R, Y_R), el movimiento hacia adelante siempre coincide con el eje X_R.

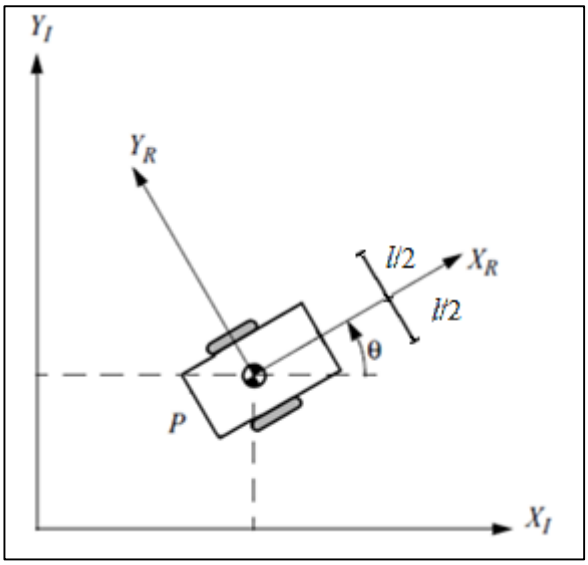


Figura 2.10 Robot diferencial dentro de los marcos de referencia propio y general. (1)

- La velocidad de giro de cada rueda se expresan mediante las variables φ_R y φ_L, por lo tanto la velocidad lineal o desplazamiento que genera cada rueda está dado por $v_R = \varphi_R \cdot c$, $v_L = \varphi_L \cdot c$, donde c es el radio de la rueda.

- El ángulo θ representa el desplazamiento angular del marco de referencia propio del robot con respecto al marco general (X_1, Y_1).

2.2.2. Ecuaciones cinemáticas

En primer lugar, se definen las variables que se deben conocer en todo momento para definir con certeza la ubicación del robot. A esto se le denomina estado, y está formado por el vector que contiene las coordenadas x e y del punto P con respecto al marco general, y la orientación θ del robot $E = [x \;\; y \;\; \theta]^T$.

De la misma manera, se puede definir un vector de velocidades $\dot{E} = [\dot{x} \;\; \dot{y} \;\; \dot{\theta}]^T$. Si tomamos en cuenta el vector de velocidades con referencia al marco propio del robot, se puede simplificar la ecuación a $\dot{E}_R = [\dot{x} \; 0 \; \dot{\theta}]^T$, puesto que, como se había impuesto en las condiciones del modelo, el movimiento hacia adelante se encuentra en dirección X_R, y no existe ningún deslizamiento, por lo tanto $\dot{y}_R = 0$.

2.2.2.1. Determinación de la velocidad

Para referenciar el vector de velocidades $\dot{E}_R$ dentro del marco de referencia general se necesita un elemento que relacione los dos marcos de referencia. Este elemento es una matriz de rotación $R_R^1(\theta)$. El subíndice indica el marco actual en el que se mide la velocidad ($\dot{E}_R$), y el superíndice indica con qué marco de referencia está relacionado. Esta matriz está en función del ángulo θ que existe entre los marcos de referencia (X_R, Y_R) y (X_1, Y_1).

$$\dot{E}_1 = R_R^1(\theta) * \dot{E}_R$$

$$\dot{E}_1 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_R \\ 0 \\ \dot{\theta} \end{bmatrix}$$

A continuación, se debe expresar el vector de velocidades en función de las variables que se pueden controlar de manera directa, en este caso, las velocidades angulares de las ruedas. Para el análisis dentro del marco propio del robot (X_R, Y_R) existen dos casos:

- La velocidad lineal (en dirección a X_R) es la suma de los aportes en desplazamiento que el conjunto de ruedas de cada lado genera. Debido a que las ruedas se encuentran a la misma distancia del punto P, cada una aporta con la mitad del desplazamiento:

$$V = \dot{x}_R = 0{,}5\ (v_R + v_L)$$

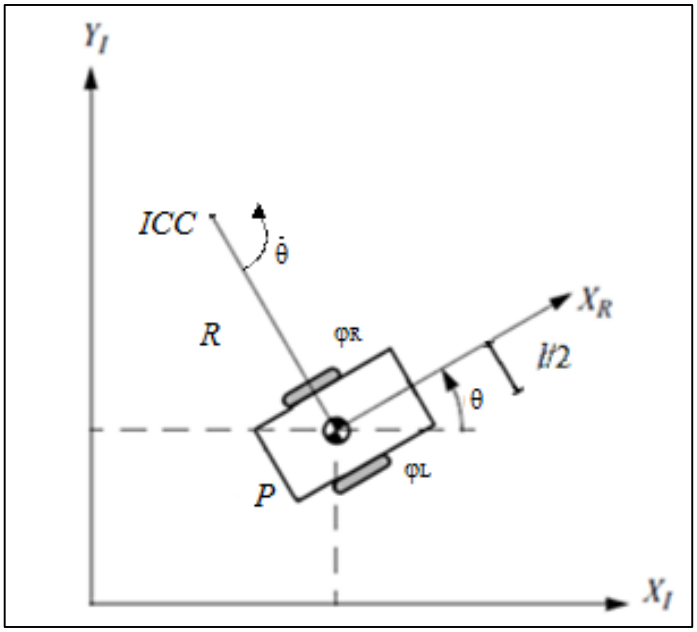


Figura 2.11 Velocidad angular del robot (1).

- La velocidad angular es la suma de los aportes que el conjunto de ruedas de cada lado genera para girar el marco de referencia propio con respecto al general. Para ello, el conjunto de ruedas de cada lado describen arcos concéntricos con distintos radios. El punto central común se denomina centro instantáneo de curvatura (*Instantaneous Center of Curvature*, ICC), y se encuentra a una distancia R de P.

Como se aprecia en la Figura 2.11, la velocidad angular $\dot{\theta}$ es común para todo el robot, por lo tanto también lo es para el conjunto de ruedas de cada lado. Es así que se pueden escribir las velocidades lineales de la siguiente manera:

$$\dot{\theta}\ (R + 0{,}5l) = v_R \quad ; \quad \dot{\theta}\ (R - 0{,}5l) = v_L$$

Y resolviendo para R y para $\dot{\theta}$:

$$R = \frac{l}{2} \cdot \frac{v_L + v_R}{v_R - v_L} \quad ; \quad \dot{\theta} = \frac{v_R - v_L}{l}$$

Reemplazando los datos obtenidos en la ecuación de velocidades con referencia al marco general, se obtiene el modelo de velocidades del robot:

$$\dot{E}_1 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{(v_R + v_L)}{2} \\ 0 \\ \frac{v_R - v_L}{l} \end{bmatrix}$$

2.2.2.2. Determinación de la Posición

La posición del robot se puede determinar en un tiempo cualquiera $t_0 + dt$ conociendo el estado inicial del robot con respecto al marco general E_0, el vector de velocidades con respecto al marco general, y el tiempo transcurrido. La ecuación general se puede expresar como:

$$E_1(t + dt) = R_R^1(\theta)\, \dot{E}_1 \cdot dt + E_0$$

$$\begin{bmatrix} x_{t_0+dt} \\ y_{t_0+dt} \\ \theta_{t_0+dt} \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & 0 \\ \sin\theta_0 & \cos\theta_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{(v_R + v_L)}{2} \\ 0 \\ \frac{v_R - v_L}{l} \end{bmatrix} \cdot dt + \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix}$$

La ecuación matricial anterior define la posición del robot para cualquier tiempo, siendo conocidos los estados iniciales y las velocidades de los motores. Sin embargo, para determinar la posición de manera precisa es necesario calcular el estado en intervalos de tiempo cortos para evitar errores. Esto implica el manejo de gran cantidad de información en poco tiempo, y creando la necesidad de sistema de procesamiento de importantes prestaciones. A pesar de conseguir un sistema de procesamiento con las características necesarias, seguirían existiendo errores debido a la resolución de los sensores, la descarga de baterías, demora en las comunicaciones, etc.

Por lo tanto, es necesario simplificar el sistema o disminuir la cantidad de información a manejar. Para ello, se limita la velocidad de los motores a dos casos:

1. Si $v_R = v_L$, entonces el movimiento será en línea recta, $\dot{\theta} = 0$.

$$\begin{bmatrix} x_{t_0+dt} \\ y_{t_0+dt} \\ \theta_{t_0+dt} \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & 0 \\ \sin\theta_0 & \cos\theta_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_L \\ 0 \\ 0 \end{bmatrix} \cdot dt + \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix}$$

$$\begin{bmatrix} x_{t_0+dt} \\ y_{t_0+dt} \\ \theta_{t_0+dt} \end{bmatrix} = \begin{bmatrix} x_0 + v_L \cos(\theta_0)\, dt \\ y_0 + v_L \sin(\theta_0)\, dt \\ \theta_0 \end{bmatrix}$$

2. Si $v_R = -v_L$, entonces R = 0, y el ICC se encuentra en el punto P. Además, se cumple que $V = \dot{x}_R = 0$.

$$\begin{bmatrix} x_{t_0+dt} \\ y_{t_0+dt} \\ \theta_{t_0+dt} \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & 0 \\ \sin\theta_0 & \cos\theta_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \frac{2v_L dt}{l} \end{bmatrix} \cdot dt + \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix}$$

$$\begin{bmatrix} x_{t_0+dt} \\ y_{t_0+dt} \\ \theta_{t_0+dt} \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 + \frac{2v_L dt}{l} \end{bmatrix}$$

Estas dos condiciones se cumplirán en el control del robot, sea en su forma autónoma o a control remoto. La programación del equipo remoto y de las placas controladoras tomará en cuenta estas condiciones de diseño.

2.2.3. Modelo dinámico

La aplicación de un modelo dinámico a robots móviles implica gran complejidad debido a las incógnitas en cuanto a la medición de las fuerzas que interactúan en el movimiento del robot. No existen ecuaciones que describan fielmente la interacción entre las llantas y el terreno, la influencia de la suspensión y la fuerza de los motores. Sin embargo, existen aproximaciones prácticas que son útiles en la creación de un modelo. Este modelo, en conjunto con el análisis cinemático, provee de grandes herramientas para poder aplicar sistemas de control y conocer las capacidades y limitaciones del robot (9).

El análisis se basa en la determinación de las fuerzas que influyen sobre el robot durante el movimiento. La fuerza de acción principal es la tracción generada por las ruedas a través de los motores, y la principal fuerza de reacción la genera el rozamiento de las ruedas sobre el suelo, influenciado por el peso del robot.

Las fuerzas y momentos son presentados en la Figura 2.12, donde F_{xi} representa la fuerza de tracción generada por los motores y R_{xi} representa la fuerza resistiva en cada llanta (cada fuerza está referida al lado izquierdo o derecho). Las llantas de la izquierda son activadas con la misma señal, al igual que las llantas del lado izquierdo, por lo que $F_{dx1}=F_{dx2}=F_{dx3}$ y $F_{ix1}=F_{ix2}=F_{ix3}$. Las fuerzas laterales resistivas R_{yi} son consecuencia del deslizamiento lateral, el cual es mínimo cuando el robot se mueve hacia adelante o atrás; sin embargo, cuando el robot realiza giros, esta fuerza es considerable. Estas fuerzas aportan al momento de inercia total M, ya sea a favor o en contra (M_r). (10)

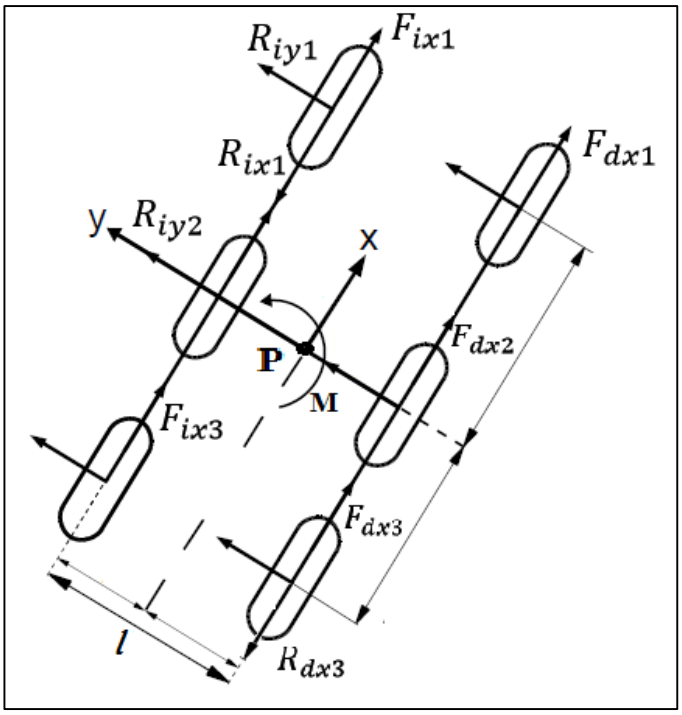


Figura 2.12 Diagrama de fuerzas para modelo dinámico.

Con este análisis se construyen las ecuaciones dinámicas para el momento de inercia I, y para la aceleración en las direcciones X e Y:

$$m \cdot a_x = 3F_{dx} + 3F_{ix} - R_x$$

$$m \cdot a_y = -R_y$$

$$M = I \cdot \ddot{\theta} = 0.5 \cdot l \cdot (3F_{dx} - 3F_{ix}) - M_r$$

Donde *m* es la masa del robot e *I* es el momento de inercia alrededor del punto P considerado el centro de masa. R_x y R_y representan el conjunto de fuerzas de reacción contrarias al movimiento.

Las fuerzas R_y y R_x pueden determinarse mediante pruebas de campo, ya que dependen del coeficiente de fricción entre el material de las llantas y el suelo. El momento resistivo M_r depende de las fuerzas R_y, R_x, de los coeficientes de rozamiento y del radio de curvatura descrito por el robot.

CAPÍTULO 3

DISEÑO ELECTRÓNICO

El diseño electrónico es una de las partes principales del sistema. Esto incluye una fuente de alimentación con gran autonomía y potencia, chips controladores programables para manejo de motores, adquisición de datos de los sensores y circuitos de comunicación interna y externa con el equipo remoto.

3.1. Arduino

Arduino inició en 2005 como un proyecto para los estudiantes del Interaction Design Institute de Ivrea en Italia (IIDI). El objetivo era crear una plataforma para programación de microcontroladores que sea mucho más barata, potente y fácil de programar que las que se encontraban en el mercado. Además, era necesario que el dispositivo sea *plug-and-play* y multiplataforma, por lo que su comunicación con el computador debía ser USB para evitar la necesidad de módulos adicionales (como grabadores y circuitos de comunicación con el computador) y poder ser usada en PC y MAC (11).

El proyecto Arduino es de hardware libre, bajo licencia Creative Commons[3], y el software posee una licencia libre GNU General Public License[4]. El producto final es una plataforma de hardware basado en microcontroladores Atmel incluidos en una placa de circuito impreso con entradas y salidas, la cual se puede programar mediante USB en un entorno de desarrollo basado en software Wiring y Processing, el cual se explica en los siguientes capítulos.

[3] Creative Commons. *About* [en línea]. [12 diciembre de 2013]. Disponible en web: <http://creativecommons.org/about/>.
[4] GNU. *GNU General Public License* [en línea]. [11 diciembre de 2013]. Disponible en web: <http://www.gnu.org/copyleft/gpl.html>.

Figura 3.1 Logo de Arduino.
Fuente: Wikimedia Commons. *File:Arduino Logo.svg* [en línea]. [11 diciembre de 2013]. Disponible en web: <http://commons.wikimedia.org/wiki/File:Arduino_Logo.svqg>.

Estas características han hecho de Arduino la plataforma electrónica de desarrollo más usada por aficionados, como artistas y diseñadores, por la facilidad de programación y creación de proyectos. Sin embargo, el uso por parte de profesionales y estudiantes de ingeniería no se queda atrás, puesto que sus facilidades de programación permiten el emprendimiento de proyectos complejos con mayor facilidad y en menor tiempo. Además, al ser libre, gran parte de la evolución, mantenimiento, corrección y creación de accesorios es realizada por las comunidades en internet, en especial a través de su foro (http://forum.arduino.cc/), wiki (http://playground.arduino.cc/) y comunidad Google +, las cuales aportan en todos los niveles y mejoran la calidad de los productos finales.

Su desarrollo ha sido tan exitoso que existen proyectos "clones" de Arduino, creados para mercados más específicos (placa Netduino[5] programable con .NET), a menor precio (placas de la empresa SainSmart[6]), basados en otra marca de microcontroladores (chipKIT, hechos con microcontroladores Microchip PIC), creados para una aplicación específica (FlyDuino[7] creado para robots voladores), y muchísimos más. Además, existe una variedad de placas oficiales Arduino diseñadas para distintas necesidades y aplicaciones.

[5] NetDuino. *Netduino* [en línea]. [11 diciembre de 2013]. Disponible en web: < http://netduino.com/>.
[6] Sainsmart. *Sainsmart* [en línea]. [11 diciembre de 2013]. Disponible en web: <http://www.sainsmart.com>.
[7] Flyduino. *Flyduino* [en línea]. [11 diciembre de 2013]. Disponible en web: < http://flyduino.net/>.

3.2. Hardware Arduino

La placa Arduino consiste en un microcontrolador principal Atmel AVR de 8 bits con elementos complementarios para facilitar su programación y uso. El microcontrolador principal se encuentra pregrabado con un programa conocido como *bootloader* o gestor de arranque, el cual permite que el microcontrolador sea reconocido como un dispositivo Arduino. Para su programación y comunicación con el computador, las placas de Arduino, en general, poseen un pequeño microcontrolador Atmel programado como un convertidor de USB a Serial, por medio del cual se puede comunicar el computador con el microcontrolador principal para grabar información en el mismo o realizar pruebas de comunicación. En otras placas, el microcontrolador principal realiza también la comunicación USB (12).

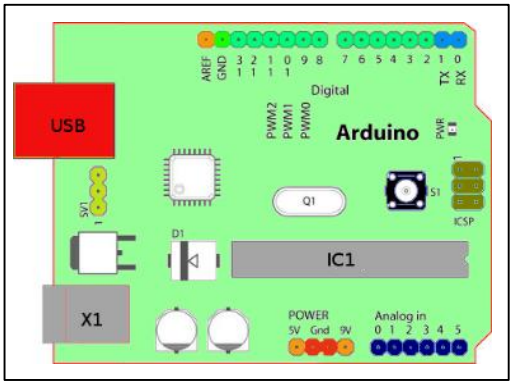


Figura 3.2 Esquema general de una placa Arduino.
Fuente: Ikkaro. *Que es Arduino* [en línea]. [12 diciembre de 2013]. Disponible en web: <http://www.ikkaro.com/definicion-arduino/>.

La mayoría de placas poseen un conector genérico para alimentarse mediante una fuente exterior con un regulador lineal, o simplemente obtienen la energía del puerto USB. Su funcionamiento es de 5 VDC, y también poseen compatibilidad con 3,3 VDC. Las entradas y salidas de la placa están disponibles a través de conectores hembra tipo peineta, los cuales están a distancias normalizadas en todos los tipos de placas, lo que permite la conexión de cualquier tipo de sensor o actuador, principalmente placas de expansión con diversas aplicaciones conocidas como *shields*. También es común que las placas posean un botón de reinicio o RESET, e indicadores LED que permiten hacer pruebas iniciales o muestran el estado de las comunicaciones.

3.2.1. Placas Arduino Oficiales

Existe una gran cantidad de placas oficiales de Arduino, las cuales han sido creadas para aplicaciones y necesidades específicas. A continuación, se muestra la lista de placas oficiales actualizada.

- Arduino Uno, Arduino Mega 2560, Arduino Due, Arduino Micro, Arduino Mini, Arduino Nano, Arduino Pro Mini, Arduino Pro, Arduino Leonardo son placas genéricas de entrada-salida con diferentes formas y tamaños.

- Arduino Yún y Arduino Ethernet poseen elementos de control adicionales para manejo de redes mediante cable Ethernet.

- Arduino Robot posee sensores, pantalla LCD y ruedas para prácticas básicas de robótica.

- Arduino Esplora tiene forma de control de juegos, con botones y joystick.

- Arduino Mega ADK permite comunicación con dispositivos Android.

- Arduino Tre (en desarrollo) es una plataforma que incluye un sistema Linux embebido.

A continuación, se describirán las placas Arduino Uno y Arduino Mega 2560, las cuales serán utilizadas dentro del prototipo.

3.2.1.1. Arduino Uno

La placa más popular es el Arduino UNO, debido a su tamaño reducido y bajo precio que actualmente se encuentra en su versión R3. Está basado en el microcontrolador de 8 bits de Atmel ATmega328[8], con 14 pines de entrada-salida y 6 entradas analógicas disponibles, el cual funciona a una velocidad de 16 MHz con un cristal externo. También, posee un microcontrolador ATmega16U2, el cual está programado como un convertidor de USB a Serial. Posee una memoria de 32 kB (kilobytes), de los cuales 500 bytes están usados por el gestor de arranque. Entre sus características principales, se encuentran:

- Un sistema de comunicación Serial nativo USART (*Universal Asynchronous Receiver-Transmitter*), pines 0 y 1.
- Un sistema de comunicación SPI (*Serial Peripheral Interface*), en los pines 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).
- Un sistema de comunicación I2C (*Inter-Integrated Circuit*), pines A4 o SDA y A5 o SCL.
- Dos interrupciones externas en los pines 2 y 3.
- 6 canales de PWM (Modulación de Ancho de Pulso), en los pines 3, 5, 6, 9, 10 y 11.

Figura 3.3 Placa Arduino UNO R2.

Fuente: Arduino. *Arduino Uno* [en línea]. [21 diciembre de 2013]. Disponible en web <http://arduino.cc/en/uploads/Main/ArduinoUno_r2_front.jpg>.

[8] ATMEL. *8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash, ATmega48PA ATmega88PA ATmega168PA ATmega328P.* Rev. 8161D–AVR–10/09 [en línea]. 2009. [20 diciembre de 2013]. Disponible en web: <http://www.atmel.com/Images/doc8161.pdf>

3.2.1.2. Arduino Mega 2560

La placa Arduino Mega 2560 está basada en el microcontrolador de 8 bits ATmega2560[9], el cual posee 54 pines de entrada-salida y 16 entradas analógicas. Su velocidad es de 16 MHz, y su memoria es de 256 kB, de los cuales 8 están usados por el gestor de arranque. Igual que el Arduino Uno, esta placa posee un microcontrolador convertidor de USB a Serial para su comunicación con el computador. A continuación, se enumeran sus principales características:

- 4 sistemas de comunicación Serial nativo (USART):
  - Serial: 0 (RX), 1 (TX).
  - Serial 1: 19 (RX), 18 (TX).
  - Serial 2: 17 (RX), 16 (TX).
  - Serial 3: 15 (RX), 14 (TX).
- Un sistema de comunicación SPI, en los pines 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).
- Un sistema de comunicación I2C en los pines 20 (SDA) y 21 (SCL).
- 6 interrupciones externas, en los pines 2, 3, 18, 19, 20 y 21.
- 15 canales de PWM, desde el pin 2 al 13, y del 44 al 46.

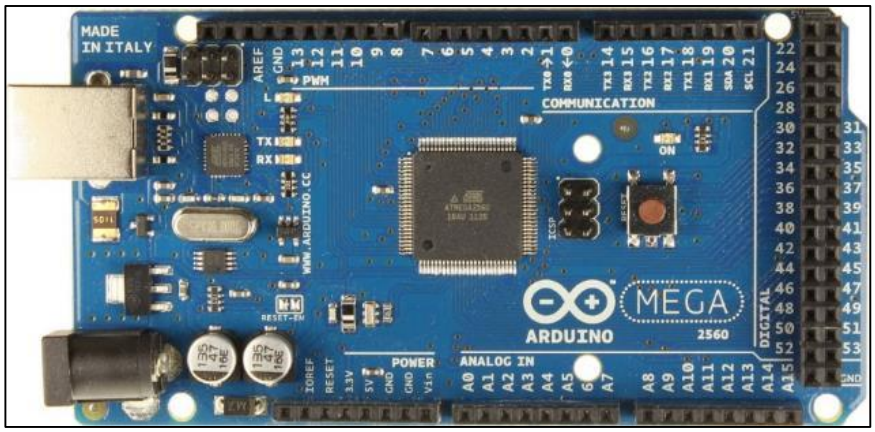


Figura 3.4 Placa Arduino Mega 2560

Fuente: Arduino. *Arduino Mega 2560* [en línea]. [20 diciembre de 2013]. Disponible en web <http://arduino.cc/en/uploads/Main/ArduinoMega2560_R3_Front.jpg>.

[9] ATMEL. *8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash, ATmega640/V ATmega1280/V ATmega1281/V ATmega2560/V ATmega2561/V*. 2549P–AVR–10/2012 [en línea]. 2012. [20 diciembre de 2013]. Disponible en web: <http://www.atmel.com/Images/doc2549.pdf>

A pesar de que la placa Arduino Mega 2560 es más grande y posee más entradas-salidas que el Arduino UNO, es importante notar que la distancia entre los conectores tipo peineta que se encuentran paralelos en los extremos de la placa es la misma para ambos.

3.2.2. Plataforma chipKIT compatible con Arduino

La placa de desarrollo chipKIT Uno32 compatible con Arduino está basada en microcontroladores Microchip PIC32. Tiene la misma forma y tamaño que un Arduino Uno y es compatible con los *shields* creados para el mismo. Permite una programación USB a través de un chip Serial a USB[10].

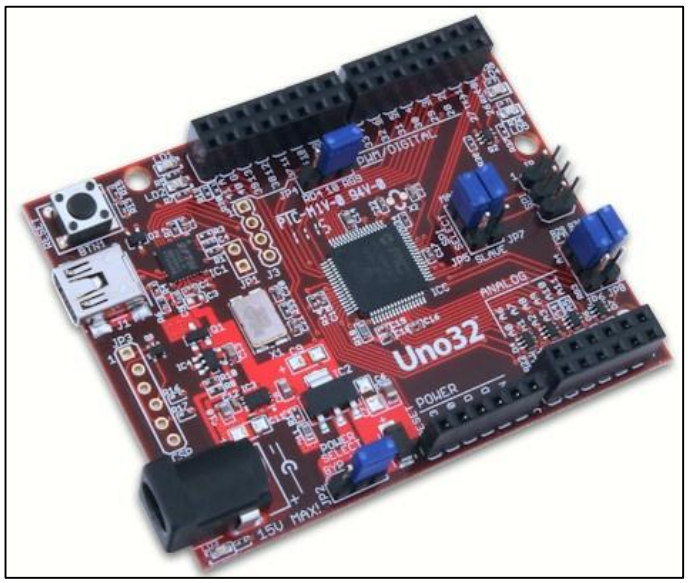


Figura 3.5 Placa chipKIT Uno32 compatible con Arduino.
Fuente: Digilent. *ChipKIT Uno32* [en línea]. [22 diciembre de 2013]. Disponible en web: <http://www.digilentinc.com/Data/Products/CHIPKIT-UNO32/chipKIT-Uno32-obl-500.jpg>.

Su ventaja se encuentra en el uso de un microcontrolador PIC32MX320F128 con arquitectura de 32 bits, una velocidad de procesamiento de 80 MHz, 128 kB de memoria, y en la existencia de 42 pines de entrada-salida. A pesar de que su operación es a 3,3 VDC, es compatible con cualquier placa de expansión a 3,3 VDC o 5 VDC. A continuación, se enumeran las características principales de esta placa:

[10] Digilent. *ChipKIT™ Uno32™ Board Reference Manual.* [en línea]. [21 diciembre 2013]. Disponible en web: <http://www.digilentinc.com/Data/Products/CHIPKIT-UNO32/chipKIT-Uno32-RevC_rm.pdf>.

- 2 sistemas de comunicación Serial nativo (USART):
    - Serial: 0 (RX), 1 (TX).
    - Serial 1: 39 (RX), 40 (TX).
- 2 sistemas de comunicación SPI, el primero en los pines 12 (MISO), 11 (MOSI), 13 (SCK), 10 (SS) y el segundo en los pines 0 (MISO), 1 (MOSI), 38 (SCK), 39 (SS).
- 2 sistemas de comunicación I2C en los pines A4 (SDA1) y A5 (SCL1), y en 39 (SDA2) y 14 (SCL2).
- 5 interrupciones externas, en los pines 2, 7, 8, 35 y 38.
- 5 canales de PWM, en los pines 3, 5, 6, 9 y 10.

Esta placa supera a Arduino en cuanto a velocidad, número de pines disponibles de entrada-salida, periféricos de comunicación y memoria de programación disponible. Sin embargo, su compatibilidad con código de Arduino no es completa, ya que su código es traducido para acomodarse al sistema PIC de Microchip.

3.2.3. Shields Arduino

Las placas Arduino oficiales, así como sus similares o clones permiten el uso de *shields*, que son placas de circuitos impresos para expandir las funcionalidades de la placa base, y que generalmente se colocan sobre los conectores de la misma. Entre las funciones que se pueden añadir se encuentran controladores de motores, pantallas, antenas de todo tipo (GPS, Wireless, ZigBee, radiofrecuencia), sensores, luces, etc.

Existen varios *shields* oficiales de Arduino, sin embargo, la mayor cantidad y variedad de estos dispositivos han sido diseñadas por terceros con aplicaciones específicas, muchos de los cuales se encuentran en el mercado. Para propósitos de este proyecto se utilizarán *shields* comerciales y otros realizados específicamente para ajustarse a las necesidades del prototipo. Muchos de ellos incluso permiten la conexión de otros *shields* encima de los mismos (Figura 3.6), permitiendo ampliar sus funcionalidades aún más.

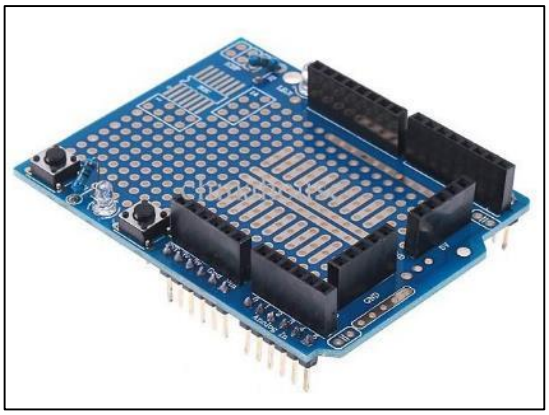


Figura 3.6 *Shield* Genérico para Arduino.
Fuente: Arduino. *Arduino Proto Shield* [en línea]. [21 diciembre 2013]. Disponible en web: <http://arduino.cc/en/Main/ArduinoProtoShield>.

3.2.3.1. Ethernet Shield

Es una placa de expansión que permite la conexión a redes de tipo Ethernet 10/100 mediante un cable RJ-45. Está basado en el controlador Wiznet W5100, el cual provee un acceso a redes IP a través de TCP (*Transmission Control Protocol*) o UDP (*User Datagram Protocol*), soportando hasta cuatro conexiones simultáneas. Este chip se encuentra en una placa de circuito impreso que se coloca sobre cualquier placa Arduino o compatible, permitiendo la colocación de otro *shield* de expansión sobre el mismo.

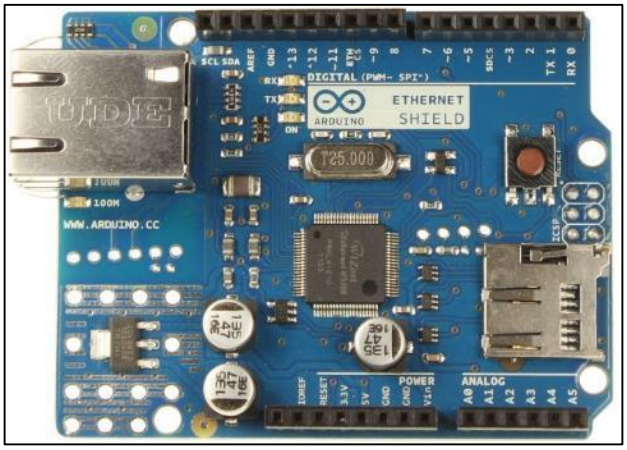


Figura 3.7 Arduino Ethernet *Shield*.
Fuente: Arduino. *Arduino Proto Shield* [en línea]. [22 diciembre de 2013]. Disponible en web: <http://arduino.cc/en/uploads/Main/ArduinoEthernetShield_R3_Front.jpg>.

El chip Wiznet W5100[11] permite el control, configuración, envío y recepción de datos a través de la interfaz SPI de la placa principal. Con el objetivo de aprovechar esta interfaz (una por cada Arduino) y dar mayor funcionalidad a la placa de expansión, en ella se encuentra un conector para tarjetas de memoria de tipo micro-SD, el cual comparte el bus de comunicaciones con el controlador Wiznet. La placa posee, además, LEDs que indican el estado del dispositivo y las comunicaciones en proceso, y un botón de RESET que reinicia tanto el microcontrolador principal como el controlador Wiznet en el *shield.*

Con esta placa de expansión, y mediante una conexión inalámbrica hacia un computador, se realizará el envío de datos de control y de información del medio en ambas direcciones. En el anexo 2 se muestra el esquema del circuito de esta placa de expansión.

3.2.3.2. Shield de Motores

Los motores incluidos en el chasis del Wild Thumper 6WD son de corriente continua, con imanes extra fuertes para poder mover con facilidad la caja de reducción de 75:1. Su rango de voltaje está entre los 6 y 7,2 VDC, generando una velocidad de 160 revoluciones por minuto (rpm) y un consumo mínimo de corriente de 450 mA, generando un par motor de 11 kg-cm.

Figura 3.8 Motor DC con caja reductora 75:1.
Fuente: Pololu Robotics & Electronics. *75:1 Metal Gearmotor 25Dx54L mm HP* [en línea]. [22 de diciembre de 2013]. Disponible en web: <http://www.pololu.com/picture/view/0J2644>.

[11] WIZnet. *W5100 Datasheet Version 1.2.2.* [en línea]. [22 de diciembre de 2013]. Disponible en web: <http://www.wiznet.co.kr/UpLoad_Files/ReferenceFiles/W5100_Datasheet_v1.2.2.pdf>.

Debido a la necesidad de manejar 6 motores, 3 de cada lado, con una corriente máxima de 6,6 A DC cada uno (en promedio 2 A cada uno), se requieren controladores de potencia con características suficientes para manejar con holgura la cantidad de 19,8 amperios por lado, con un total de 39,6 A a 7.2 V en el peor caso. Además, se requiere que el controlador sea capaz de regular el voltaje que se le entrega a los motores, con el objetivo de controlar su velocidad.

El circuito controlador genérico para este propósito es un puente H o puente de transistores (Figura 3.9), que puede ser realizado con elementos individuales conectados entre sí. En este circuito, los transistores actúan como interruptores, pudiendo conectar al motor a la fuente de energía con polaridad directa (T1 y T4 encendidos) e inversa (T2 y T3), permitiendo mover el motor en ambos sentidos. Cuando todos los transistores (T1-T4) se encuentran apagados, el motor estará libre bajo su propia inercia, deteniéndose eventualmente debido a la fricción.

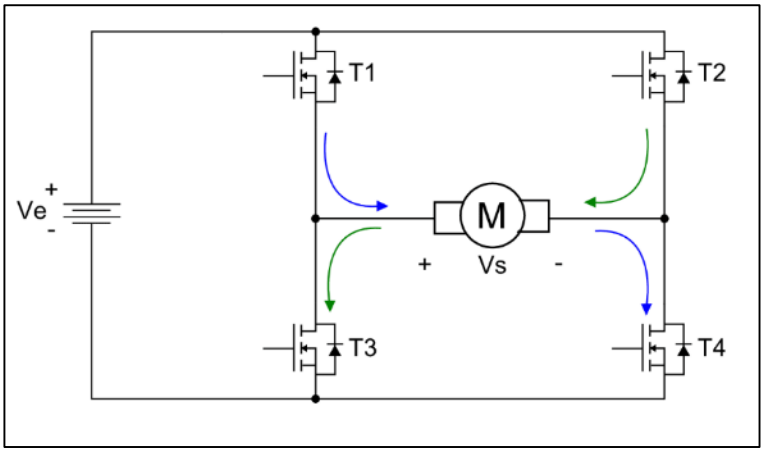


Figura 3.9 Esquema del circuito de un puente H o puente de transistores.

También es posible aplicar un frenado rápido mediante la activación de los transistores T1 y T2. Esta combinación es poco recomendada puesto que detiene bruscamente el motor, pudiendo producirse un daño mecánico en el eje o en el devanado interior. Además, es recomendable siempre pasar por el estado de libre inercia del motor antes de invertir su giro, para evitar la misma situación descrita.

Para el control del voltaje entregado al motor es necesario utilizar una señal de PWM en uno de los 2 transistores activos. Una señal de PWM es periódica y tiene un estado en alto y uno en bajo, los cuales son variables pero sumados dan un período. El ciclo de trabajo o *duty cycle* es la relación entre el ancho de pulso en estado encendido con relación al período, se expresa en porcentaje y es proporcional al voltaje promedio que se genera.

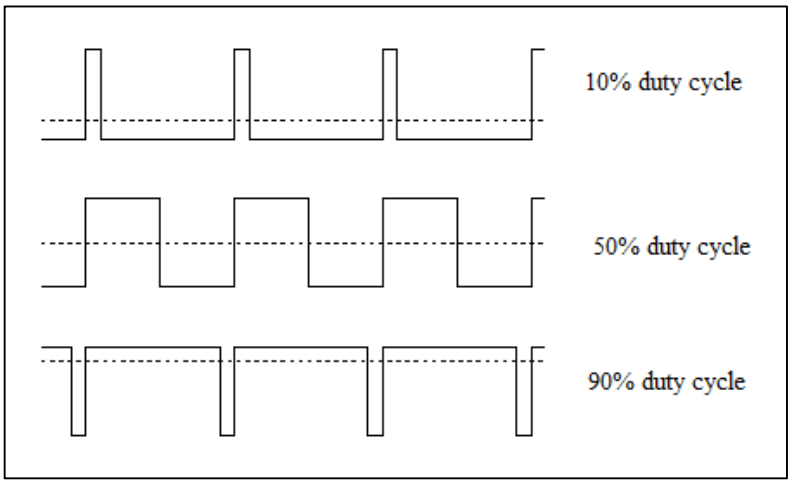


Figura 3.10 Ejemplos de ondas PWM con su promedio (línea discontinua).

Para que el voltaje promedio sea eficiente son necesarios: una onda de PWM de una frecuencia considerable (sobre los 10 kHz) y elementos de encendido y apagado eficientes a esa frecuencia con pérdidas mínimas debido a la corriente que soportan. Asimismo, debido al ruido de alta frecuencia que se genera, se producen pérdidas de potencia en los motores, lo cual debe ser solucionado con circuitos de filtrado.

Por estas razones, un circuito de estas características requeriría grandes costos y condiciones de diseño especiales (materiales de disipación, circuitos impresos), y no poseería la fiabilidad de dispositivos integrados comerciales con similares funciones. Además, este circuito debería ser duplicado exactamente, ya que cada uno controlaría un grupo de motores de cada lado.

Con esto presente, se ha buscado en el mercado electrónico componentes con estas capacidades, y que tengan un costo razonable para este proyecto. El componente elegido es el *shield* compatible con Arduino controlador de motores dual de Pololu[12], basado en dos chips VNH019[13] de la empresa STMicroelectronics. A continuación, se muestran algunas de las características de este chip:

- Puente H para control de motor, basado en transistores integrados MOSFET de potencia.
- Corriente máxima 30 A.
- Voltaje máximo 41 V.
- Resistencia del transistor en modo encendido de 18 mΩ.
- Control PWM de hasta 20 kHz.
- Protección contra sobrecargas.
- Medición de la corriente de salida del motor.
- Encapsulado diseñado para soportar y disipar el calor de forma eficiente.

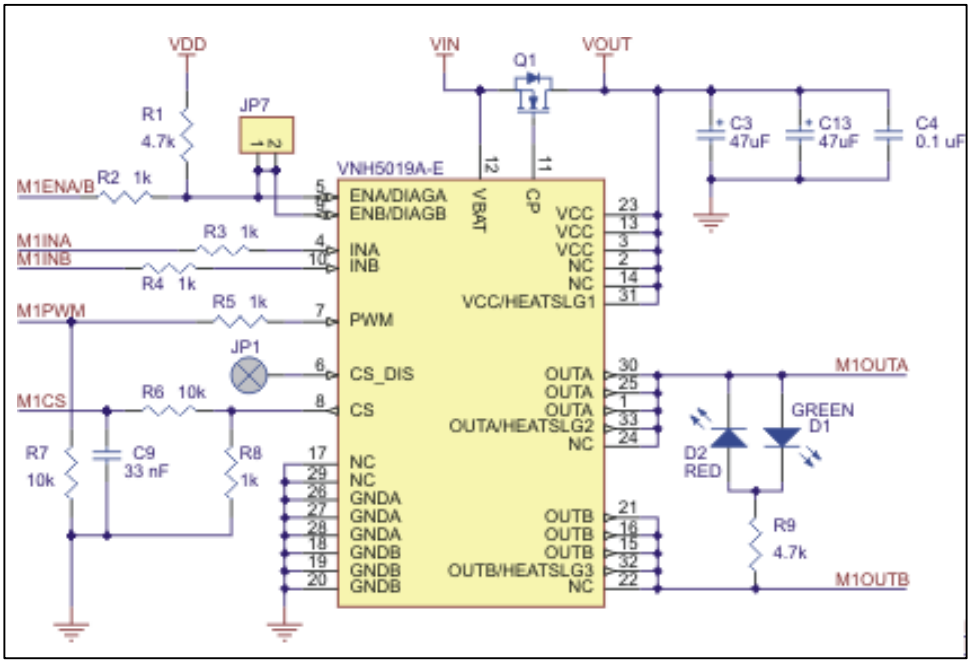


Figura 3.11 Esquema de conexión de chip VHN019 en placa Pololu.

[12] Pololu Corporation. *Pololu Dual VNH5019 Motor Driver Shield User's Guide.* [en línea]. [22 de diciembre de 2013]. Disponible en web: <http://www.pololu.com/docs/pdf/0J49/dual_vnh5019_motor_driver_shield.pdf>.

[13] STMicroelectronics. *VNH5019A-E Automotive fully integrated H-bridge motor driver.* [en línea]. 2010. [22 diciembre de 2013]. Disponible en web: <http://www.pololu.com/file/download/VNH5019A-E.pdf?file_id=0J504>.

Dos de estos chips se encuentran en el *shield* controlador de motores, y están conectados como se muestra en la Figura 3.11**.** Este *shield* es apto para Arduino UNO y Arduino MEGA, y ocupa los siguientes pines de la placa base (Tabla 3-1):

Pin Arduino	Pin Shield VNH5019	Función Básica
Digital 2	M1INA	Entrada A de Motor 1
Digital 4	M1INB	Entrada B de Motor 1
Digital 6	M1EN/DIAG	Habilitación de Motor 1
Digital 7	M2INA	Entrada A de Motor 2
Digital 8	M2INB	Entrada B de Motor 2
Digital 9	M1PWM	Velocidad Motor 1
Digital 10	M2PWM	Velocidad Motor 2
Digital 12	M2EN/DIAG	Habilitación de Motor 2
Analog 0	M1CS	Medición de Corriente 1
Analog 1	M2CS	Medición de Corriente 2

Tabla 3-1 Lista de entradas y salidas en placa de motores.

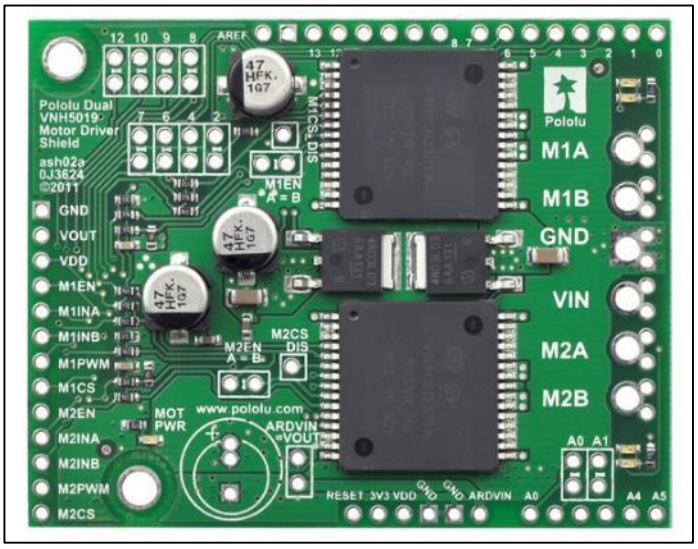


Figura 3.12 Shield Pololu dual VNH5019 motor driver para Arduino.

Fuente: Pololu Corporation. *Pololu dual VNH5019 motor driver shield for Arduino* [en línea]. [23 diciembre de 2013]. Disponible en web: <http://www.pololu.com/picture/view/0J3748>.

3.2.3.3. Pantalla

Con el objetivo de mostrar información directamente a través del robot, sin necesidad del equipo remoto, se implementará una pantalla interactiva, la cual permita la visualización de la información captada por los sensores. La pantalla permite al usuario acceder a través de su menú táctil, al monitoreo de la información captada por el robot.

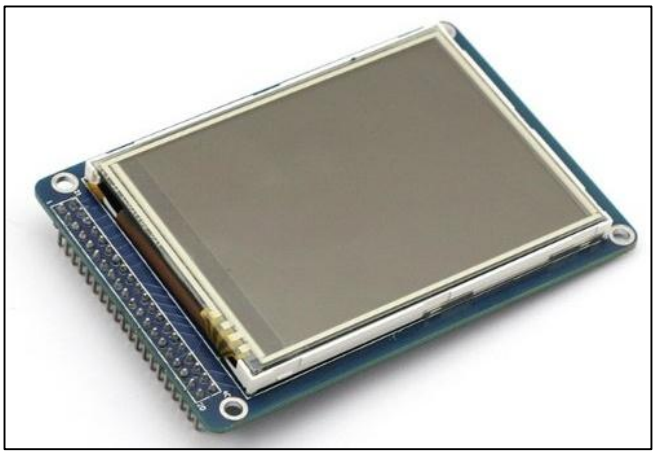


Figura 3.13 Pantalla Táctil SainSmart de 3,2 pulgadas.

Fuente: SainSmart. *SainSmart 3.2" SSD1289 TFT LCD Display Touch Screen* [en línea]. [23 diciembre de 2013]. Disponible en web: <http://www.sainsmart.com/arduino/arduino-shields/lcd-shields/sainsmart-3-2-tft-lcd-display-touch-panel-pcb-adapter-sd-slot-for-arduino-2560.html>.

Para ello se ha escogido la pantalla LCD TFT de 3,2” de la marca SainSmart, la cual posee una interfaz de 40 pines, un controlador de pantalla SSD1289 con control de 8 o 16 bits, y un controlador para pantallas táctiles ADS7843, el cual utiliza un bus de 4 pines de tipo Serial, ambos de fácil manejo mediante microcontroladores. A continuación, se enumeran sus principales características:

- Color de 65K.
- Resolución de 320 x 240.
- Tamaño de 3,2 pulgadas.
- Controladores integrados de LCD y pantalla táctil
- Conector para memoria SD, donde se pueden guardar imágenes y cargarlas a la pantalla.

Esta pantalla posee la ventaja de conectarse a una placa adaptadora y acoplarse a un Arduino Mega2560 o similar (Figura 3.14).

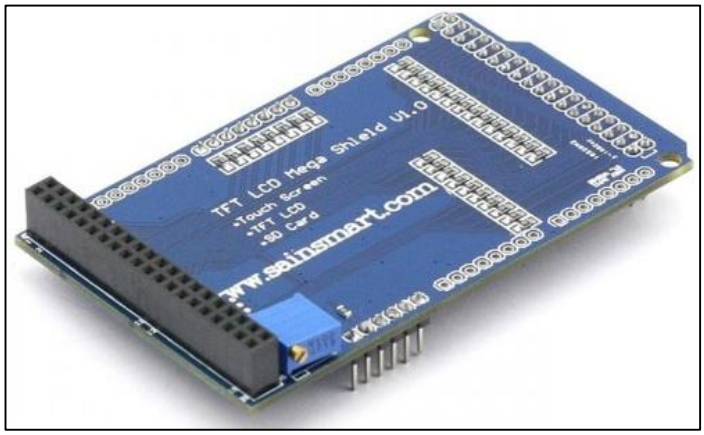


Figura 3.14 Placa SainSmart para pantalla LCD (Shield de Arduino Mega 2560).
Fuente: SainSmart. *SainSmart TFT LCD Adjustable Shield for Arduino Mega 2560 R3 1280 A082 Plug* [en línea]. [23 diciembre de 2013]. Disponible en web: <http://www.sainsmart.com/arduino/arduino-shields/sainsmart-tft-lcd-adjustable-shield-for-arduino-mega-2560-r3-1280-a082-plug.html>.

3.3. Sensores Ultrasónicos de Distancia

Para la actividad de mapeo es necesario utilizar sensores capaces de medir la distancia de los objetos del entorno. Los sensores ultrasónicos utilizan señales sonoras de alta frecuencia, las cuales rebotan en los objetos cercanos y son captados de vuelta en la fuente, en la cual se puede realizar un estimado de la distancia al objeto conociendo la velocidad promedio del sonido en el ambiente. Este método hace que las mediciones no sean afectadas por variaciones de luz o la opacidad de los objetos como con sensores infrarrojos. Sin embargo, se pueden producir errores de medición en ambientes con excesiva acústica o sobre materiales en los que el sonido es absorbido (algodón o plumas).

Figura 3.15 Sensor Ultrasónico de distancia HC-SR04.

Fuente: Cytron Technologies. *Product User's Manual – HC-SR04 Ultrasonic Sensor* [en línea]. [23 diciembre de 2013]. Disponible en web: <https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit?pli=1>.

Se ha elegido los sensores HC-SR04 debido a su disponibilidad y precio accesible. Al mismo tiempo, el consumo de energía permite su uso dentro de un equipo autónomo, no así sensores de tipo industrial que requieren fuentes de alimentación más potentes. A continuación, se listan algunas características de este sensor.

- Alimentación a 5 VDC.
- Consumo de corriente de 15 mA.
- Ángulo de acción menor a 15º.
- Medición de distancia de 2 a 400 cm.
- Resolución de 1 cm.

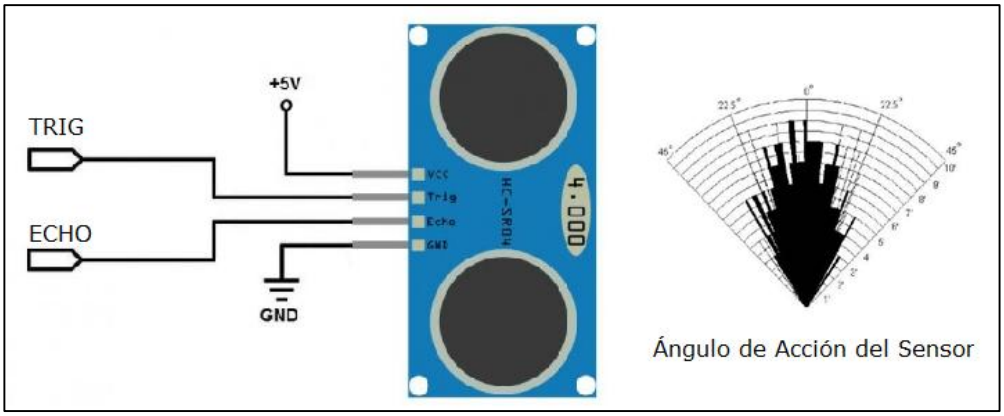


Figura 3.16 Conexión y ángulo de acción del sensor HC-SR04.

Fuente: Elecfreaks. *Ultrasonic Ranging Module HC - SR04* [en línea]. [23 de diciembre de 2013]. Disponible en web: <http://users.ece.utexas.edu/~valvano/Datasheets/HCSR04b.pdf>.

3.3.1. Funcionamiento del Sensor HC-SR04

Como se aprecia en la Figura 3.15, el sensor consta de un emisor de sonidos de alta frecuencia y un receptor o micrófono. Para iniciar la medición, se envía un pulso de 5 V durante 10 microsegundos en la entrada TRIG. Esto hace que el sensor inicie la transmisión de 8 pulsos de un sonido a 40 kHz a través del parlante. Cuando el sensor detecta el sonido devuelto por algún objeto cercano a través del micrófono, el pin ECHO se coloca en 5 V y se mantiene así por un tiempo proporcional a la distancia, la cual está dada por la fórmula:

$$distancia = (tiempo\ ECHO\ [s]) * \frac{vel\ sonido\left[\frac{m}{s}\right]}{2}$$

Si se toma la velocidad del sonido en 340 m/s, se mide el tiempo del pin ECHO en microsegundos y la distancia en centímetros, la fórmula puede simplificarse a:

$$distancia\ [cm] = (ECHO\ [us]) * \frac{0.034\left[\frac{cm}{us}\right]}{2} \qquad distancia\ [cm]\frac{(ECHO\ [us])}{58}$$

3.3.2. Ubicación de los sensores

Para conseguir un mapeo de todo el entorno del robot es necesario utilizar varios de los sensores de distancia ubicados estratégicamente para captar objetos cercanos en todas direcciones, evitando zonas sin mapeo o puntos ciegos que puedan generar errores.

Es así que se han colocado 6 sensores ultrasónicos de distancia como se muestra en la Figura 3.17, sobre una placa de expansión o *shield* para facilitar la realización de pruebas. En esta disposición, con dos sensores en la parte delantera, y dos en cada lado a 90 grados, existe, en teoría, una redundancia de la información tomada por los sensores en cada dirección debido a la proximidad de su pareja. Sin embargo, se pudo comprobar que los impulsos ultrasónicos de cada sensor producían interferencias con los de sus pares, generando errores en la medición.

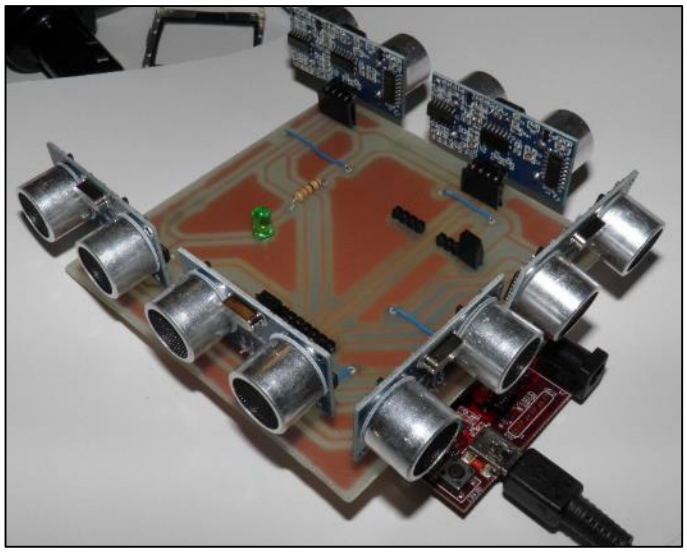


Figura 3.17 Placa de expansión (*shield*) de prueba para ubicación de sensores de distancia.

Posteriormente, se realizó un diseño basado en las pruebas anteriores, determinándose que los sensores deben apuntar en distintas direcciones y que deben estar a una distancia de al menos 4 cm entre ellos para evitar interferencias y aprovechar al máximo la información entregada por los mismos. En este caso se colocan los sensores como se muestra en la Figura 3.18.

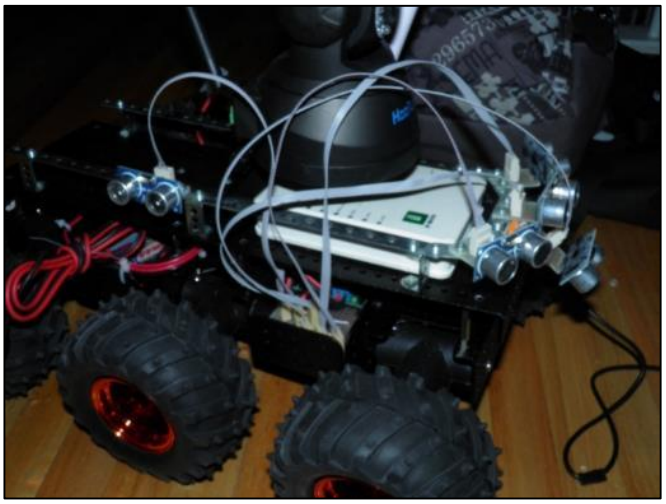


Figura 3.18 Disposición final de los sensores sobre chasis del robot.

En la Figura 3.19 se muestra la ubicación esquemática de los sensores sobre el robot. Aquí se colocan dos sensores a cada lado del robot a 90 grados, los cuales permiten mapear a izquierda y derecha del robot. El siguiente par de sensores se encuentra ubicado a 45 grados hacia el frente, los cuales tienen el objetivo de determinar objetos en las proximidades en diagonal al robot, ampliando su zona de mapeo. A continuación, se coloca un sensor en dirigido hacia el frente del robot y paralelo al piso, el cual tiene la función de mapeo y evasión de obstáculos. Por último, se coloca un sensor dirigido hacia el frente del robot, pero apuntando 45 grados hacia el piso, el cual tiene como objetivo indicar discontinuidades en el camino. Este último sensor es de vital importancia, puesto que el robot no está diseñado para salir de desniveles de gran altura como fosas o escalones de gran tamaño.

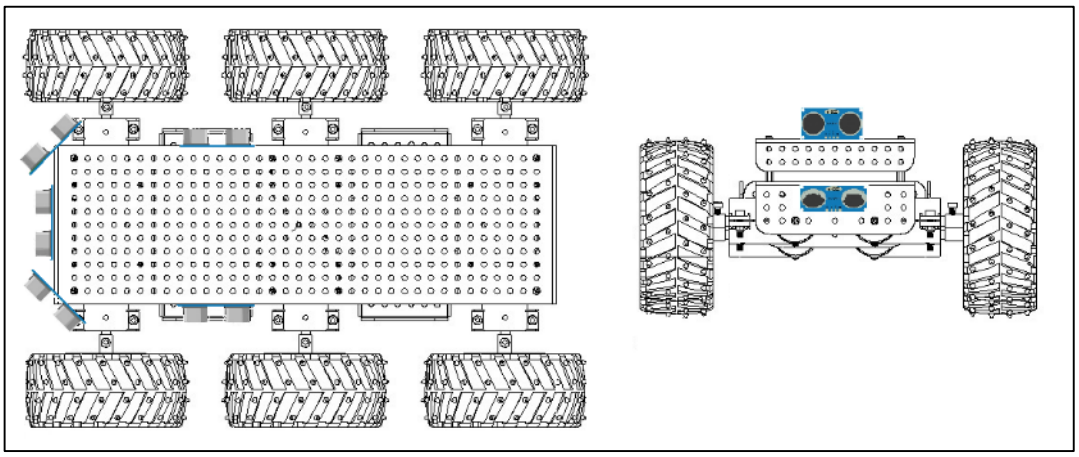


Figura 3.19 Esquema de ubicación de los sensores de distancia.

3.4. Sensores Ambientales

Con el objetivo de obtener información extra acerca de sus alrededores, se han tomado en cuenta sensores para variables críticas como la temperatura, humedad y la presencia de ciertos gases comunes en las minas, que pueden significar peligros para la actividad del robot.

3.4.1.1. Sensor de Temperatura y Humedad DHT-11

El dispositivo DHT-11 es un sensor integrado de temperatura y humedad, de tamaño reducido (menos de 2 cm^2) y que no requiere de elementos exteriores. Incluye mediciones de alta confiabilidad de la humedad a través de un método resistivo, y de la temperatura a través de un termistor. Estos elementos se encuentran conectados a un microcontrolador de 8 bits, el cual transmite los datos a través de un protocolo conocido como *1-Wire* o Serial de un solo cable. El sensor se encuentra protegido por un cobertor de goma suave, lo que da mayor resistencia mecánica a los elementos internos. A continuación, se listan algunas características técnicas importantes del sensor:

- Rango de medición del 20% al 90% de humedad relativa.
- Precisión en la medición de humedad de ±5%, con 1% de resolución.
- Rango de medición de temperatura de 0º a 50º centígrados
- Precisión de en la medición de temperatura de ±1ºC con 1ºC de resolución.
- Operación de 3 a 5 VDC, a 2,5 mA.
- Pin 1: VCC, pin 2 DATA, pin 3 No Conectado, Pin 4 GROUND (Figura 3.20).

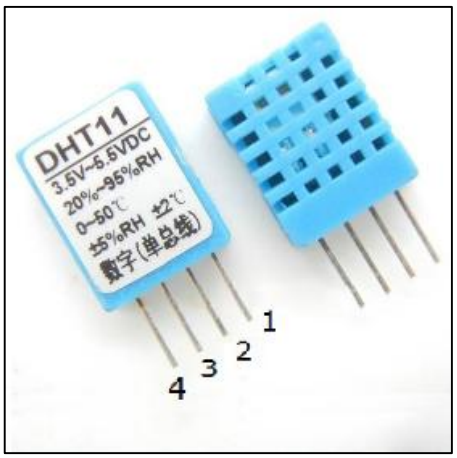


Figura 3.20 Sensor de temperatura y humedad DHT11.

Fuente: D-Robotics. *DHT11 Humidity & Temperature Sensor* [en línea]. 30 de diciembre de 2010. [1 enero 2014] Disponible en web: <http://www.micro4you.com/files/sensor/DHT11.pdf>.

En la Figura 3.21 se esquematiza la conexión del sensor a un microcontrolador. Cuando el cable de datos es menor a 20 metros se recomienda la conexión de un resistor de 5 kΩ hacia 5 VDC (resistor de *pull-up*).

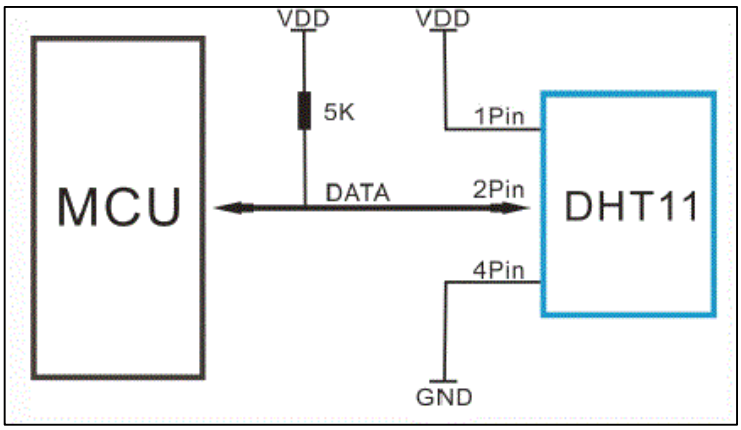


Figura 3.21 Esquema de conexión del sensor DHT11.

3.4.1.2. Sensor de Gas MQ-4

La presencia de gas natural es muy común dentro del ambiente de una mina, el cual está formado principalmente por gas metano (CH_4). El sensor MQ-4 posee material sensitivo basado en dióxido de estaño (SnO_2), el cual se encuentra en un micro tubo de cerámica que es calentado por un pequeño electrodo, generando las condiciones adecuadas para la medición. El sensor posee 6 pines, dos de los cuales se usan para proveer la corriente de calentamiento, y los otros 4 se usan para recoger las señales[14]. A su salida, el sensor genera una señal analógica, provocada por un cambio en la conductividad del elemento sensitivo cuando existe un cambio en la concentración de gas, en especial de los gases metano, propano y butano. Es poco sensible a los gases producidos por el alcohol, y poco sensible al humo. El sensor está cubierto por una malla de acero para proteger el material sensitivo, permitiendo a la vez la entrada de gas del ambiente.

[14] HANWEI ELECTRONICS. *Technical Data MQ-4 Gas Sensor* [en línea]. [20 enero de 2014]. Disponible en web: <https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-4.pdf>.

Figura 3.22 Sensor de gas metano MQ-4.
Fuente: Sparkfun. *Methane CNG Gas Sensor - MQ-4* [en línea]. [20 enero de 2014]. Disponible en web: <https://www.sparkfun.com/products/9404>.

El sensor se encuentra colocado en una placa de circuito impreso, con un resistor para calibrar la sensibilidad y un amplificador operacional en forma de comparador. Esto permite una salida digital cuando se alcanza el nivel calibrado (con un indicador LED). Su salida analógica de 0 a 5 V es proporcional a la concentración de gas detectada. A continuación, se muestran las características principales del sensor colocado en la placa de circuito impreso:

- Rango de medida: de 200 a 10 000 partes por millón (ppm).
- Operación a 5 VDC, a 100 mA.
- Pin 1: VCC, pin 2 DOUT (salida digital), pin 3 AOUT (salida analógica), Pin 4 GROUND.

Figura 3.23 Sensor MQ-4 en placa de circuito impreso.

3.5. Giroscopio

El giroscopio es un dispositivo que permite medir la orientación del objeto que lo contiene. Debido a su carácter mecánico, los giroscopios siempre fueron sensores de gran tamaño, delicados y basados en pesos. Sin embargo, en la actualidad, debido a la creación de sistemas microelectromecánicos o MEMS, los sensores se miniaturizaron y se encapsularon junto a otros elementos (memorias, microcontroladores, sistemas de comunicación). Uno de estos elementos es el sensor MPU-6050, el cual combina un giroscopio de 3 ejes, un acelerómetro de 3 ejes y un procesador digital especializado en el mismo encapsulado de silicio, el cual se comunica con el exterior mediante una interfaz I2C.

Figura 3.24 Giroscopio MPU-6050 en placa de circuito impreso.
Fuente: Arduino Playground. *MPU-6050 Accelerometer + Gyro* [en línea]. [20 enero de 2014]. Disponible en web: <http://playground.arduino.cc/Main/MPU-6050>.

Este chip se encuentra integrado en una pequeña placa de circuito impreso, la cual permite una conexión más sencilla con los pines de comunicación y posee un LED que indica su funcionamiento. El chip posee integrada una memoria FIFO (*First In, First Out*) de 1024 bytes que permite guardar los datos captados por el sensor en grandes ráfagas, para luego ser enviado a través del sistema de comunicación, sin producir retrasos.

3.6. Cámara IP

Con el objetivo de monitorear mediante video el entorno del robot, se incorpora una cámara de video IP, la cual transmite las imágenes mediante red Ethernet por un cable RJ-45. Para ello se ha elegido la cámara HooToo HT-IP206, la cual posee las siguientes características:

- Conectividad Wireless tipo N y conectividad mediante cable de red.
- Visión nocturna mediante 10 LED de luz infrarroja y sensor de luz.
- Cámara de 0,3 Megapíxeles con resolución de hasta 640 x 480 a 30 cuadros por segundo en formato MJPEG.
- Configuración de alarmas por detección de movimiento.
- Personalización sencilla de características mediante interfaz de red.
- Doble servomecanismo para control de inclinación (120º) y orientación (270º) mediante mensajes tipo GET.

Figura 3.25 Cámara IP HooToo HT-IP206.

Fuente: HooToo. *IP Wireless / Wired Camera Remote Pan/Tilt rotate User Manual* [en línea].2010. [13 enero de 2014]. Disponible en web: <http://www.hootoo.com/media/upload/support/HooToo%20HT-IP206%20User%20Manual.pdf>.

3.7. Punto de acceso inalámbrico

Para enviar los datos desde el Ethernet *shield* de Arduino y las imágenes captadas por la cámara, es necesario tener un equipo que capte estas dos fuentes de información y que las transmita de forma segura sin cables hacia un equipo remoto. Además, debe permitir que este equipo remoto envíe datos de regreso para el control del robot o de la cámara. Un equipo con estas características es un punto de acceso inalámbrico, el cual interconecta dispositivos alámbricos con inalámbricos en una misma red. En este caso, el punto de acceso inalámbrico o WAP (*Wireless Access Point*) es parte del robot, y tanto el Ethernet *shield* como la cámara estarán conectados a él mediante cables de red.

Figura 3.26 Punto de Acceso Inalámbrico QPCOM QO-WA252G.
Fuente: QPCOM Integral Networking Solutions. *User's Manual QP-WA252G 802.11b/g Wireless Access Point* [en línea]. 2012. [13 enero de 2014]. Disponible en web: <http://www.qpcom.com/LinkClick.aspx?fileticket=7QlpiXoH45U%3d&tabid=4700&mid=13110>.

El equipo elegido es un punto de acceso inalámbrico QP-WA252G de QPCOM con capacidad de conexión con redes tipo b/g de 11 Mbps y 54 Mbps con antena de rosca intercambiable. Posee dos puertos Ethernet 10/100M LAN con conector RJ-45 y seguridad WEP y WPA. Es de tamaño y peso reducidos, con un funcionamiento de 7 a 12 VDC y un consumo máximo de 0,8 A. Su rango de alcance es de 100 metros en interiores y 300 metros en exteriores, sin embargo, esto puede ser mejorado con la conexión de una antena de mayor potencia.

3.8. Baterías

El robot debe poseer autonomía energética, es decir, debe llevar consigo una fuente de energía para funcionar por un período considerable, sin la necesidad de cables. Para ello, en primer lugar, es necesario tomar en cuenta el consumo de energía de los componentes antes mencionados que están incluidos en el robot. En la Tabla 3-2 se muestran los consumos de los elementos incluidos en el robot. Se ha considerado una corriente de 1,1 A máximo por motor y no los 6 A indicados en sus especificaciones, puesto que es una medida más cercana al real funcionamiento cuando los 6 motores se reparten la carga del chasis del robot.

DISPOSITIVO	CONSUMO MÁXIMO mA
Arduino Uno	200
Arduino Mega Pantalla	500
Arduino Mega Ethernet	500
chipKIT UNO32	300
Ethernet Shield	800
Pololu Dual Motor Shield	20
Sensor Ultrasónico x6	90
Sensor DHT11	0,3
Sensor Gas	100
Giroscopio	100
Cámara IP	500
Access Point Inalámbrico	800
Pantalla Táctil	200
Motor x6	6600
TOTAL	**10710,3**

Tabla 3-2 Consumo de energía de los dispositivos electrónicos dentro del robot.

Con este dato se ha decidido utilizar dos baterías de 7,2 V Tenergy recargables con capacidad de 3 800 mAh de NiMH, con corriente máxima de 38 A, diseñadas para autos de control remoto y robots pequeños, lo que permite un funcionamiento normal (sin caídas de voltaje) a máximo consumo. La capacidad total de las baterías suma 7200 mAh.

Figura 3.27 Batería recargable de NiMH Tenergy de 3 800 mAh.

Tomándose en cuenta el consumo máximo en unos 10,71 A, las baterías permitirán una autonomía de unos 42 minutos. Este puede considerarse como un dato mínimo, puesto que la autonomía real será mucho mayor debido a que el robot no siempre se encuentra trabajando a máxima potencia.

3.9. Interconexión del sistema

Todos los elementos antes descritos deben formar un solo conjunto dentro del robot. Por lo tanto es necesario realizar la conexión física y eléctrica entre todos estos elementos de acuerdo a las necesidades. Para ello se toma en cuenta lo siguiente:

- La placa controladora de motores debe obtener de primera mano la información de los sensores que ayudan en la ubicación (sensores de distancia y giroscopio), con el objetivo de que, si se produce una caída de comunicación con el equipo remoto, sea posible la navegación independiente.

- El giroscopio demanda gran capacidad de procesamiento a través del bus I2C, por lo que es conveniente que sus datos sean procesados por una placa controladora de manera exclusiva.

- La placa controladora de la pantalla debe tener forma de Arduino Mega 2560 debido a su interfaz de 16 bits.

- El controlador de la placa Ethernet debe tener comunicación con el controlador de la pantalla y con el controlador de los motores, puesto que envía señales de control captadas desde el equipo remoto, envía los datos captados por los sensores a través del punto de acceso inalámbrico y hacia la pantalla, y los almacena en la memoria microSD.

- Los sensores ambientales, los cuales no son de prioridad crítica, serán controlados por cualquier dispositivo tomando en cuenta su disponibilidad física y de procesamiento.

- La comunicación entre las placas de control será de tipo Serial, y su velocidad será determinada en el la creación de los programas.

- Ya que cada placa controladora posee un regulador lineal, el voltaje de las baterías se conecta directamente puesto que se encuentra dentro de los límites seguros de funcionamiento de cada regulador.

- Debido a la gran cantidad de conexiones distintas entre los dispositivos, es necesario crear placas de adaptación, cables y conectores, los cuales permitan a cada conexión ser de fácil acceso, segura y confiable.

Las placas de adaptación son circuitos impresos realizados en el software de National Instruments NI Multisim (entorno de simulación y creación de circuitos electrónicos) y NI Ultiboard (entorno para el diseño de circuitos impresos). Debido al tamaño, necesidad de elementos de conexión simples y posibles modificaciones a causa del espacio dentro del robot, las placas de adaptación serán realizadas con métodos artesanales, realizando pruebas sencillas de su correcto funcionamiento.

Con todos estos puntos en consideración, el esquema de conexión de todo el sistema se muestra en la Figura 3.28.

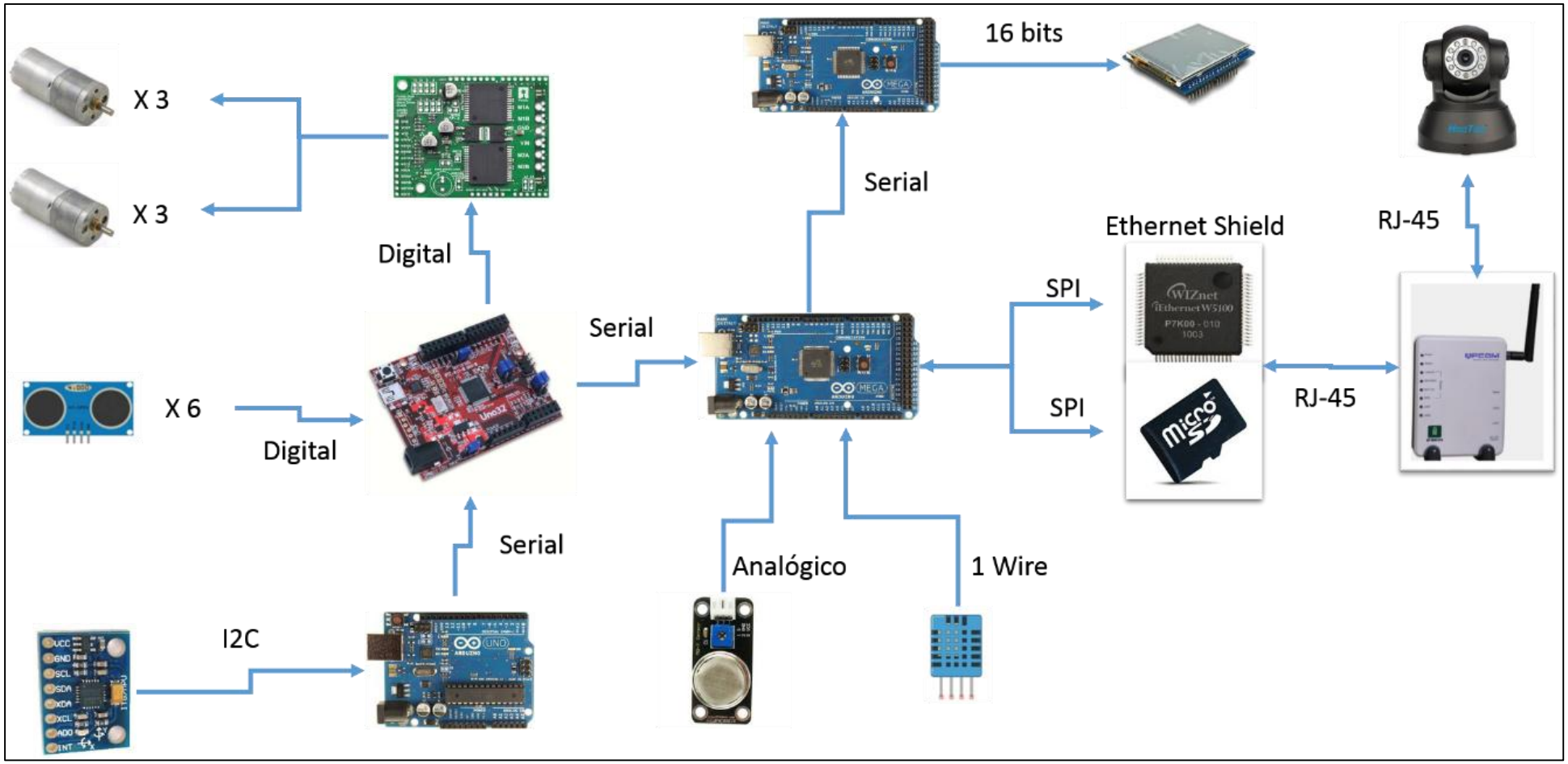


Figura 3.28 Esquema de la conexión del sistema electrónico dentro del robot.

3.9.1. Placa adaptadora para controlador de motores y sensores de distancia

Se ha escogido la placa chipKIT Uno32 como la placa controladora de motores debido a su velocidad de ejecución y cantidad de pines de entrada-salida, puesto que ésta debe adquirir la información captada por los 6 sensores de distancia.

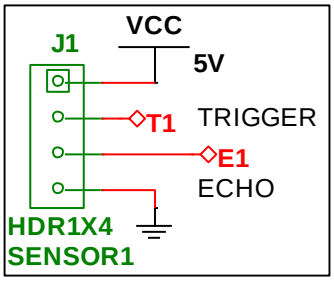


Figura 3.29 Esquema de conexión para sensor de distancia en NI Multisim.

La placa de adaptación creada para este conjunto conecta los 6 sensores de distancia a pines libres de la placa controladora. Además, está diseñada para permitir la conexión del *shield* controlador de motores sobre la misma, sin modificar su funcionamiento. Es importante que no se vea afectada la conexión del *shield,* puesto que cualquier cambio en la ocupación de pines puede crear problemas en el momento del desarrollo del software.

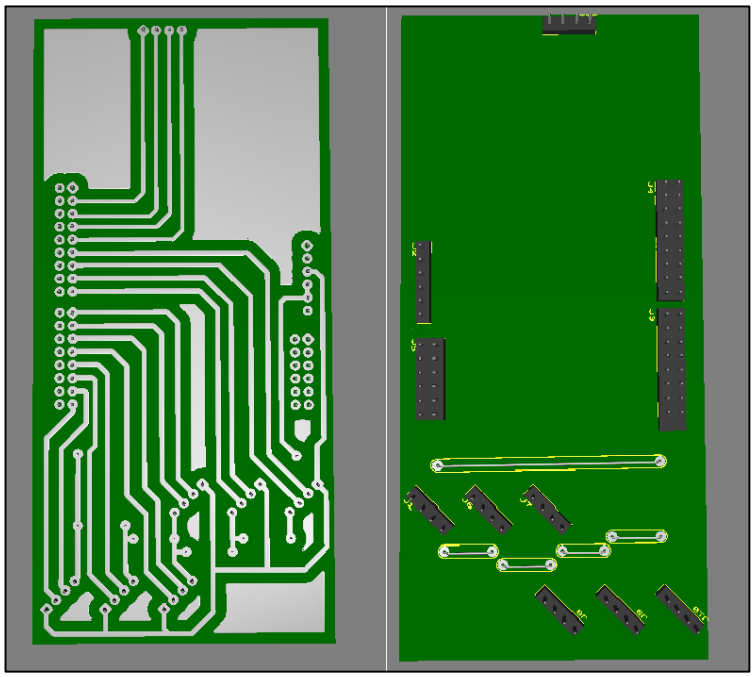


Figura 3.30 Vista previa 3D del diseño creado en NI Ultiboard.

Debido a que los sensores se encuentran lejos del lugar de conexión, es necesario utilizar conectores que no permitan errores de enlace con la placa en el proceso de instalación, como los que se muestran en la Figura 3.31.

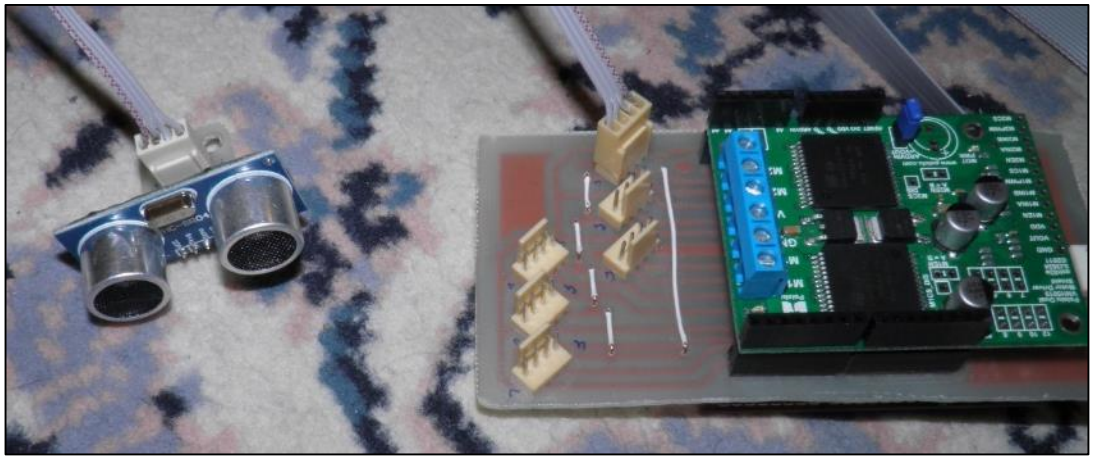


Figura 3.31 Placa de adaptación para sensores de distancia.

Como se puede apreciar en la Figura 3.32, la placa se ha diseñado alargada para aprovechar el espacio existente en el chasis del robot, y permitir, además, la conexión de los motores a través del conector central (blanco).

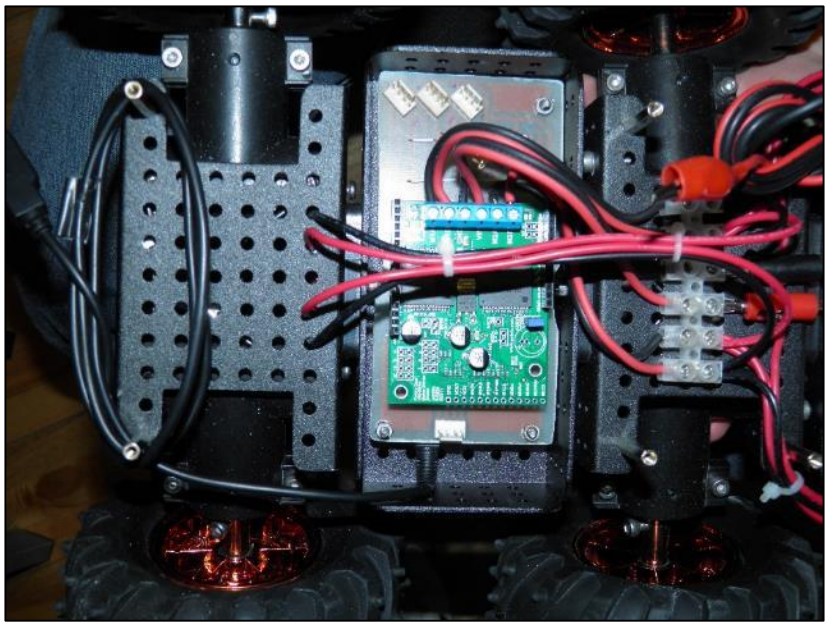


Figura 3.32 Ubicación de la placa controladora de motores dentro del robot.

En el anexo 3 se muestra el circuito esquemático de esta placa.

3.9.2. Placa adaptadora para Ethernet y sensores ambientales

Se ha escogido la placa Arduino Mega 2560 para controlar el *shield* Ethernet y los sensores ambientales. Aunque con la cantidad de entradas-salidas dentro de una placa más pequeña (como un Arduino UNO) es posible controlar tanto los sensores como la comunicación Ethernet, este elemento, al ser el centro de las comunicaciones internas, puesto que por aquí pasan los datos de control y de información de los sensores, necesita de al menos dos sistemas de comunicación Serial físicos, una cantidad considerable de memoria para procesar los datos enviados y recibidos, y pines de entrada-salida para los sensores, prestaciones no disponibles en la placa Arduino UNO.

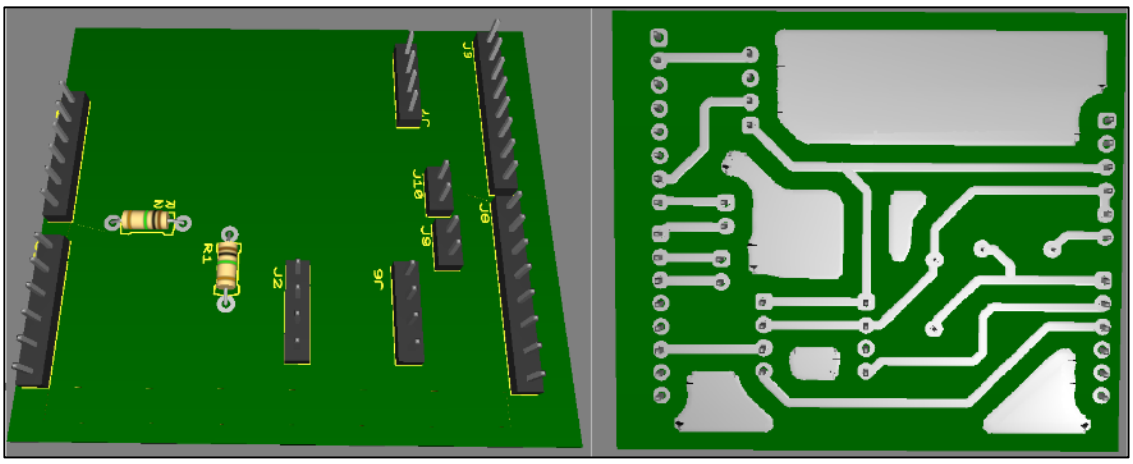


Figura 3.33 Vista previa 3D del diseño creado en NI Ultiboard.

De igual manera que en la placa anterior, se han utilizado conectores que no permitan error en el enlace, puesto que los sensores ambientales deben estar colocados en la parte externa del robot para captar adecuadamente las variables del ambiente.

En el anexo 4 se muestra el circuito esquemático de esta placa.

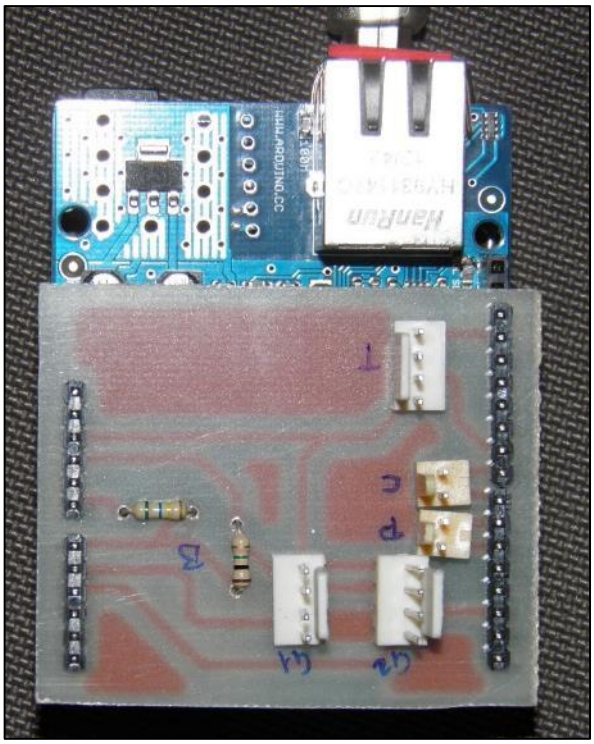


Figura 3.34 Placa de adaptación para sensores ambientales sobre *shield* Ethernet.

3.9.3. Placa adaptadora para controlador de giroscopio

Debido a la gran demanda de procesamiento por parte del giroscopio electrónico MPU-6050, se ha colocado a este sensor sobre una placa Arduino UNO, la cual enviará los datos de la orientación de manera simplificada mediante una conexión Serial a la placa controladora de motores.

La placa creada para conectar el giroscopio tiene la función de facilitar su enlace con el controlador Arduino Uno y también la de darle un adecuado soporte mecánico, puesto que es crucial que este sensor esté firmemente adherido al robot para que sea activado solo por los movimientos del mismo. Además, en esta placa se han incluido un circuito de reinicio mediante un pulsante y un indicador LED que serán utilizados para realizar pruebas posteriores de software.

En el anexo 5 se muestra el circuito esquemático de esta placa.

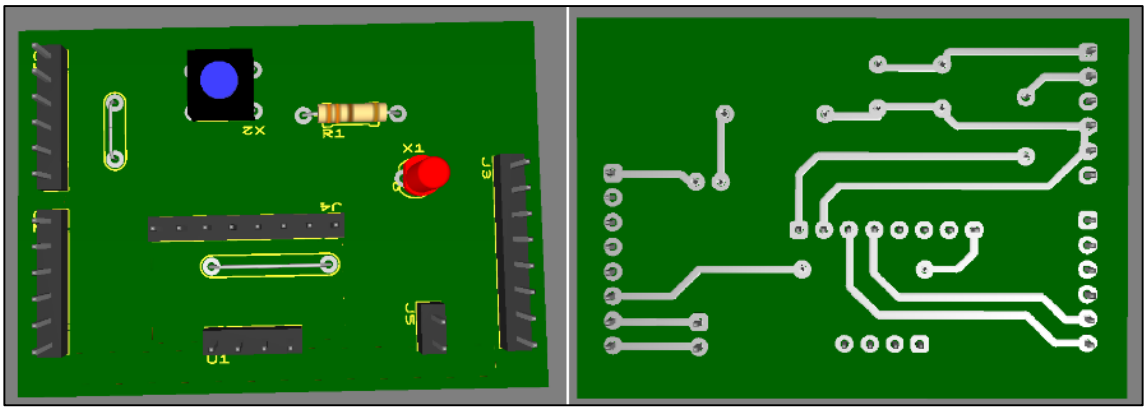


Figura 3.35 Vista previa 3D del diseño creado en NI Ultiboard.

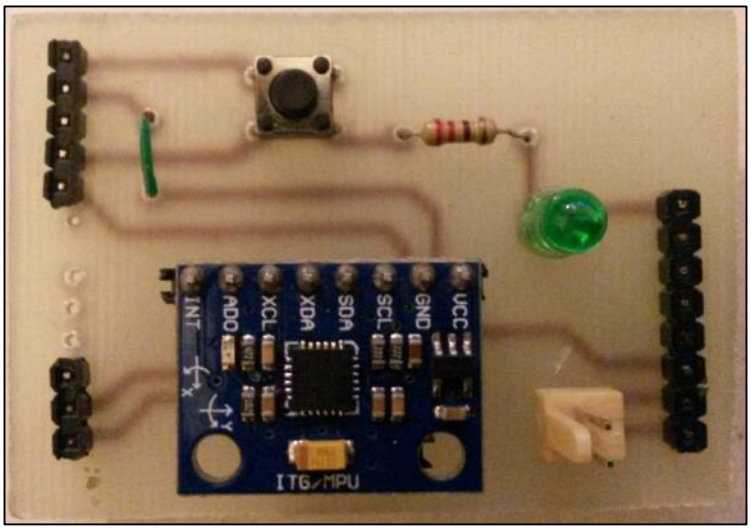


Figura 3.36 Placa de adaptación con giroscopio.

La placa controladora del giroscopio y su conjunto se colocan en la parte de arriba del robot, con mayor facilidad de acceso para posibles calibraciones en la posición del sensor (Figura 3.37).

Figura 3.37 Ubicación del controlador y giroscopio dentro del robot.

3.9.4. Ubicación y conexión de otros dispositivos

El punto de acceso inalámbrico es uno de los elementos de mayor tamaño, por lo que ha sido colocado sobre el chasis del robot de tal manera que sus puertos LAN y el conector de la antena sean de fácil acceso.

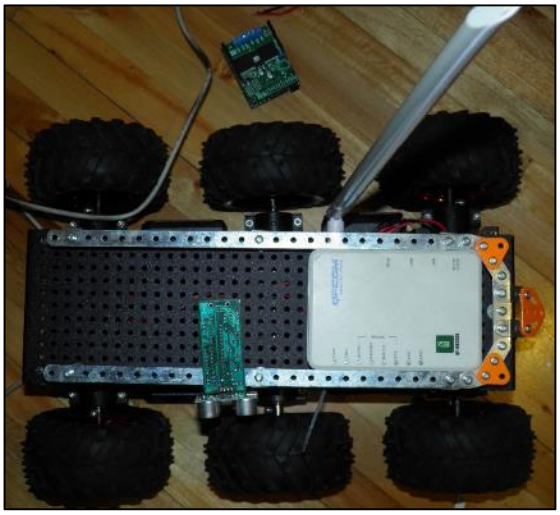


Figura 3.38 Ubicación del punto de acceso inalámbrico con antena de mayor ganancia.

La cámara ha sido colocada sobre el punto de acceso inalámbrico a través de tornillos en la parte más alta, para que su visión no se vea afectada por ningún elemento dentro del robot.

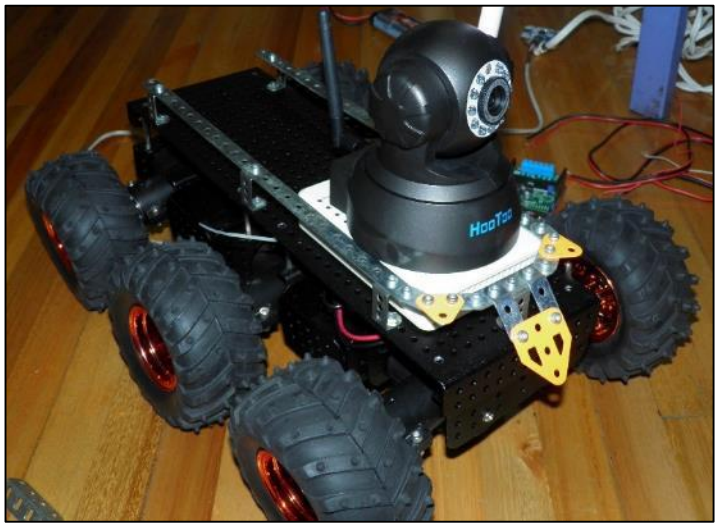


Figura 3.39 Ubicación de la cámara sobre el robot.

Los sensores ambientales y de distancia, debido a su ubicación exterior, fueron soldados sobre pequeñas placas de circuito impreso que permiten su conexión mediante cables y conectores sencillos, y que, además, permiten su sujeción a la estructura del robot mediante tornillos. Es importante indicar que en cada uno de ellos se ha colocado láminas de papel aislante para evitar posibles circulaciones de corriente a través de la estructura metálica del robot. Además, cada cable se ha cubierto con cinta espiral protectora, y en su trayectoria ha sido guiado mediante amarras de plástico.

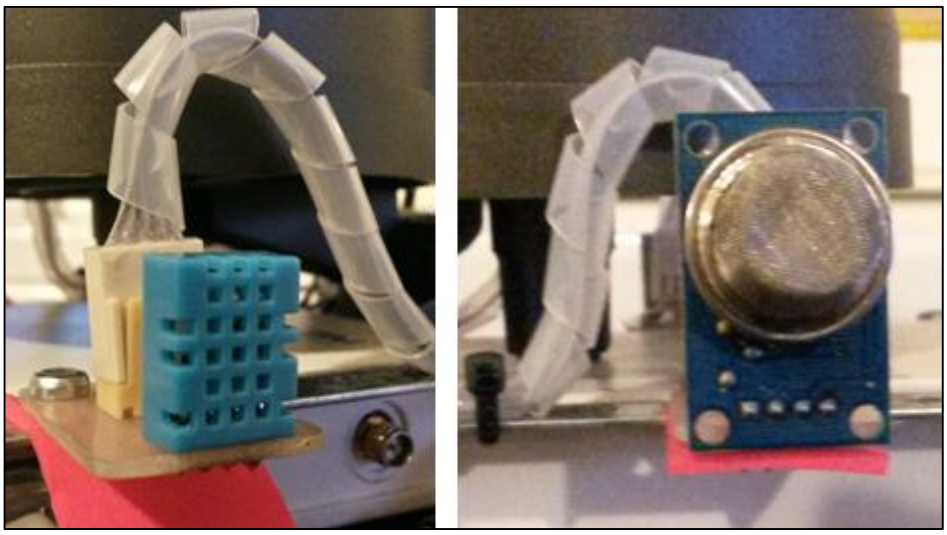


Figura 3.40 Sensores ambientales ubicados en el chasis del robot.

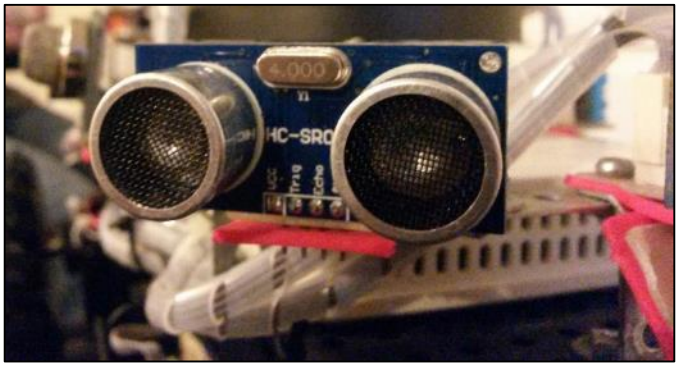


Figura 3.41 Sensor de distancia ubicado en el chasis del robot.

Para el encendido y apagado del robot se han implementado dos interruptores de dos posiciones cada uno (Figura 3.42). En la primera posición las baterías son conectadas a todos los reguladores y adaptadores de los dispositivos, encendiendo el robot. En la segunda posición, las baterías son desconectadas de los dispositivos del robot y se enlazan a dos conectores de carga, como se ve en la Figura 3.43.

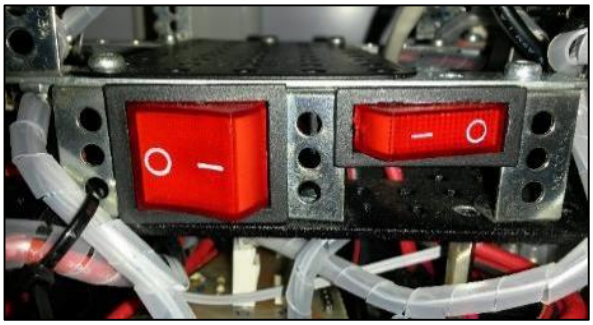


Figura 3.42 Interruptores de encendido-recarga del robot.

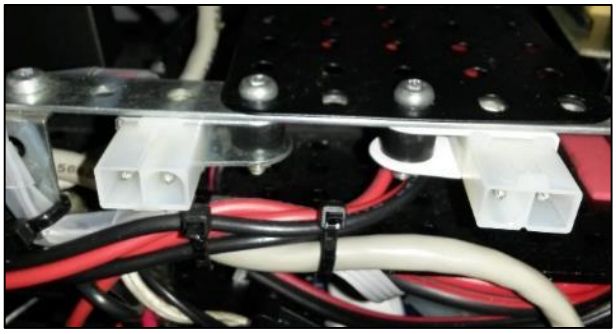


Figura 3.43 Conectores para la recarga de baterías.

La pantalla táctil ha sido colocada en la parte superior del robot para fácil acceso y visualización por parte del operador. El robot, con todos sus dispositivos electrónicos enlazados entre sí y colocados sobre el chasis, se muestra en las siguientes imágenes.

Figura 3.44 Vista frontal del robot

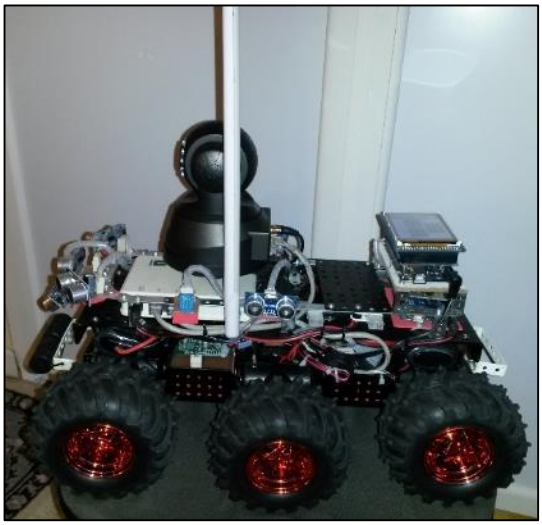


Figura 3.45 Vista lateral del robot, con antena de alta ganancia.

Figura 3.46 Vista posterior del robot, donde se aprecia pantalla LCD.

CAPÍTULO 4

PROGRAMA DE CONTROL DEL ROBOT

En este capítulo se explica el desarrollo de los programas en las placas controladoras. El objetivo es dotar al robot de una lógica de control de movimiento basada en sus sensores de posición y distancia. También debe configurarse un sistema de comunicación interna y externa, que permita la adquisición e intercambio de datos de manera ordenada y segura.

4.1. Entorno de Desarrollo Integrado

Un entorno de desarrollo integrado, conocido por sus siglas en inglés como IDE (*Integrated Development Environment*), es una aplicación de software que proporciona facilidades a los programadores para el desarrollo de nuevo software.

Normalmente, consiste en un editor de código fuente, herramientas de programación y un depurador. Actualmente, los nuevos IDE contienen un compilador intérprete y un examinador de objetos diseñados para maximizar la productividad del programador.

El objetivo principal de un IDE es reducir la configuración necesaria para construir un programa, con ello se reduce el tiempo utilizado, llegando así a obtener programas muy desarrollados y legibles en poco tiempo.

4.1.1. IDE de Arduino

Arduino es una plataforma de hardware y software libre, consistente en una placa de entradas y salidas, descrita anteriormente (hardware), y un entorno de programación basado en Wiring/Processing (software).

El lenguaje de programación del microcontrolador dentro de Arduino está basado en Wiring, que es una aplicación escrita en Java, pero su IDE fue desarrollado en Processing para facilitar su escritura.

Processing es un lenguaje de programación de código abierto y basado en Java, especialmente diseñado para programadores no expertos como artistas o diseñadores gráficos, que permite crear proyectos multimedia de manera sencilla. Su IDE se encuentra muy desarrollado y ha servido como base para el IDE de Arduino. A continuación, se presenta una pequeña muestra de algunas diferencias entre estos lenguajes[15].

Arduino	Processing
Matrices (Array)	
Int bar[8];	int[] bar = new int[8];
bar[0] = 1;	bar[0] = 1;
int foo[] = { 0, 1, 2 };	int foo[] = { 0, 1, 2 };
Bucles	
int i; for (i = 0; i < 5; i++) { ... }	for (int i = 0; i < 5; i++) { ... }
Impresión	
Serial.println("hola mundo");	println("hola mundo");
int i = 5; Serial.println(i);	int i = 5; println(i);
int i = 5; Serial.println(); Serial.print("i = "); Serial.println(i); Serial.println();	int i = 5; println("i = " + i);

Tabla 4-1 Comparación entre los lenguajes Arduino y Processing.

El entorno de desarrollo Arduino contiene un editor de texto para la escritura de código, un área de mensajes, una consola de texto, una barra de herramientas con botones para funciones comunes, y una serie de menús. Se conecta con el hardware Arduino para cargar programas y comunicarse con ellos[16].

[15] Arduino – Comparison [en línea]. 2012. [22 enero de 2014]. Disponible en web: http://arduino.cc/es/Reference/Comparison?from=Main.ComparisonProcessing

[16] Arduino – Enviroment [en línea]. 2012. [22 enero de 2014]. Disponible en web: <http://arduino.cc/es/guide/Environment>.

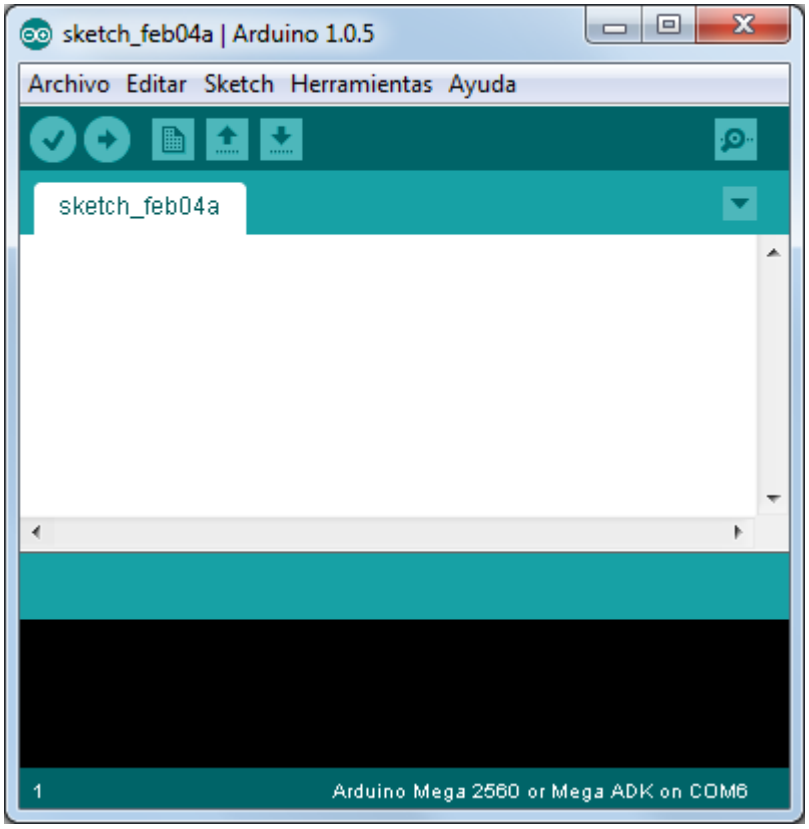


Figura 4.1 Ventana del IDE de Arduino.

Arduino utiliza los denominados *sketchs* para escribir los programas. Posee también una consola para poder revisar los errores de compilación y una barra de herramientas que se describirá a continuación.

- Verify/Compile: Chequea el código en busca de errores.
- Upload: Compila y graba en la placa Arduino el *sketch* programado.
- New: Crea un nuevo *sketch*.
- Open: Presenta un menú de todos los programas *sketch* de su *sketchbook*, (librería de *sketchs*).
- Save: Guarda el programa *sketch*.
- Serial Monitor: Inicia la monitorización por el puerto Serie.

4.1.2. MPIDE

Como se explicó anteriormente, la plataforma chipKIT es una placa de desarrollo basada en Arduino pero con prestaciones de un microcontrolador de 32 bits.

El IDE para programación se denomina MPIDE, es de distribución libre y tiene una interfaz similar a la de Arduino. Posee una barra de herramientas y una barra de menús similar a la de Arduino.

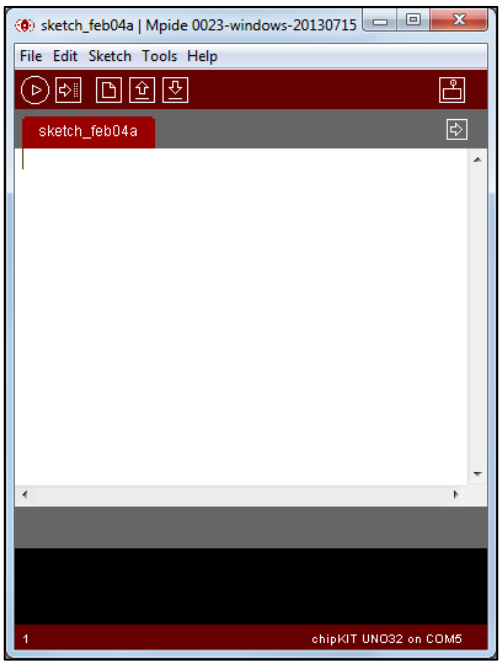


Figura 4.2 Ventana de MPIDE para programación de chipKIT.

4.1.3. Librerías

Son un conjunto de subprogramas que se utilizan para hacer más comprensivo el desarrollo de un software determinado. Las librerías contienen programas independientes que pasan a ser parte de un programa en desarrollo, esto permite que el software sea modificado por bloques o variables de paso.

Las librerías, tanto en Arduino como en MPIDE, son elaboradas para cada microcontrolador o para una función específica del mismo. Un ejemplo de esto se encuentra en la Figura 4.3.

```
#include <SD.h>

const int chipSelect = 4;

void setup()
{

  Serial.begin(9600);
  pinMode(10, OUTPUT);

  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
      return;
  }
  Serial.println("card initialized.");{

}

void loop()
{

  String dataString = "";

  for (int analogPin = 0; analogPin < 3; analogPin++) {
    int sensor = analogRead(analogPin);
    dataString += String(sensor);
    if (analogPin < 2) {
      dataString += ",";
    }
  }


  File dataFile = SD.open("datalog.txt", FILE_WRITE);

  if (dataFile) {
    dataFile.println(dataString);
    dataFile.close();
    Serial.println(dataString);
  }

  else {
    Serial.println("error opening datalog.txt");
  }
}
```

Figura 4.3 Ejemplo de código en Arduino o MPIDE.

En este caso se observa que para poder manejar una memoria SD, es necesario importar la librería SD y la librería SPI, la primera maneja las funciones específicas de la memoria microSD, mientras que la segunda ayuda en la comunicación de la misma con el microcontrolador a través del protocolo SPI.

En el ejemplo se puede observar cómo se inicializa una librería (#include <Librería.h>) en el *sketch* principal para luego ser utilizada en la programación; en este caso se muestra el uso de la librería SD y DHT11, que sirve para poder manejar a un nivel más alto la programación, ahorrando tiempo y facilitando su comprobación.

4.1.4. Esquema del programa

Para poder programar sobre Arduino son necesarias dos funciones: setup() y loop().

La función setup() es invocada al inicio por el microcontrolador para inicializar variables, llamar a librerías, establecer las configuraciones de los pines como entradas o salidas, etc., y se ejecuta una sola vez al iniciar el programa o cuando el microcontrolador sufre un reinicio o *reset*, ya sea por pulsar el botón de la placa o por programación.

Después de crear la función setup(), se establece la función loop(), la cual se ejecuta en un bucle infinito o hasta ser detenida. Dentro de esta función se coloca toda la programación a ser ejecutada constantemente. El programa puede salir de esta función si se produce alguna interrupción externa o interna, previamente activada. Después de procesada la interrupción, el proceso continua en la función loop().

4.2. Programas para el control, comunicaciones y adquisición de datos

Después de conocer las funciones básicas tanto de Arduino como de chipKIT, se describe a continuación la programación de cada microcontrolador con sus respectivas aplicaciones. Para mayor facilidad y generalidad, se analizarán los programas a base de diagramas de flujo.

4.2.1. Programa de la placa de control de motores

En primer lugar se llama a la librería para uso del *shield* de motores DualVNH5019MotorShield.h, y para el uso de los sensores ultrasónicos, Ultrasonic.h. A continuación se establecen los pines que serán utilizados para los sensores ultrasónicos. Cada sensor utiliza 2 pines: uno como entrada para ECHO y otro como salida para TRIGGER. Los pines ocupados van desde el 26 hasta el 37. Además, se declaran las variables que almacenarán las distancias obtenidas por los sensores y las tramas de comunicación recibidas a través del Serial1 (placa controladora de Ethernet) y Serial (placa controladora de giroscopio).

4.2.1.1. Función Setup

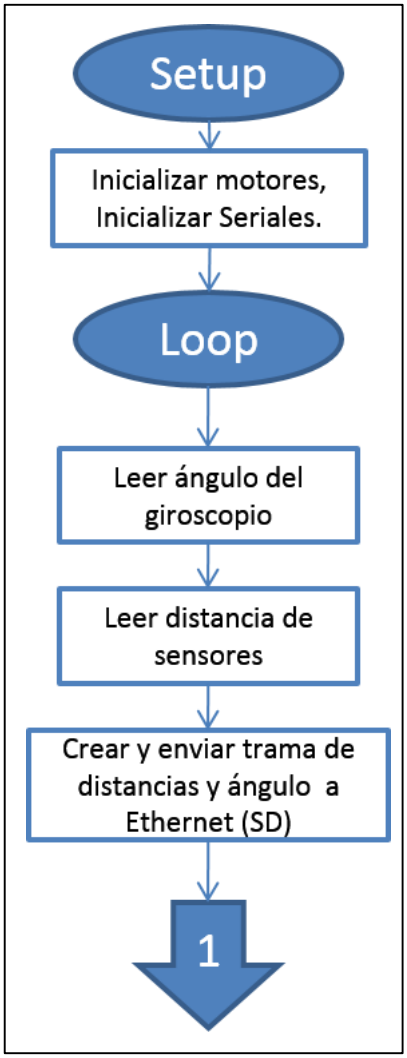


Figura 4.4 Esquema de la función setup() para el control de motores y sensores de distancia.

En la función inicial se activan las comunicaciones Serial1 y Serial a la velocidad de 115 200 baudios. También se inician las funciones de la librería para el *shield* de motores (Figura 4.4).

4.2.1.2. Función Loop

La primera acción es llamar a las subrutinas creadas para la adquisición de datos de los sensores ultrasónicos y los datos de orientación adquiridos con el giroscopio mediante su placa de control. Con esta información se construye una trama de comunicación y se la envía a través del Serial1 al controlador de Ethernet.

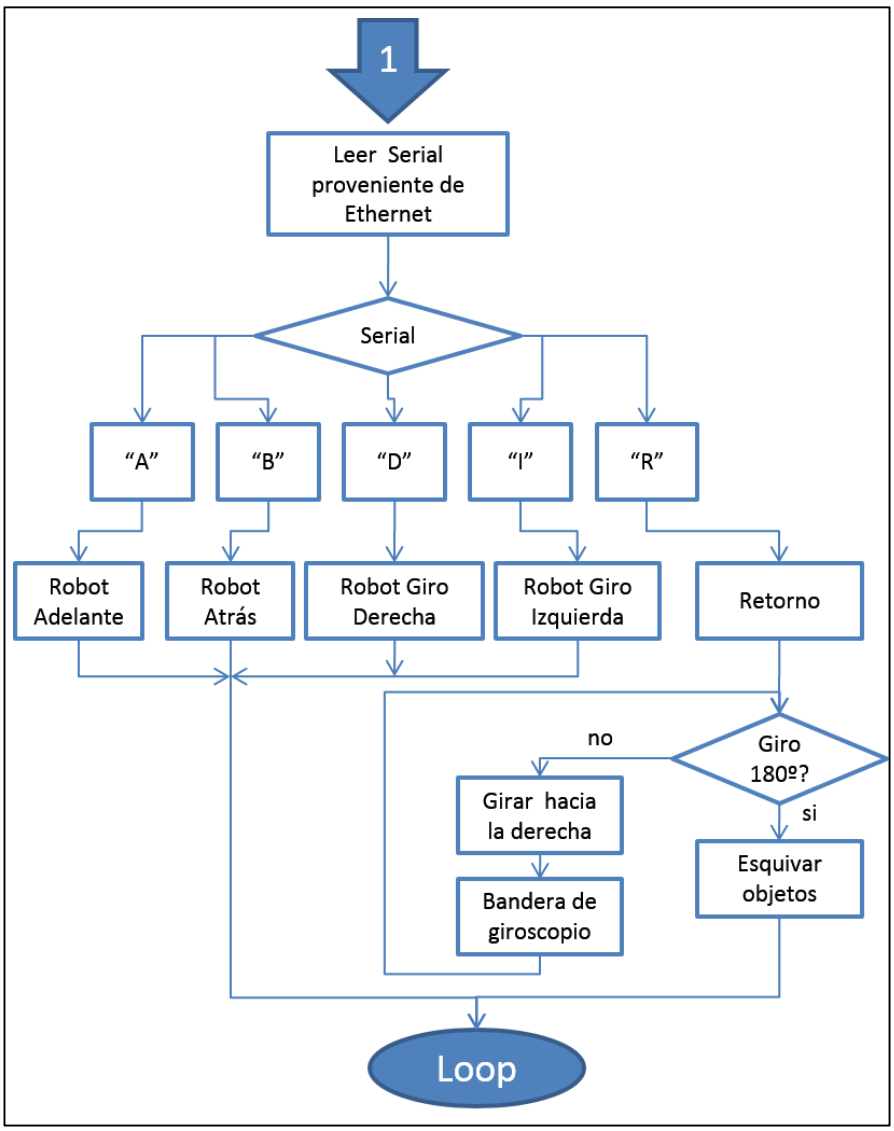


Figura 4.5 Función loop() para la placa controladora de motores y sensores de distancia.

A continuación, se llama a la subrutina de lectura de peticiones de la placa controladora de Ethernet a través del Serial1, en la cual se selecciona la acción del robot de acuerdo al comando enviado. Los comandos se muestran en la Tabla 4-2.

Comando	Movimiento
A	Motores hacia Adelante
B	Motores hacia Atrás
D	Motores hacia Derecha
I	Motores hacia Izquierda
R	Retorno

Tabla 4-2 Tabla de comandos para acciones del robot.

El comando Retorno se activa cuando se comprueba la pérdida de la comunicación con el equipo remoto. En este caso es necesario que el robot regrese sin la intervención del equipo remoto hasta una posición anterior en donde se pueda establecer una conexión y el robot pueda ser recuperado. Para ello, el robot gira 180 grados guiado por el giroscopio. Si se ha realizado el giro, el robot empieza a navegar de regreso, tomando la información del medio a través de los sensores de distancia. Este estado se mantiene hasta que la placa controladora de Ethernet detecte la conexión del equipo remoto, el cual ya puede recuperar el robot mediante otros comandos.

En el anexo 6 se presenta el código MPIDE completo del sistema.

4.2.2. Programa de la placa para adquisición de datos de giroscopio

Al principio del *sketch* se incluyen: la librería de manejo de comunicaciones I2C I2Cdev.h para comunicación con el giroscopio, y la librería MPU6050_6Axis_MotionApps20.h para procesar los datos del giroscopio como ángulos de orientación. También se declaran las variables en donde se almacenarán los datos de la memoria FIFO enviados por el giroscopio. Por último, se establece el vector de interrupciones para atender la recepción de datos cada vez que el giroscopio lo solicite.

4.2.2.1. Función Setup

En esta función se inicia la comunicación I2C con el giroscopio y se envían los comandos para comprobar su estado y presencia en el puerto a través de subrutinas de la librería. Luego, se inicia el puerto Serial a 115 200 baudios para la comunicación con la placa de control de motores. A continuación se activa la interrupción correspondiente al giroscopio y se la relaciona con el vector de interrupción antes creado. Si el giroscopio se inicia adecuadamente y empieza a enviar datos se procede a la función loop(), caso contrario se genera un mensaje de error.

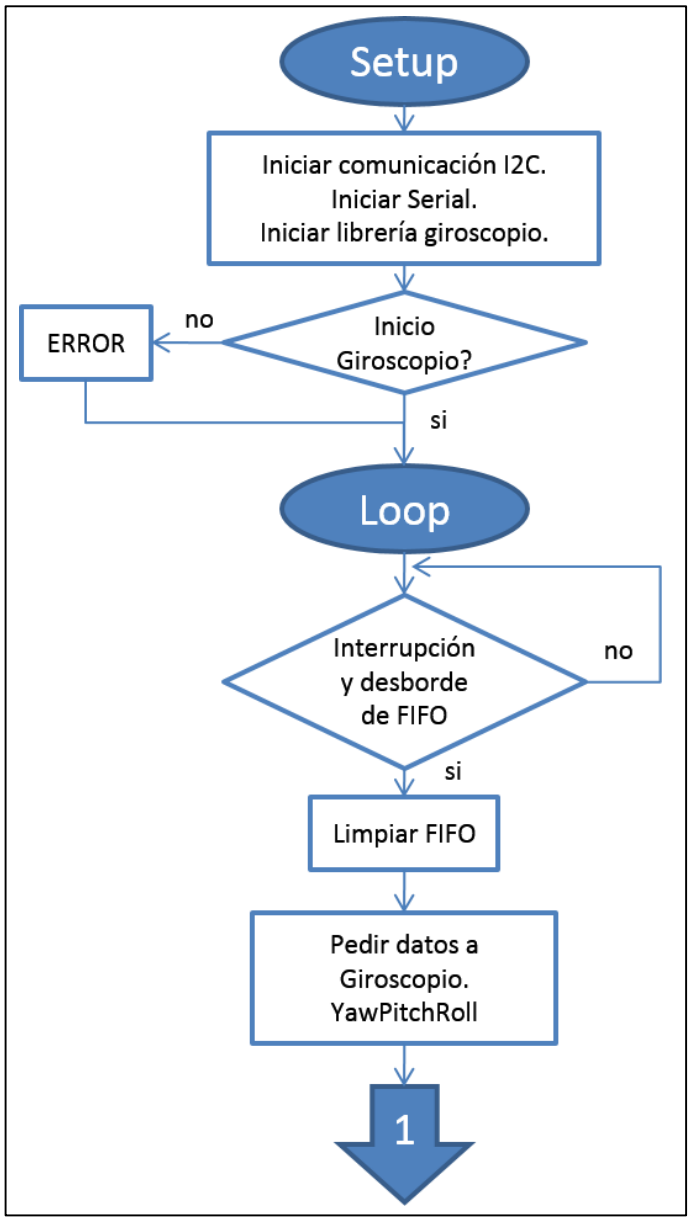


Figura 4.6 Función *setup* y parte de función *loop* de placa de control de giroscopio.

4.2.2.2. Función Loop

Esta función espera por una interrupción del giroscopio o un desborde de la memoria FIFO. Cuando ocurre uno de los dos eventos, se procesa la interrupción, se limpia la memoria que contiene los datos de le pila FIFO si esta se ha desbordado, y se obtiene la información de la orientación de los ángulos en 3 ejes, de los cuales se utilizará solo la orientación correspondiente a la dirección de movimiento del robot.

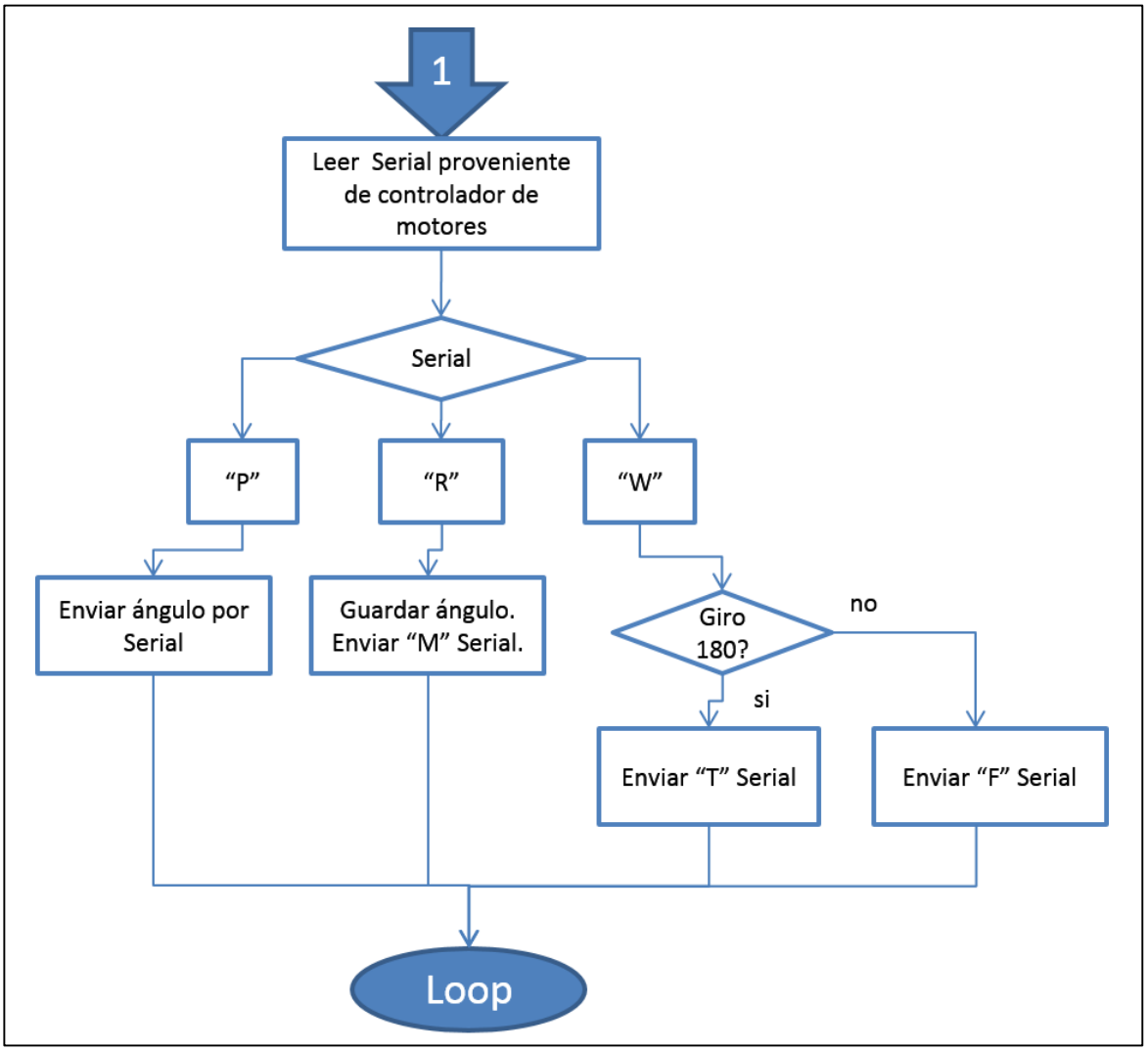


Figura 4.7 Función *loop* (continuación) de placa controladora de giroscopio.

Con el ángulo de orientación establecido, se espera la trama enviada por la placa controladora de motores a través del puerto Serial. La trama puede indicar tres acciones: un pedido de información (ángulo de orientación del robot), almacenamiento del ángulo actual para comparación con futuros movimientos en el modo autónomo (realización del giro de 180 grados), y confirmación del giro de 180 grados para activación del modo autónomo y retorno del robot.

En el anexo 7 se presenta el código completo sobre Arduino para la implementación de la comunicación con el giroscopio.

4.2.3. Programa de la placa para comunicación Ethernet

Antes de la declaración de variables y el código de programa se incluyen las librerías necesarias mediante las siguientes directivas:

```
#include <SD.h>
#include <SPI.h>
#include <Ethernet.h>
#include <dht11.h>
```

Como se explicó anteriormente, el chip Wiznet W5100 y la memoria flash microSD comparten el mismo bus SPI dentro del Ethernet *shield*, es decir, el microcontrolador de la placa principal (Arduino Mega 2560) se puede comunicar solo con uno de los dos a la vez. Esta comunicación, en su protocolo básico, es controlada mediante la librería SPI.

La librería SD permite manipular los archivos de la memoria microSD presente en la ranura, es decir que se usan subrutinas incluidas en la librería que facilitan el manejo de archivos, enviando los comandos específicos de control para una memoria a través del bus SPI. Las acciones principales que se llevan a cabo son la creación de archivos de texto, los cuales son abiertos y modificados para incluir nuevos datos y llevar registros de gran cantidad de variables a una gran velocidad de muestreo.

De igual manera, la librería Ethernet permite utilizar subrutinas creadas específicamente para enviar comandos de comunicación y configuración específicos al chip Wiznet W5100, como el uso de protocolos (TCP o UDP), la especificación de direcciones IP y el puerto a usar.

La librería DTH11 contiene la implementación a bajo nivel del protocolo *1-Wire* para la comunicación con el sensor de humedad y temperatura. Se debe mencionar que esta es una implementación de software, es decir, que el microcontrolador no posee un periférico para este protocolo de comunicaciones, sino que sus parámetros han sido aplicados mediante código.

4.2.3.1. Función Setup

En esta función se define el pin 9 de la placa para comunicación con el sensor de humedad y temperatura. La librería DTH11 se encarga internamente de intercambiar este pin de entrada a salida y viceversa para el envío y recepción de datos.

A continuación, se activan las comunicaciones Serial1 a 9 600 baudios (comunicación con el controlador de la pantalla), y Serial2 a 115 200 (comunicación con el controlador de motores). El puerto Serial a USB se activa a 9 600 baudios con el objetivo de enviar datos relacionados con errores de cualquiera de los dispositivos, y se usa solo en estados de prueba o mantenimiento.

En seguida, se realiza la activación del bus SPI comprobando la presencia de la memoria microSD mediante el pin 4 de la placa principal o CS (*Chip Select*) correspondiente a la memoria. De igual manera se inicia la comunicación con el chip de Ethernet indicado la dirección física (MAC), IP, puerta de enlace y máscara de red.

Por último, se inicia el servidor Ethernet, el cual estará listo para aceptar hasta 4 conexiones o peticiones al mismo tiempo. De igual manera, si en cualquier parte del proceso se presenta algún error, este será reportado a través del puerto Serial a USB. El diagrama de bloques simplificado de esta función se encuentra en la Figura 4.8.

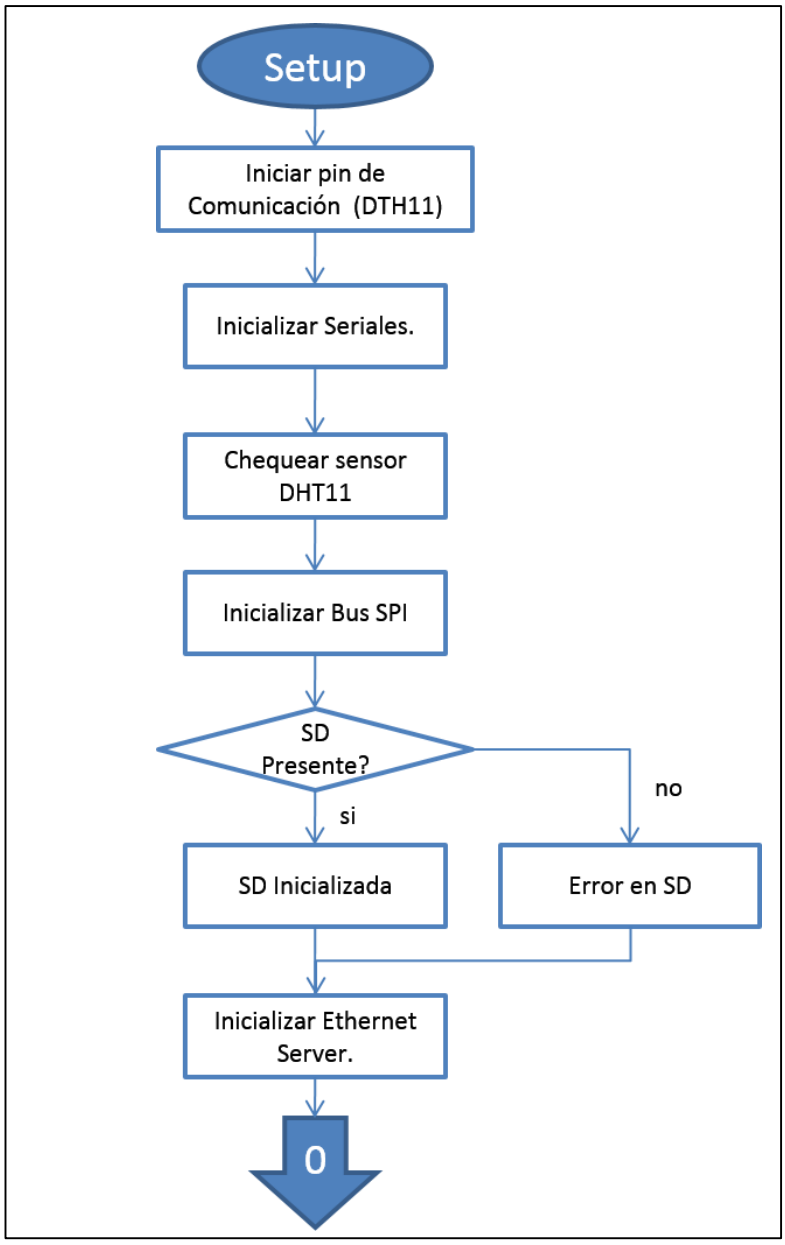


Figura 4.8 Secuencia de configuración (*setup)* para controlador de Ethernet.

4.2.3.2. Función Loop

En esta función se recogen los datos provenientes del Serial2 conectado a la placa de control de motores. La trama recibida se concatena y almacena hasta la llegada de un carácter de fin de trama.

A continuación, se toman los datos de los sensores de gas (lectura directa del convertidor analógico-digital en el pin A1), humedad y temperatura, y se crea una trama de datos que es enviada al controlador de la pantalla a petición por el Serial1.

La trama completa se crea uniendo la primera trama, que contiene la información de los sensores de distancia y de orientación, y la segunda trama, formada por las variables ambientales. Esta trama tiene la siguiente configuración:

Axxx**B**xxx**C**xxx**D**xxx**E**xxx**J**xxx**X**xxx**H**xxx**T**xxx**R**xxx**G**xxx**F**

Las letras mayúsculas permiten diferenciar los datos recibidos. Desde A hasta J se encuentran las distancias obtenidas por los sensores ultrasónicos; después de X está la orientación dada por el giroscopio, después de H está el porcentaje de humedad, de T la temperatura, de R el estado del batería y de G la presencia de gases inflamables. La F indica fin de transmisión de trama.

Con la trama completa se abre el archivo DATALOG.txt contenido en la memoria microSD utilizando las subrutinas de la librería SD. Si el archivo no existe se procede a crearlo, o simplemente se escriben los datos de la trama creada anteriormente. Si el proceso de escritura no es exitoso, se crea una entrada de error en el archivo y se informa por el puerto Serial a USB (Figura 4.9).

A continuación se procede a atender las comunicaciones con el servidor Ethernet iniciado anteriormente. Para ello se consulta si existe alguna petición de un cliente, y se comprueba el comando enviado a través de métodos GET y POST de HTTP (*Hypertext Transfer Protocol*). Este protocolo realiza una petición al servidor indicando su dirección IP, el puerto TCP a escuchar, y el comando o palabra clave que solicita (http://ipdelservidor:puerto?comando$). Los comandos específicos creados para las funciones del robot se muestran en la Tabla 4-3.

Direcciones URL para Robot	
http://10.0.1.177:80?T$	Enviar Trama
http://10.0.1.177:80?A$	Enviar comando Adelante
http://10.0.1.177:80?B$	Enviar comando Atrás
http://10.0.1.177:80?D$	Enviar comando Giro Derecha
http://10.0.1.177:80?I$	Enviar comando Giro Izquierda

Tabla 4-3 Lista de comandos a través de métodos de HTTP.

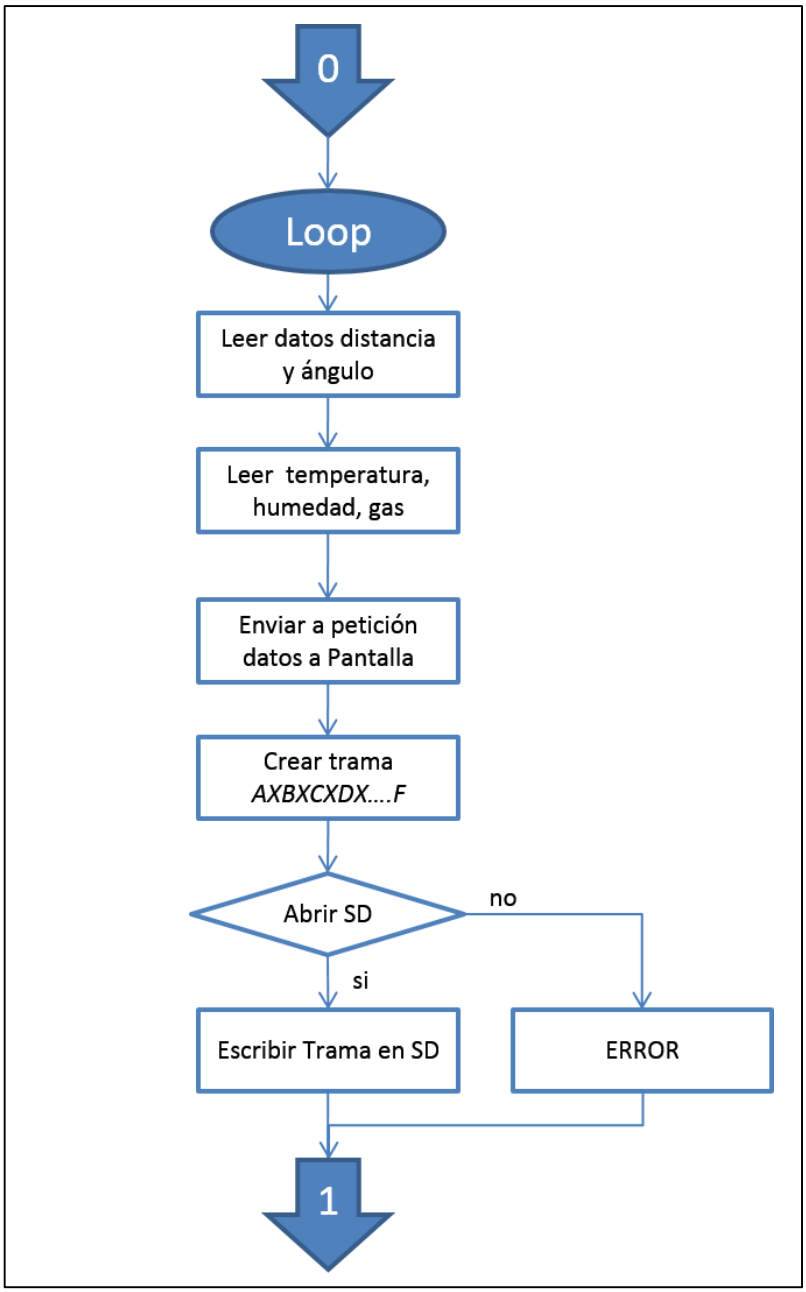


Figura 4.9 Esquema de función *loop* del controlador Ethernet (parte 1).

En caso de no existir conexión con algún cliente, se lleva un conteo de las veces en las que la conexión ha sido fallida. Esto permite comprobar el estado de la conexión con el equipo remoto, y, en caso de haberse perdido contacto, activar la secuencia autónoma de regreso del robot (Figura 4.10).

Si el conteo de conexiones fallidas es igual a 90 se envía el comando de activación de retorno a través del Serial2 hacia la placa controladora de motores. La cantidad de intentos de conexión permite dar un espacio de tiempo considerable para que la señal sea restablecida, puesto que en muchos casos pueden existir pérdidas en la conexión esporádicas (intermitencias) que no implican fallo en la conexión.

En el anexo 8 se presenta el código completo sobre Arduino.

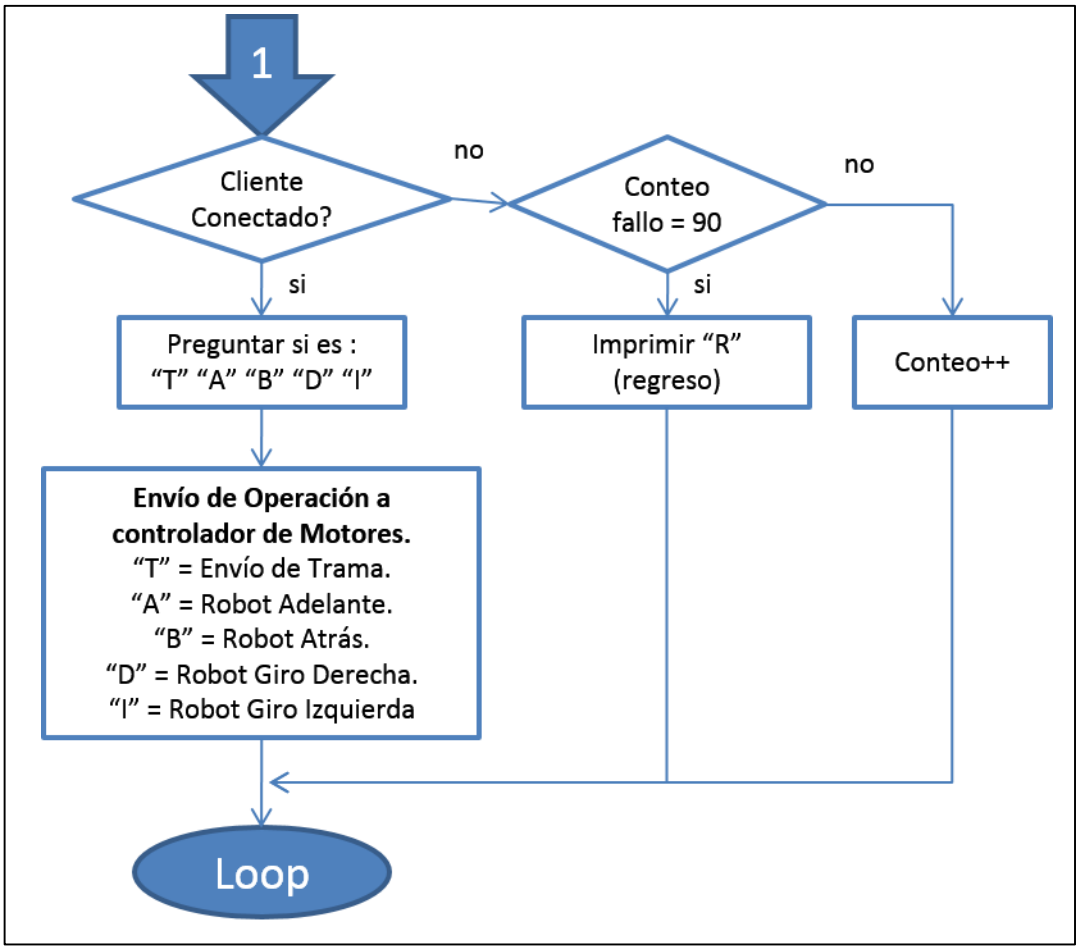


Figura 4.10 Esquema de función *loop* del controlador Ethernet (parte 2).

4.2.4. Programa de la placa controladora de pantalla LCD

Al inicio de la programación se incluyen las librerías UTFT.h (para la visualización de las imágenes en el LCD a partir de matrices) y Utouch.h (para el manejo del controlador de la superficie táctil). También se declaran las variables de tipo matricial para el manejo de texto e imágenes.

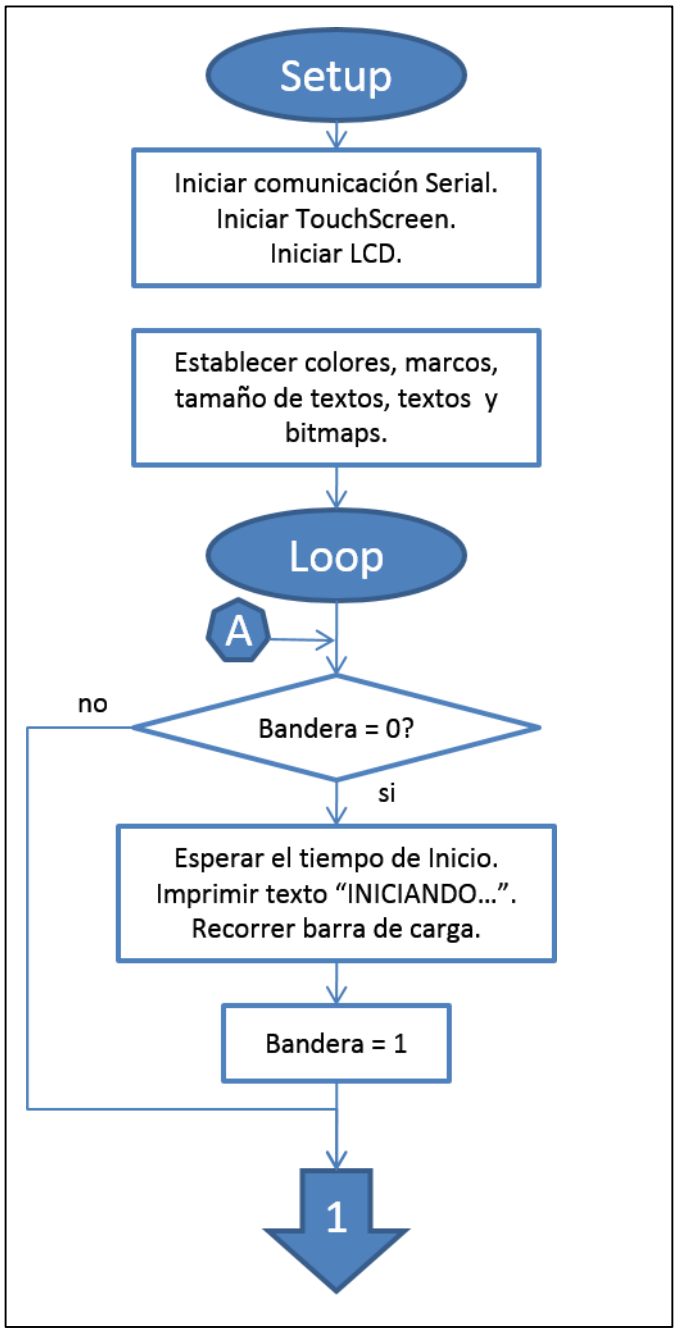


Figura 4.11 Función *setup* y parte de la función *loop* para control de pantalla LCD táctil.

4.2.4.1. Función Setup

En esta función se inicializa la librería para la detección de áreas presionadas en la pantalla. También se inicia el LCD y se establecen los colores, cuadros y los logos creados en formato de mapa de bits. Por último, se activa la comunicación en el puerto Serial a 9 600 baudios para la petición de datos de los sensores ambientales.

4.2.4.2. Función Loop

La pantalla inicia mostrando una secuencia de introducción que genera un intervalo de tiempo para que el resto de dispositivos puedan iniciarse, especialmente al punto de acceso inalámbrico. El logo de la Universidad del Azuay también es visualizado, el cual se ha construido como un arreglo de pixeles y guardado en la memoria del controlador (Figura 4.12).

Cuando ha culminado el tiempo de iniciación, la tarjeta controladora de Ethernet envía una confirmación (bandera) y se inician las peticiones periódicas a través del puerto de comunicación Serial conectado con la placa controladora de Ethernet. Los datos solicitados son los de temperatura, humedad y presencia de gases, los cuales son visualizados en cuadros separados en la pantalla (Figura 4.13).

Figura 4.12 Pantalla inicial con logo de la Universidad del Azuay.

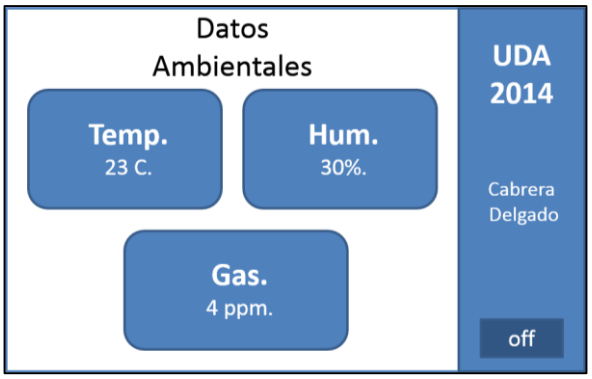


Figura 4.13 Diseño de la pantalla para visualización de variables ambientales.

Para evitar el consumo excesivo de energía por parte de la pantalla, ésta posee un botón que permite apagarla parcialmente y no mostrar los datos ambientales. Cuando la pantalla se encuentra apagada los datos se siguen enviando al equipo remoto y almacenando en la memoria microSD, pero el controlador de pantalla no hará peticiones, mejorando la velocidad del sistema.

En el anexo 9 se presenta el código Arduino, y el manual de uso de la pantalla UTFT 3,2” que suministró el proveedor.

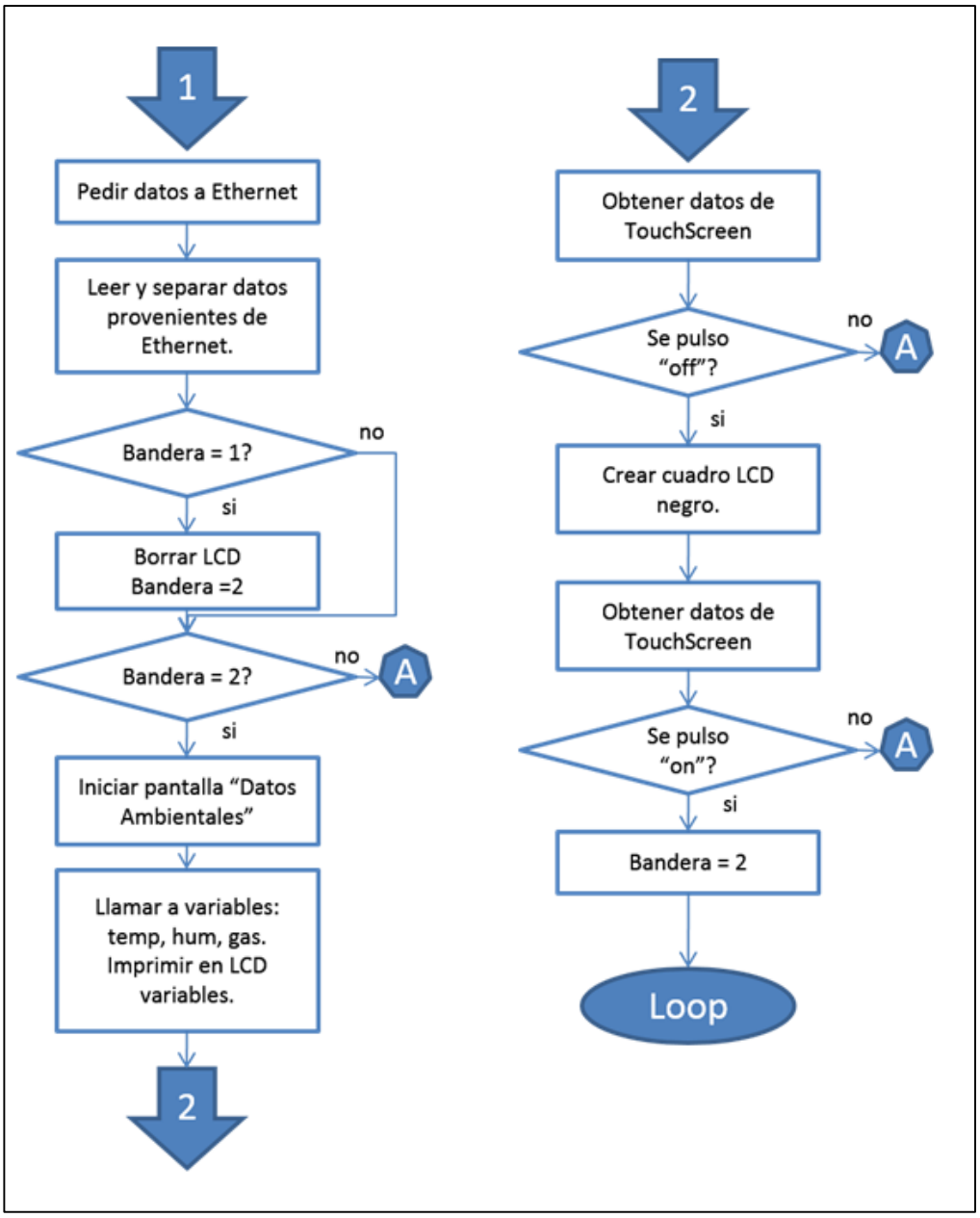


Figura 4.14 Función *loop* (continuación) para control de pantalla LCD táctil.

4.3. Configuración del punto de acceso inalámbrico

El punto de acceso inalámbrico permite la personalización y configuración de varios parámetros a través de un menú web. Para ello es necesario conectarse a uno de sus puertos LAN mediante un cable, y su configuración de fábrica asignará una dirección IP al computador. La dirección por defecto es 192.168.10.1, en la cual está el menú de configuración.

Una vez dentro del menú de configuración, se establecen sus funciones como Servidor DHCP (*Dynamic Host Configuration Protocol*), es decir, que asignará una dirección IP automáticamente a cualquier cliente conectado. Su SSID (*Service Set Identifier* o nombre de red inalámbrico) se establece como "Robot", y se cambia su dirección base a 10.0.1.25 (clase A, privada, máscara de red 255.255.255.0). Es importante que todos los elementos de la red estén dentro del rango de direcciones IP de la red del punto de acceso para poder comunicarse. Así, la dirección del servidor de la placa controladora de Ethernet se ha establecido en 10.0.1.177 y la de la cámara en 10.0.1.150.

Por último se introduce una contraseña para el acceso inalámbrico y se coloca la banda de radiofrecuencia en 2,4 GHz.

Figura 4.15 Menú de configuración del punto de acceso inalámbrico.

4.4. Configuración de la cámara IP

Para la configuración inicial de la cámara, es necesario conectarla a un computador a través de un cable Ethernet, y entrar en su menú de configuración que se encuentra en la dirección establecida de fábrica 192.168.10.1:8080.

En el menú se cambia la dirección de la cámara a 10.0.1.150 y se cambia el puerto de escucha a 9080. Se establece su resolución en 640 x 480 a 30 cuadros por segundo. Por último, se desactiva la transmisión de imágenes vía inalámbrica, puesto que la cámara está conectada a uno de los puertos LAN del punto de acceso inalámbrico, permitiendo disminuir el consumo de energía.

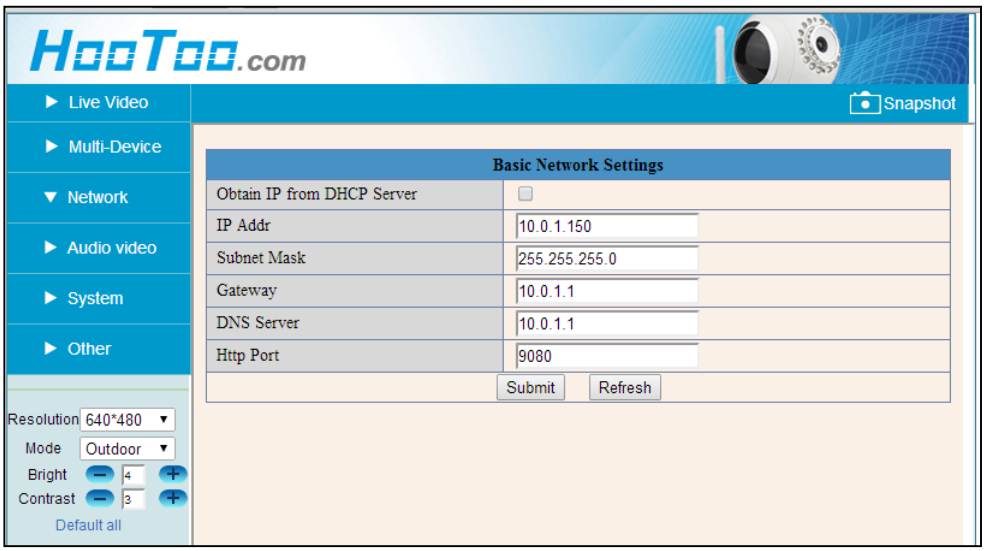


Figura 4.16 Menú de configuración de la cámara IP.

CAPÍTULO 5

PROGRAMACION DEL EQUIPO REMOTO

El programa del computador (equipo remoto) permitirá la comunicación con el robot para obtener su estado (coordenadas y orientación) y la información recopilada por sus sensores. Esta información será almacenada y procesada para determinar el mapa de la mina, temperatura, humedad, presencia de gases peligrosos, e imágenes que sean útiles para establecer su seguridad.

5.1. Plataforma LabVIEW

La programación del computador está realizada en LabVIEW (*Laboratory Virtual Instrumentation Engineering Workbench*), una plataforma de programación gráfica de la empresa National Instruments. Los programas creados en LabVIEW se llaman instrumentos virtuales o "VI", denominados así porque en su origen el objetivo de la programación en esta plataforma era crear sistemas de medición electrónica, especialmente para simulación. En la actualidad, además de realizar sistemas de medición e instrumentación electrónica, los programas creados en LabVIEW se utilizan sobre sistemas embebidos, comunicación y automatización industrial en general, especialmente en conjunto con hardware propio de National Instruments o de otros fabricantes. Ejemplos de ello son laboratorios de hardware y software para telecomunicaciones en todas las frecuencias, sistemas de computadores en tiempo real, visión artificial, adquisición de datos, programación para sistemas FPGA, etc.

El lenguaje que usa se conoce como lenguaje G, que proviene de su naturaleza gráfica, el cual se caracteriza por el uso de estructuras, bloques y diagramas en los que el programador conecta nodos, entradas, salidas y controla el orden de la ejecución dibujando cables. Cada programa o VI posee tres partes diferenciadas: diagrama de bloques, panel frontal y panel de conexión (10).

El diagrama de bloques contiene el código fuente en forma gráfica en el que el programador realiza la lógica de ejecución, y coloca entradas, funciones y salidas.

El panel frontal contiene las entradas y salidas que se han colocado en el diagrama de bloques, los cuales pueden ser modificados para mostrarse de varias maneras (gráficas, termómetros, tanques, texto, luces, etc.)

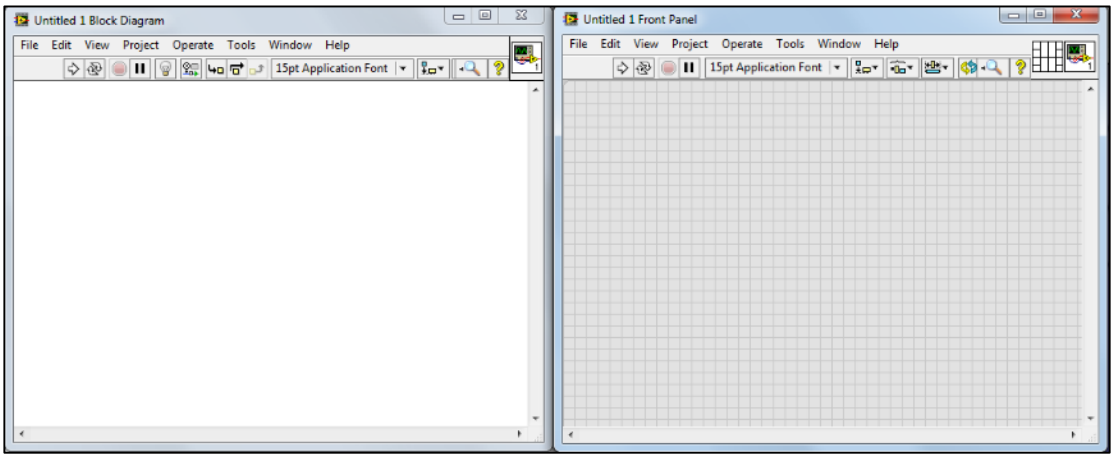


Figura 5.1 Ventana del diagrama de bloques (izquierda) y ventana del panel frontal en LabVIEW.

El panel de conexión es la representación gráfica de un VI para poder ser referido por otros, en conjunto con sus entradas y salidas. Se podría equiparar con el nombre para poder llamar a una subrutina, con los valores de ingreso y de retorno de la misma (parámetros).

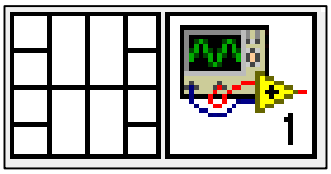


Figura 5.2 Panel de conexión. En la izquierda se colocan las entradas y en la derecha, las salidas.

En LabVIEW, los elementos programáticos se encuentran en los menús, tanto del panel frontal como del diagrama de bloques. Estos se muestran al hacer clic derecho sobre una de las dos ventanas.

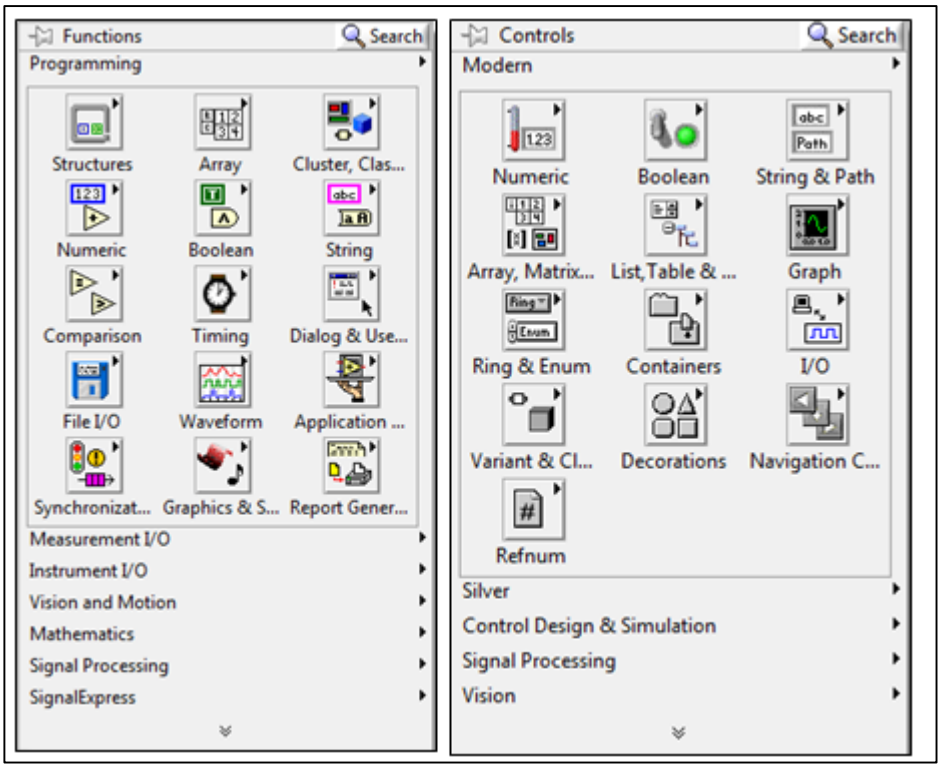


Figura 5.3 Menú de diagrama de bloques (izquierda) y del panel frontal (derecha).

Los elementos colocados como entradas se denominan controles, las salidas se conocen como indicadores, y ambos tendrán una correspondencia en el panel frontal. Los elementos se conectan mediante líneas a las cuales se conoce como cables.

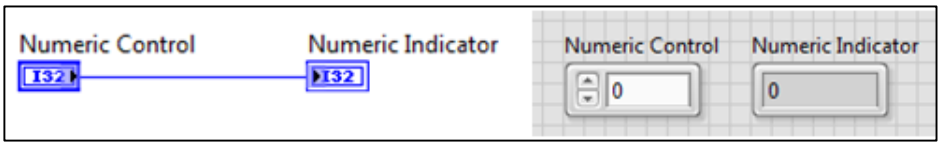


Figura 5.4 Control e indicador en el diagrama de bloques y en el panel frontal.

Para ejecutar los programas se utilizan los controles de la barra de herramientas de Operación. Estos son *Run* (ejecuta 1 sola vez el programa), *Run Continuously* (ejecuta continuamente el programa), *Abort* (finaliza el programa) y *Pause* (detiene el programa en la instrucción actual).

Figura 5.5 Barra de herramientas de control de operación.

5.1.1. Estructuras de control de programa

LabVIEW es un lenguaje compilado, es decir que el sistema en segundo plano del programa crea una tabla con las conexiones de cables, variables y estructuras de control, y los compila a un código ejecutable cada vez que el programador realiza un cambio. Si la compilación no es exitosa, implicando que el código gráfico no está completo en alguna de sus conexiones, el programa no se puede ejecutar y el ícono de *Run* se encuentra roto.

El orden de ejecución del código no está definido en todos los casos; el sistema del entorno de programación selecciona qué actividades realizar primero. Sin embargo, el programador puede controlar la secuencialidad del programa tomando en cuenta que ninguna función u operación se realiza si no tiene todas sus entradas disponibles. Esto hace que el flujo de la programación, en general, sea de izquierda a derecha. En los ordenadores que poseen varios núcleos e hilos de procesamiento, LabVIEW ejecuta el código en paralelo ocupando los núcleos e hilos de acuerdo a su disponibilidad (14).

5.1.1.1. Bucle For

Figura 5.6 Gráfico que representa bucle *For*.

Ejecuta los subdiagramas contenidos las veces que se indica en el terminal N. El terminal *i* provee la iteración actual, la cual se ejecuta desde 0 hasta n-1.

5.1.1.2. Bucle While

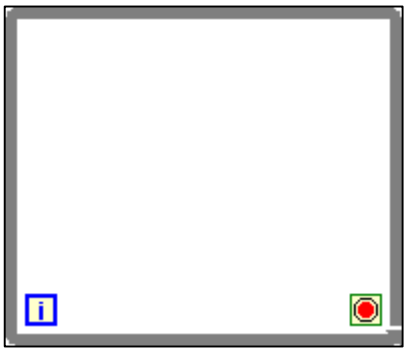


Figura 5.7 Gráfico que representa la función *While*.

Esta estructura repite los subdiagramas que se encuentren en su interior hasta que la condición de su terminal booleano se cumpla. Esta condición puede ser *Stop if True* (detener si es verdadero) o *Continue if True* (continuar si es verdadero). Similar al bucle *For*, el terminal *i* contiene la iteración actual que se ejecuta.

5.1.1.3. Estructura selectora Case Structure

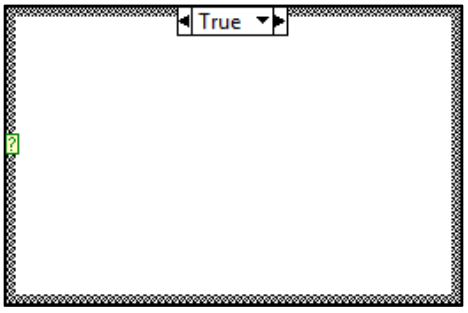


Figura 5.8 Gráfico que representa la estructura para selección de casos.

Puede contener uno o más subdiagramas o casos, los cuales son ejecutados excluyendo a los otros, de acuerdo al valor cableado al terminal selector. Por defecto, los valores a ingresar son booleanos y los casos son *True* y *False*; sin embargo, se puede colocar cualquier tipo de valor en el terminal y los casos pueden ser varios y de cualquier tipo.

5.1.1.4. Estructura Flat Sequence

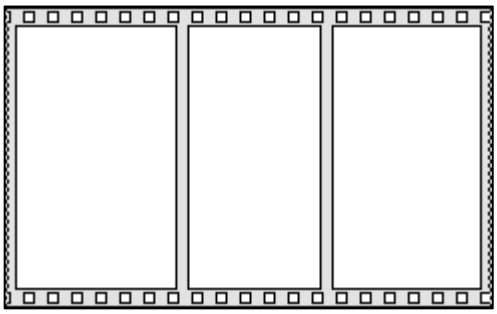


Figura 5.9 Estructura para la ejecución secuencial.

Consiste en uno o más subdiagramas que se ejecutan en secuencia. Esta estructura es útil para obligar al programa a que siga un orden de instrucciones, de izquierda a derecha, y que mantenga una actualización de las variables en cada cuadro.

5.1.2. Tipos de datos

En el entorno del programa existen varios tipos de datos. Los tipos básicos son los de tipo booleano, numérico y cadena de caracteres. Los de tipo booleano solo toman los dos estados digitales.

Los datos de tipo numérico pueden ser de distintos tipos de acuerdo a la capacidad en memoria que ocupan. Se distinguen los datos de tipo entero (con y sin signo, de 8, 16, 32 y 64 bits) y los con punto flotante (de simple o doble precisión). También existen los valores de tipo complejo (real e imaginario).

Los datos de tipo *string* o texto incluyen caracteres individuales o cadenas de caracteres. Están representados en color rosa dentro del diagrama de bloques para su fácil reconocimiento.

Los arreglos de datos pueden ser de cualquier tamaño (n dimensiones) y están formados por elementos del mismo tipo. Los elementos dentro de los arreglos son referidos mediante n subíndices que indican su posición. Los clústeres son similares a los arreglos, pero pueden contener distintos tipos de elementos, los cuales son referidos mediante sus nombres.

5.1.3. Funciones básicas

A continuación se describirán las funciones utilizadas dentro del programa de control y monitoreo del robot.

5.1.3.1. Funciones matemáticas

Las funciones matemáticas son exclusivas para números y permiten realizar todas las operaciones básicas y algunas operaciones especiales como redondeos, incrementos rápidos, etc.

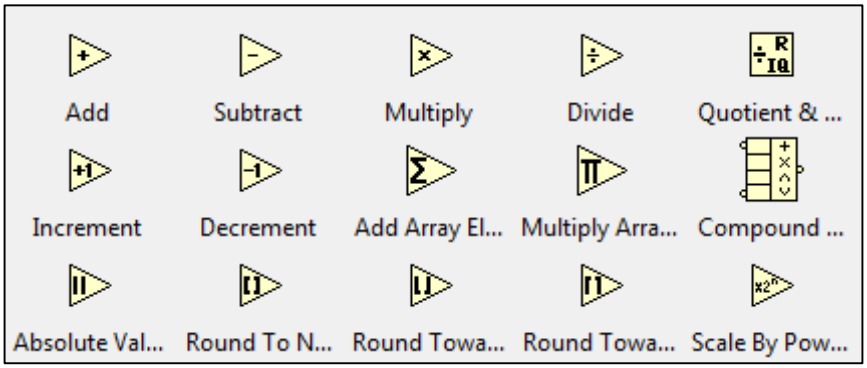


Figura 5.10 Menú de funciones numéricas dentro de LabVIEW.

5.1.3.2. Funciones booleanas y comparaciones

Las operaciones de lógica booleana se encuentran listadas en el menú de la Figura 5.11**.** Estas solo pueden ser utilizadas entre variables booleanas.

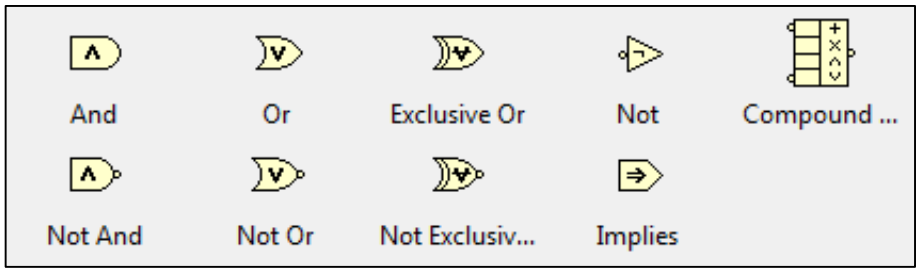


Figura 5.11 Menú de funciones booleanas (detalle).

Las funciones de comparación pueden ser utilizadas con datos de cualquier tipo, y su salida será un valor booleano.

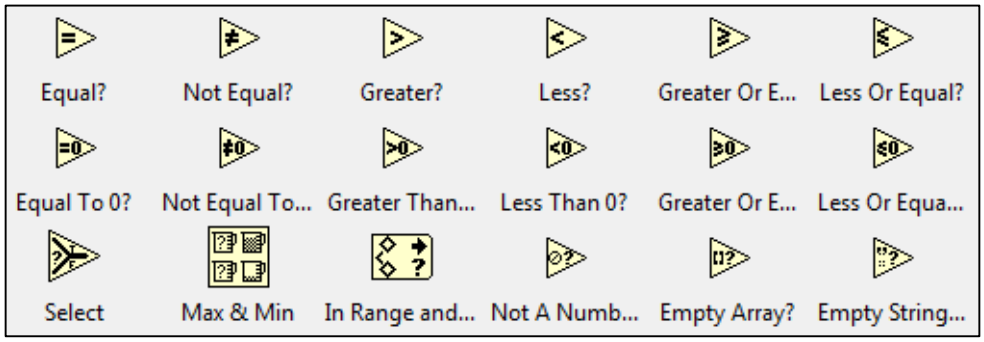


Figura 5.12 Menú de funciones de comparación (detalle)

La función *Select* de este menú devuelve el valor cableado en su terminal *true* o *false*, de acuerdo a la condición de ingreso. Es una función muy utilizada para control de flujo de valores numéricos y de caracteres.

5.1.3.3. Funciones de manejo de cadenas de caracteres

Las funciones para el control de cadenas permiten realizar comparaciones más complejas dentro de una sucesión, por ejemplo, buscar patrones comunes como palabras clave, formatos de fecha u hora. También permiten realizan inserciones y reemplazos.

Los valores de tipo cadena pueden ser transformados a sus correspondientes valores de tipo numérico y viceversa (conversión ASCII). Esto es útil cuando se utilizan comunicaciones a bajo nivel en donde el tipo de dato no está especificado.

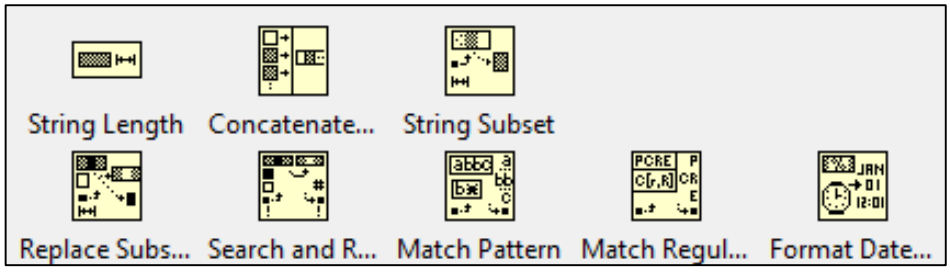


Figura 5.13 Menú de funciones para manejo de cadenas (detalle).

5.1.3.4. Funciones de manejo de arreglos

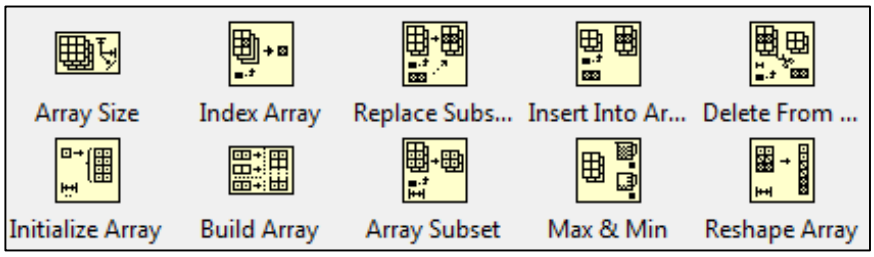


Figura 5.14 Menú de funciones para manejo de arreglos (detalle).

Las funciones para el manejo de arreglos permiten construir, modificar, eliminar y extraer sub arreglos de un arreglo principal. También existen funciones para analizar los datos dentro del mismo, como máximos, mínimos, ordenamiento y transformación.

5.1.4. Funciones de comunicación HTTP

LabVIEW permite las comunicaciones mediante varios protocolos. Uno de ellos es a través del protocolo HTTP (*Hypertext Transfer Protocol*), el cual es el más usado en las transacciones en internet. Está basado en un sistema cliente-servidor, en donde el cliente realiza peticiones al servidor y espera su respuesta.

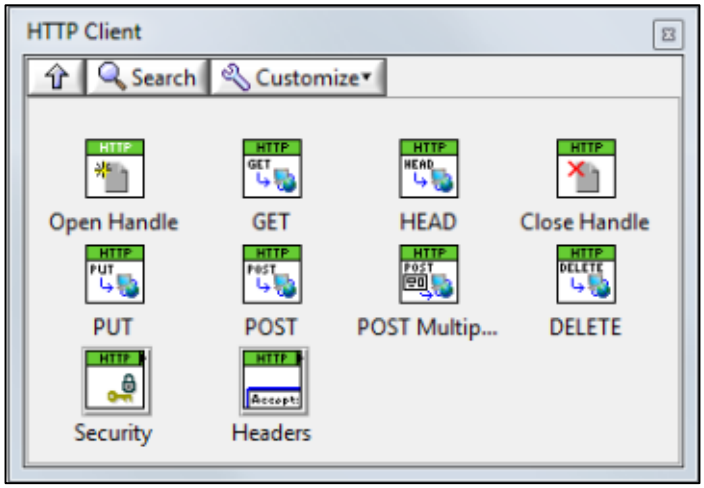


Figura 5.15 Menú de funciones con métodos de HTTP.

La transacción de HTTP se inicia con una petición por parte del cliente, el cual indica la URL (*Uniform Resource Locator*) del recurso que desea, pudiendo ser este un conjunto de datos de tipo texto o multimedia cualquiera que el servidor posea. El servidor envía un encabezado, una línea en blanco y, al final, el recurso requerido si este existe; si no existe, envía un mensaje de error. El formato general de un URL a través de HTTP es el siguiente: http://IP-servidor:puerto/directorio/recurso. Las peticiones pueden realizarse utilizando distintos métodos o verbos, siendo los principales GET y POST.

GET pide el recurso enviando información a través de la misma URL. Este será utilizado por el cliente para pedir datos al servidor dentro del robot. En este caso se utilizará la URL http://10.0.1.177/?T$, que es la petición de la trama de datos completa captada por el robot como se indica en la Tabla 4-3.

El método POST envía datos para que sean procesados por el servidor, actualizando sus archivos, creando nuevas entradas o realizando alguna acción. Las acciones del robot controladas por el equipo remoto serán enviadas a través de este método con la URL http://10.0.1.177/?X$, en donde X indica la acción de acuerdo a la Tabla 4-3.

5.1.5. Función de navegador web

La función de navegador web se encuentra en el menú .NET & ActiveX del panel frontal de LabVIEW. Esta herramienta permite insertar una instancia del navegador web del sistema, la cual permitirá la visualización de la cámara IP del robot.

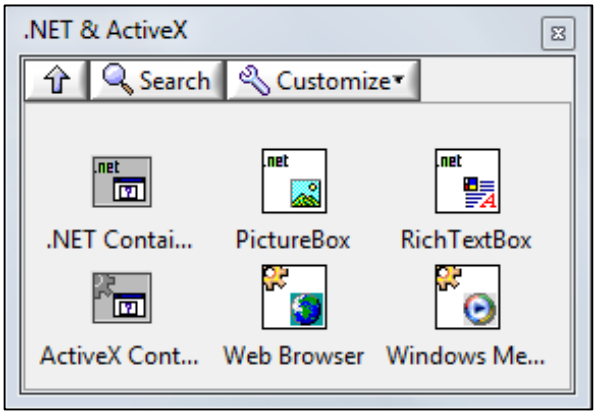


Figura 5.16 Menú .NET ActiveX del panel de control.

La dirección IP de la cámara debe ser especificada en el diagrama de bloques, y debe ser la misma que se configuró inicialmente (http://10.0.1.150:9080).

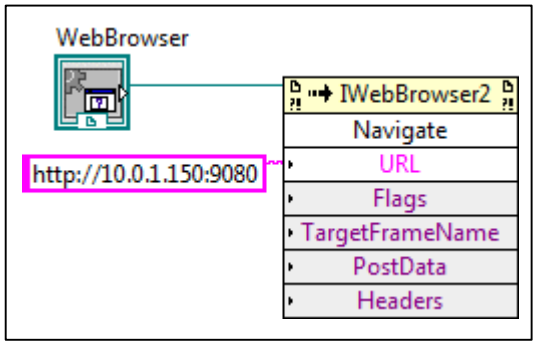


Figura 5.17 Control de navegador web configurado con la URL de la cámara IP.

5.1.6. Indicador Intensity Graph

El indicador *Intensity Graph* (Gráfico de Intensidad) permite graficar datos de 3 dimensiones en un esquema de dos dimensiones. A la posición X e Y se le aumenta la escala de colores que se le puede añadir a cada pixel o punto de la matriz. Este indicador es particularmente útil para dibujar mapas en 2 dimensiones en los que se puede añadir capas de colores según intensidad que indiquen cualquier otra variable (profundidad, densidad, temperatura, etc.). Se debe tomar en cuenta que las propiedades de cada elemento de la gráfica están especificadas por su posición X (columna), Y (fila) y su valor de 0 a 255, los cuales representan un color. Por lo tanto, las propiedades de los pixeles son discretas, enteras y positivas.

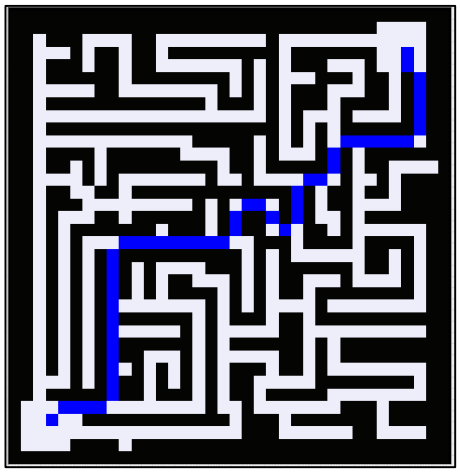


Figura 5.18 Ejemplo de un gráfico creado con un *Intensity Graph*.

5.2. Programa para control y adquisición de datos

A continuación se describe la programación utilizada para el control y adquisición de datos del robot, con el objetivo de presentar al operador las imágenes de la cámara en tiempo real, las mediciones de los sensores ambientales y la creación del mapa a partir de la información de los sensores de distancia.

El VI creado en LabVIEW consta de un cuadro de condiciones iniciales y dos hilos paralelos: un sistema de conexión HTTP que pide la trama de datos al servidor del robot, los muestra en indicadores y crea el mapa; y un sistema para enviar datos de control y mostrar el movimiento del robot en el mapa.

5.2.1. Condiciones iniciales

En la rutina inicial se crea el arreglo de igual número de filas y columnas con el color blanco (255). El tamaño del arreglo está predefinido en 900 pixeles de alto y ancho, pero puede ser modificado por el usuario al inicio del programa. En base al tamaño, se calcula la posición del pixel medio y se envía sus coordenadas al subVI “CAMINO ROBOT” para que dibuje su posición dentro del Gráfico de Intensidades “IGraph” (este se describirá más adelante).

Se inicia el control de navegador web y se indica el URL de la cámara IP como se indicó en el numeral 5.1.5. También se establece el ángulo inicial en 90 grados, y se abre una sesión como cliente HTTP. Aquí no se indica ninguna dirección IP del servidor, simplemente se prepara al sistema para la utilización de las herramientas de petición y envió de datos HTTP.

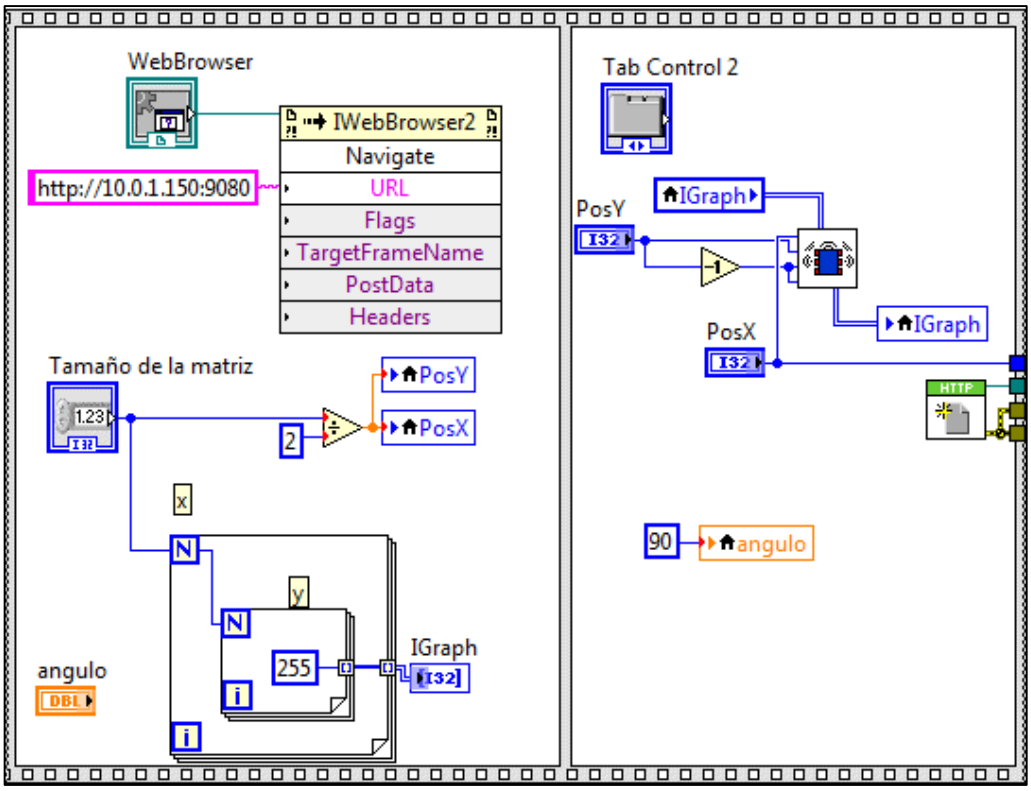


Figura 5.19 Secuencia de inicio dentro de una función *Flat Sequence.*

5.2.2. Hilo de adquisición de datos

Dentro de este proceso se realiza la petición de la trama de datos de los sensores al servidor al URL http://10.0.1.177/?T$. Esta acción se repite cada 150 milisegundos, actualizándose las variables en ese tiempo y mostrándose en los indicadores del panel frontal. Además, se realiza una comprobación de la conexión con el servidor sobre el robot y se muestra mediante un indicador en el panel frontal.

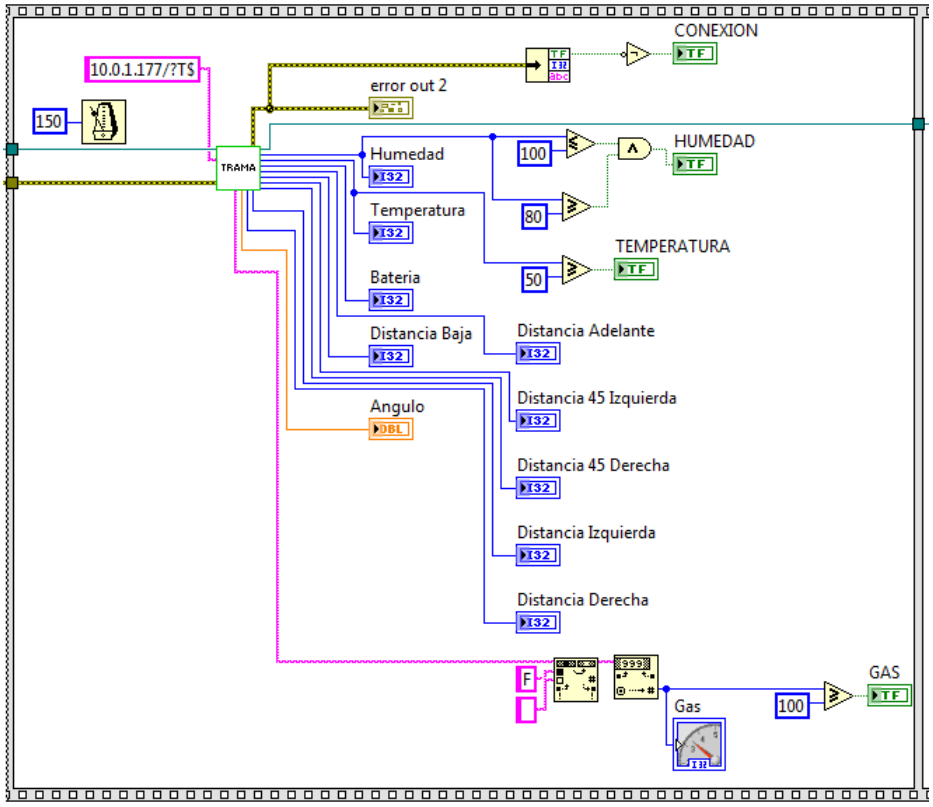


Figura 5.20 Adquisición de datos enviados por servidor dentro del robot.

El subVI "TRAMA" contiene la petición GET de HTTP y obtiene la trama de datos enviada. Esta trama es separada de acuerdo a su encabezado y enviada a variables de salida.

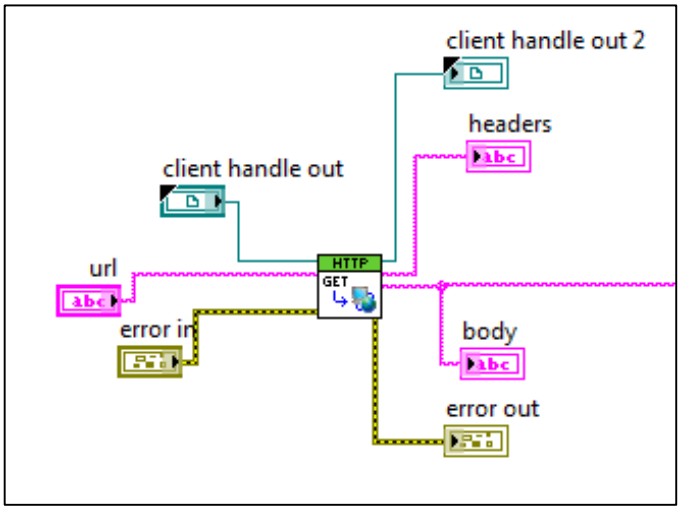


Figura 5.21 Adquisición de trama dentro del subVI "TRAMA".

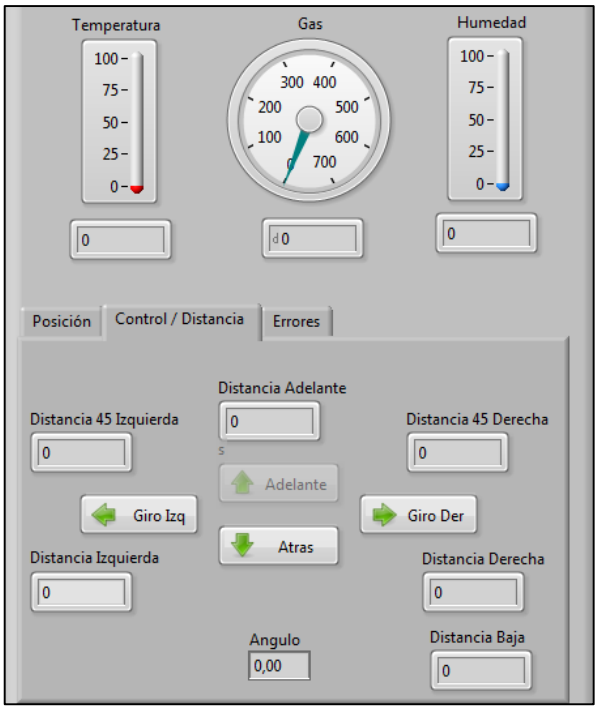


Figura 5.22 Panel frontal donde se muestran los datos de los sensores.

Por último, se envía los datos de los sensores de distancia al subVI "DISTANCIAS", el cual dibuja los rayos de acción de los 6 sensores en base a la ubicación y orientación actual del robot.

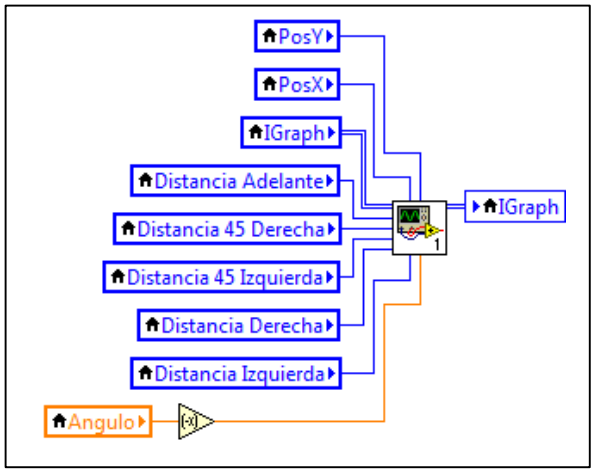


Figura 5.23 Llamada al subVI "DISTANCIAS" insertando las distancias adquiridas de los sensores.

5.2.2.1. Graficación de la acción de los sensores

El subVI "DISTANCIAS" envía los datos adquiridos para ser procesados por el subVI "SENSOR", al cual se llama 6 veces, una por cada sensor. Este subVI, dibuja con color blanco la trayectoria de la señal ultrasónica de un sensor, al cual se ha indicado el ángulo de orientación actual; es decir, limpia el camino libre (con blanco) y dibuja un punto negro en la distancia indicada por el sensor dentro del gráfico "IGraph".

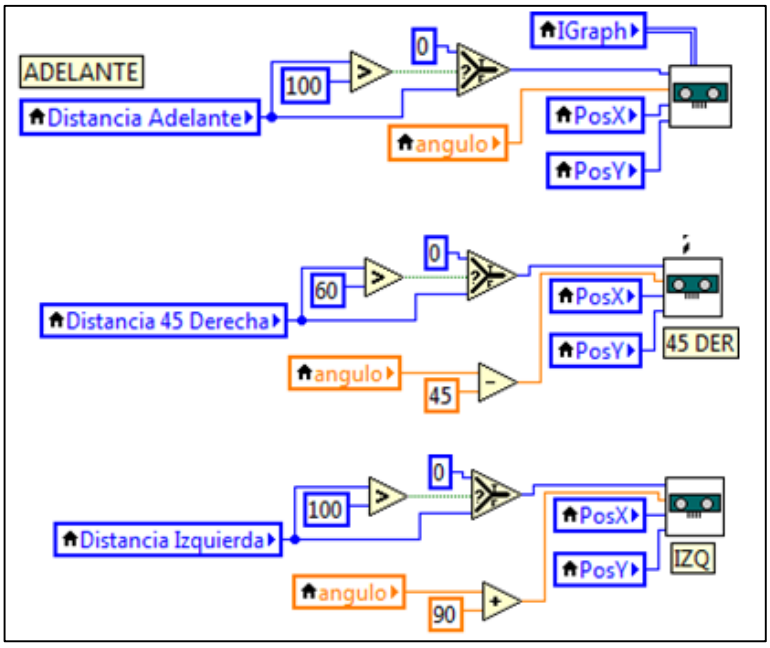


Figura 5.24 Ejemplo de uso del subVI "SENSOR" para el sensor delantero, a 45º derecha y a 90º izquierda, respectivamente.

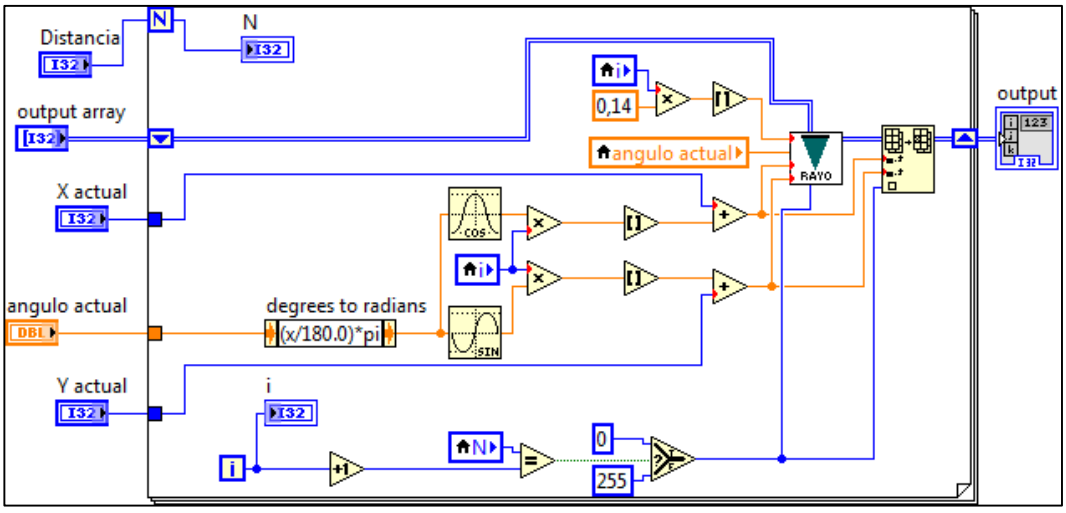


Figura 5.25 SubVI "SENSOR" dibujando sobre arreglo "IGraph".

Por el momento no se ha tomado en cuenta el ángulo de acción de cada sensor, el cual se ha establecido anteriormente en 15º. Para realizar el barrido de acción de la apertura de este ángulo sobre el arreglo de datos y a su vez sobre el control "IGraph", es necesario pasar por cada pixel del vector principal de acción del sensor y dibujar una línea perpendicular al mismo, con dimensiones que estén de acuerdo con la distancia y el ángulo, como se muestra en la Figura 5.26. El área dentro del triángulo indica el espacio sin obstáculos detectado por el sensor, por lo tanto, es necesario graficar en el mapa la representación de este espacio libre.

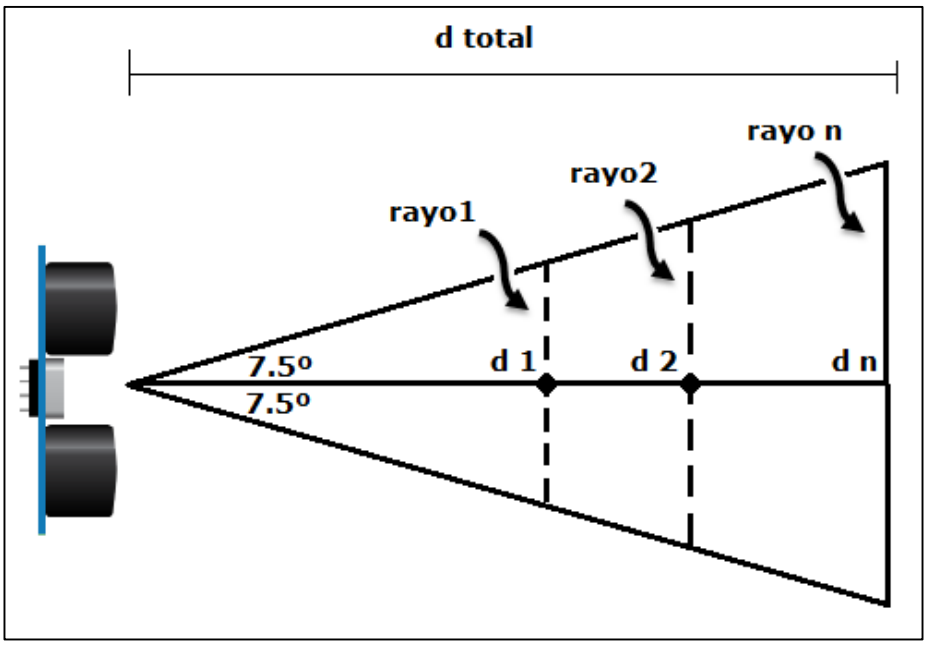


Figura 5.26 Área de acción del sensor.

La relación entre la longitud de cada rayo y la distancia se puede obtener a partir de la ley de senos. Así, se obtiene la constante de relación, la cual se toma en cuenta solo con la mitad de cada rayo, puesto que el subVI "RAYO", que se encarga de esta graficación, dibuja una mitad a la vez.

$$\frac{0.5 * rayo_i}{\sin(7.5)} = \frac{d_i}{\sin(82.5)} \qquad \frac{\sin(7.5)}{\sin(82.5)} * d_i = 0.5 * rayo_i$$

$$\frac{\sin(7.5)}{\sin(82.5)} \approx 0.14 \qquad 0.14 * d_i = 0.5 * rayo_i$$

5.2.3. Hilo de control y movimiento del robot en el mapa

Dentro de esta parte del código se encuentran los botones de control del robot en el panel frontal. Cuando son pulsados, estos envían la señal indicada a través del método POST de HTTP, manejado por el subVI "HTTP CONTROL".

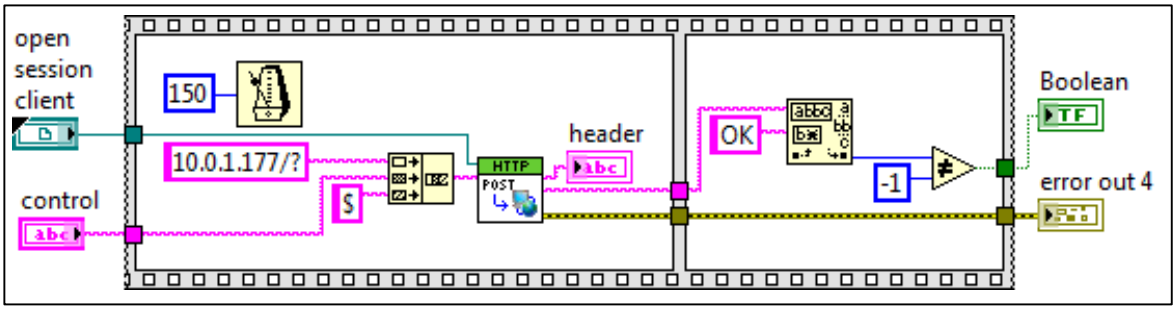


Figura 5.27 SubVI "HTTP CONTROL".

Este subVI es llamado para todos los movimientos. Sin embargo, para los movimientos de giro a la izquierda y derecha no se realiza una actualización de la posición del robot dentro del mapa, puesto que solo cambia su orientación, la cual es monitorizada por el segmento de trama adquirida correspondiente al giroscopio.

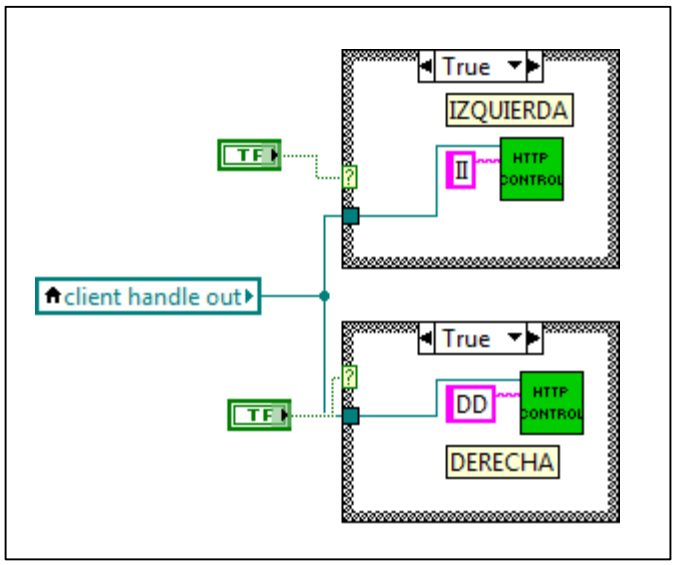


Figura 5.28 Parte del hilo de control del robot. Movimientos de giro del robot.

Por otro lado, dentro de los movimientos adelante y atrás se llama al subVI "CAMINO ROBOT", el cual actualiza la posición del robot dentro del mapa y lo desplaza 10 pixeles en la dirección indicada por el giroscopio. Se ha realizado la relación de 1 pixel a 1 cm, puesto que esta es la precisión de los sensores. Además, el movimiento de 10 pixeles con cada pulsación del botón atrás o adelante se ha definido mediante pruebas de funcionamiento, puesto que dentro del robot no existen sensores que permitan medición de la distancia recorrida (odometría), ya que la implementación de estos no mejoraría la estimación de la posición como se había explicado en capítulos anteriores.

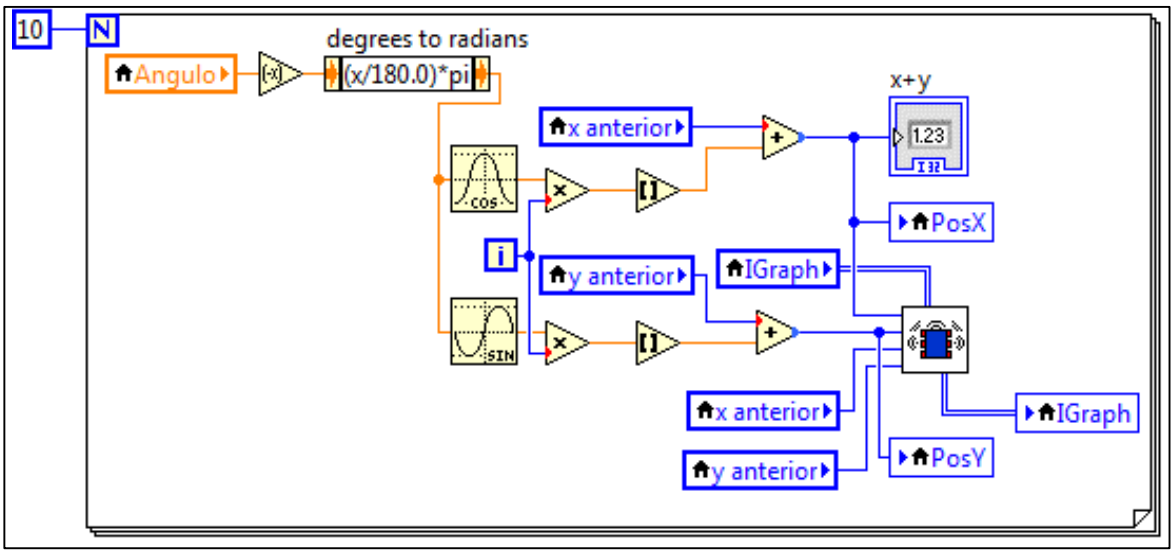


Figura 5.29 Rutina de movimiento hacia adelante llamando al subVI "CAMINO ROBOT".

En la Figura 5.30 se muestra un resumen esquematizado de la ejecución de las acciones dentro del programa principal de control; se describen las condiciones iniciales y los subVI que se ejecutan en cada uno de los dos hilos paralelos.

Por último, en la Figura 5.31 se muestra la apariencia del panel frontal, el cual posee indicadores adecuados para mostrar la información adquirida de los sensores, alarmas de mediciones peligrosas, botones de control, el mapa generado en tiempo real y el video transmitido por la cámara.

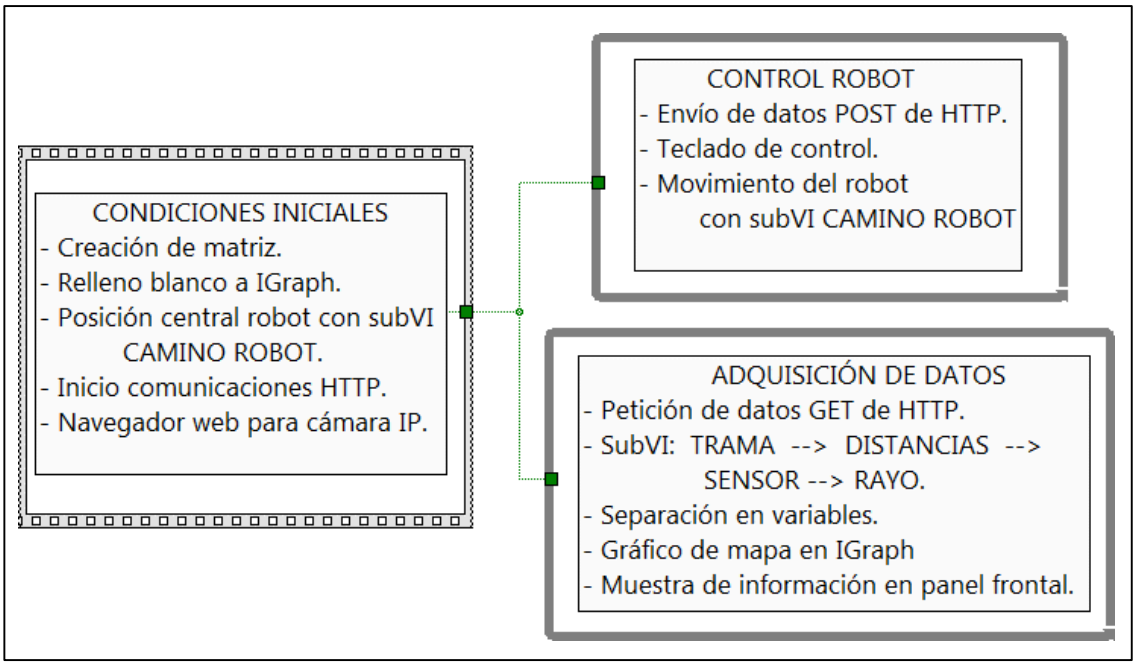


Figura 5.30 Esquema de funcionamiento del código en el computador.

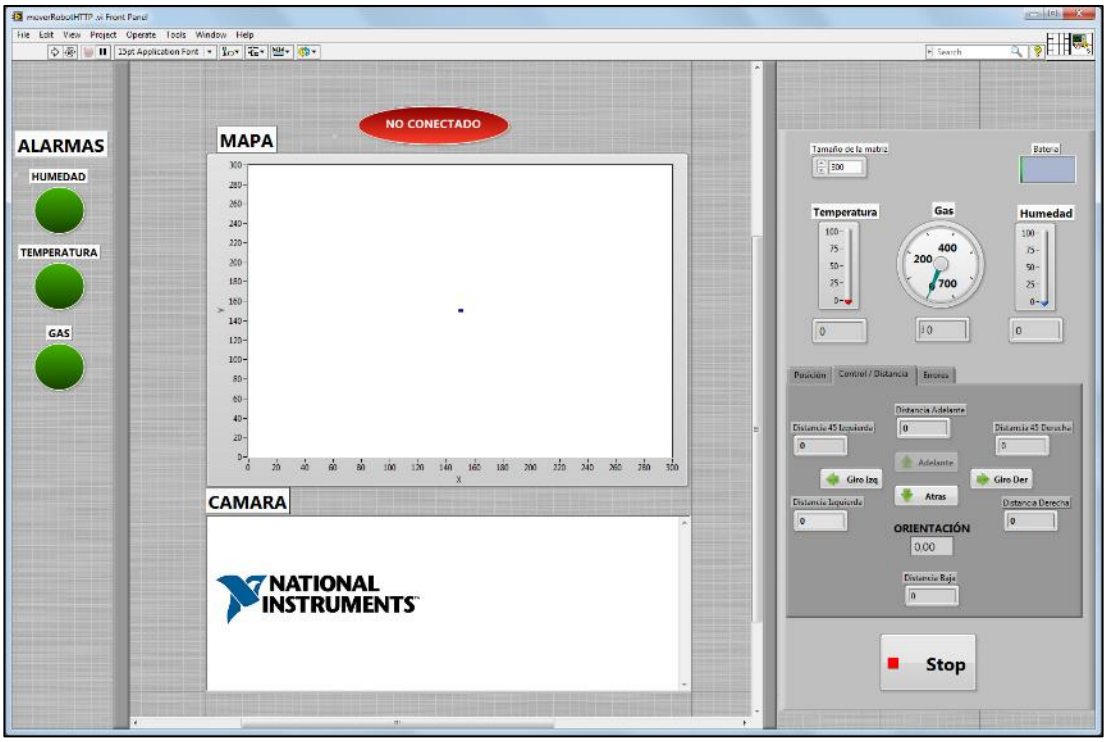


Figura 5.31 Panel Frontal de la aplicación final.

CAPÍTULO 6

PRUEBAS DE FUNCIONAMIENTO

Con el objetivo de comprobar el funcionamiento del robot se deben realizar pruebas para determinar la eficiencia del sistema. Se realizará una simulación para comprobar el correcto funcionamiento del programa de adquisición de datos de los sensores de distancia y control de los movimientos del robot. Después, se realizarán mediciones del error en la orientación, autonomía de las baterías, alcance de la señal inalámbrica y precisión del trazado del mapa con el prototipo real.

6.1. Simulación

La simulación de un sistema robótico requiere de un software de programación con características especiales. En general, los programas de simulación de robots deben poseer ciertas características para cumplir con su objetivo. La creación de un modelo dinámico que posea propiedades físicas similares a las encontradas en el mundo real, la capacidad de creación de entornos virtuales de prueba para el robot, y la aplicación de algoritmos o líneas de programación para simular tareas o comportamientos son las principales características de un simulador robótico.

A continuación, se presentarán algunas opciones de software para simulación robótica que presentan las particularidades necesarias para simular un robot móvil, de las cuales se escogerá una para la realización de pruebas dentro del computador.

6.1.1. Microsoft Robotics Studio

Microsoft Robotics Developer Studio es un entorno de programación basado en el sistema operativo Windows que permite control y simulación robótica. Está dirigido a académicos, aficionados y desarrolladores comerciales, en especial para aplicar las simulaciones sobre una plataforma robótica previamente adquirida.

Su licencia básica es de distribución libre, su código está implementado en .NET y basado en Microsoft Visual Studio. En su entorno de desarrollo se encuentran herramientas de programación visual, simulación en 3D, y una gran variedad de sensores y actuadores prediseñados. Las principales aplicaciones realizadas con este software son simulaciones de competencias como robots de batalla o robots para soccer. Adicionalmente a la simulación, este software permite comunicación con algunas plataformas robóticas a través de Wi-Fi o Bluetooth para implementación de código, por ejemplo, el robot MARK (*Mobile Autonomous Robot using Kinect*) (Figura 6.1).

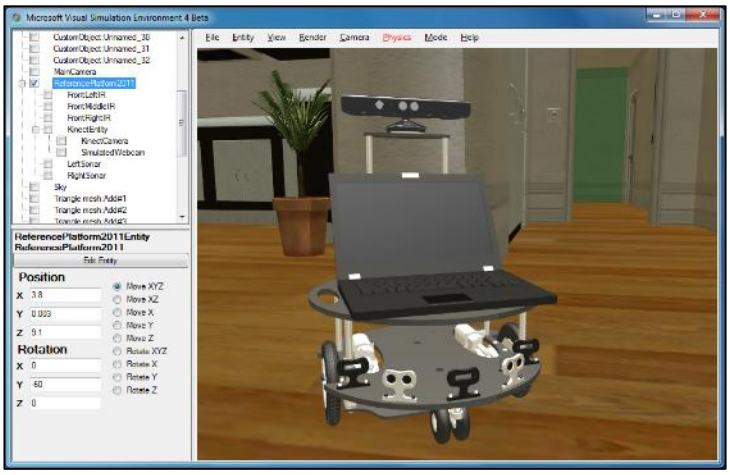


Figura 6.1 Entorno de Microsoft Robotics Studio.

Fuente: Microsoft Developer Network. Welcome to Robotics Developer Studio [en línea]. [20 enero de 2014]. Disponible en web: < http://msdn.microsoft.com/en-us/library/bb648760.aspx>.

6.1.2. LabVIEW Robotics

LabVIEW Robotics es un módulo de LabVIEW que incluye herramientas de software para el diseño de sistemas autónomos como brazos robóticos o robots móviles. Su principal bondad es la gran cantidad de librerías prediseñadas para adquisición de datos de sensores comerciales como GPS, *encoders*, sensores infrarrojos, ultrasónicos, etc. los cuales son aplicados principalmente en plataformas robóticas que incluyen hardware de National Instruments. Existen herramientas prediseñadas para procesos más complejos como filtros, rutinas de cinemática inversa y directa, establecimiento de características dinámicas, etc.

En cuanto a simulación, el módulo robótico depende del *LabVIEW Control Design and Simulation Module*, una herramienta independiente en la cual se pueden introducir los parámetros dinámicos y cinemáticos para simulación robótica, aunque esta no es su función principal, puesto que está diseñado para modelado de sistemas de control. La creación de modelos físicos de vehículos y ambientes es muy limitado, y depende de otras aplicaciones, especialmente programas de diseño de objetos vectoriales en 3D. Crear modelos físicos a partir de un robot móvil particular y aplicar comportamientos o rutinas dentro de un ambiente simulado implica gran complejidad.

Sin embargo, para plataformas de hardware propio de National Instruments, el proceso es facilitado en gran manera por las librerías prediseñadas, como es el caso del robot DaNI, parte de un kit básico de robótica para LabVIEW (Figura 6.2).

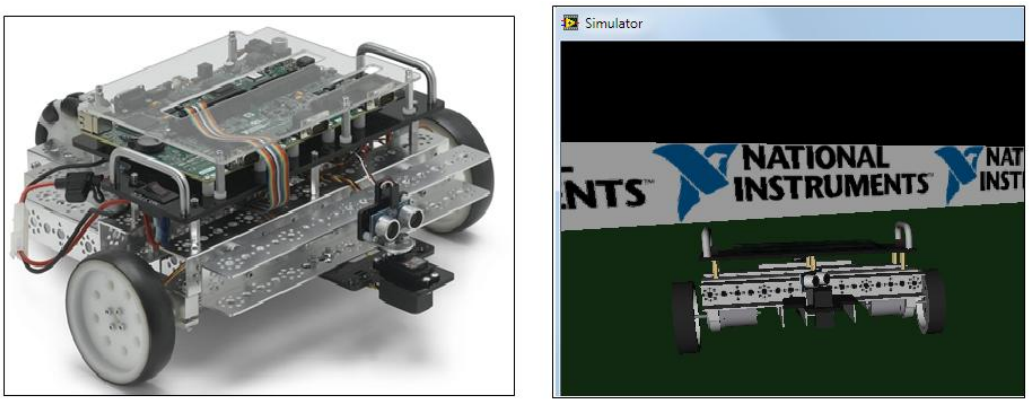


Figura 6.2 Robot "DaNI" del LabVIEW Robotics Starter Kit con entorno de simulación prediseñado.
Fuente: National Instruments. Overview of the LabVIEW Robotics Module [en línea]. [20 enero de 2014]. Disponible en web: <http://www.ni.com/white-paper/11564/en/>.

6.1.3. MATLAB

MATLAB (*MATrix LABoratory*) es una herramienta de software matemático muy potente desarrollada por MathWorks. Sus aplicaciones incluyen el cálculo a través de matrices, graficación en 2D y 3D, implementación de algoritmos y creación de interfaces de usuario. MATLAB es utilizado en amplios campos de la ciencia, desde economía, pasando por bioestadística hasta ingeniería de control, física, etc.

Uno de sus paquetes adicionales, Simulink, es un lenguaje de programación gráfica para modelado y simulación de sistemas dinámicos, muy utilizado en teoría de control y procesamiento digital de señales.

La programación y simulación de robots en MATLAB se basa en la creación del algoritmo de control que gobernará el comportamiento del robot. Este algoritmo se aplica a un modelo matemático del robot dentro de una simulación, la cual se realiza generalmente con herramientas de Simulink. Sin embargo, los modelos y simulaciones corren enteramente a cargo del programador, puesto que no existen herramientas prediseñadas que faciliten y agiliten la creación de modelos robóticos. La creación de un modelo en 3D y de un medio ambiente complejo es una tarea complicada, puesto que todo debe crearse desde cero. A pesar de esto, existen algunas plataformas robóticas educativas como LEGO MINDSTORMS que poseen herramientas prediseñadas para MATLAB y Simulink. Otros robots móviles, como el Khepera III, han sido fuente de estudio por parte de departamentos de investigación de algunas universidades, las cuales han creado simuladores de gran utilidad dentro de MATLAB (Figura 6.3).

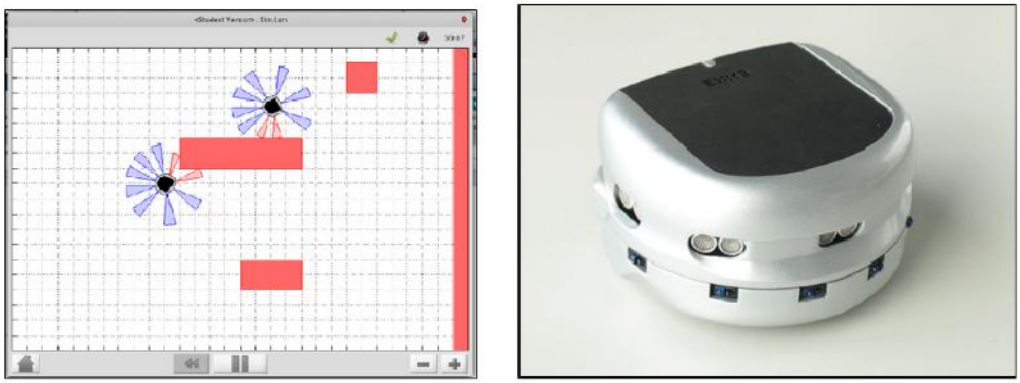


Figura 6.3 Robot Khepera III y simulación en MATLAB.

Fuente: MATLAB CENTRAL. Jean-Pierre de la Croix. Sim.I.am [en línea]. [20 enero de 2014]. Disponible en web: < http://www.mathworks.com/matlabcentral/fileexchange/40860-sim-i-am/content/simiam/+simiam/+controller/+khepera3/K3Supervisor.m>.

6.1.4. Webots

Webots es un simulador robótico profesional diseñado para propósitos educativos. El software permite la creación de prototipos robóticos virtuales y la colocación de los mismos en un ambiente en 3D cuyo objetivo es el de realizar pruebas de locomoción y estabilidad. El modelo puede incluir propiedades físicas como masa, fricción, movimientos articulares, centro de gravedad y otros, gracias a su sistema ODE (*Open Dynamics Engine*) para simulación de variables físicas. Cada elemento gráfico puede ser modificado para recrear condiciones especiales incluidas en los robots como resortes, muelles, suspensiones, materiales elásticos, etc. Además, el robot puede ser equipado con una variedad de sensores, como *encoders* para ruedas, sensores láser o ultrasónicos de distancia, cámaras, etc., muchos de los cuales están prediseñados, reduciendo el tiempo de desarrollo del modelo. Su último componente es la programación que se aplica al modelo, pudiendo utilizar lenguajes como C, C++, Java y MATLAB. Esto permite comprobar la lógica de control y el comportamiento que generará en el sistema.

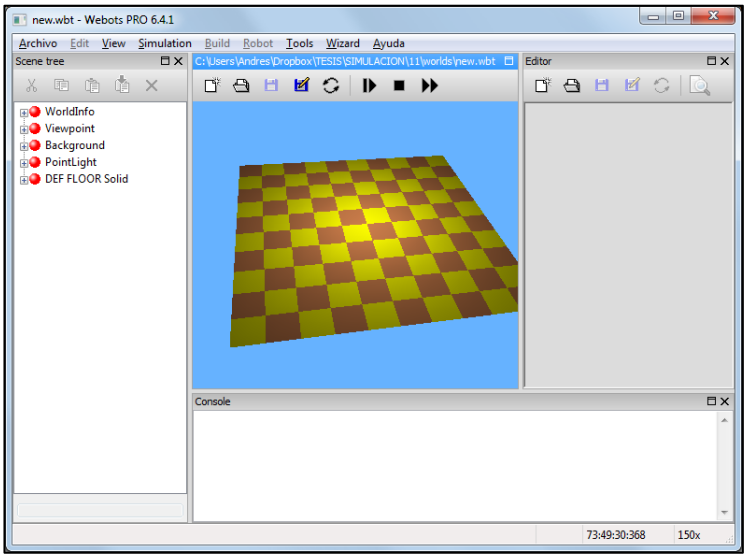


Figura 6.4 Entorno de desarrollo de Webots.

El software ofrece la posibilidad de creación de ambientes complejos con características muy variadas (inclinaciones, rocas, ondulaciones obstáculos fijos o móviles.). Además, el usuario es capaz de interactuar en tiempo real dentro de la simulación en curso, pudiendo realizarse pruebas no solo de vehículos autónomos sino de robots semiautónomos comandados por el usuario.

6.1.5. Modelado y simulación en Webots

Dentro del programa de simulación Webots se ha implementado el modelo físico del robot, basado en un modelo de muestra de 6 ruedas de las que se puede controlar sus velocidades. Se han añadido los sensores utilizados en el prototipo, es decir, un giroscopio y 6 sensores ultrasónicos en sus ubicaciones originales. Por último, se ha adicionado la cámara en su posición real. El modelo se muestra en la Figura 6.5.

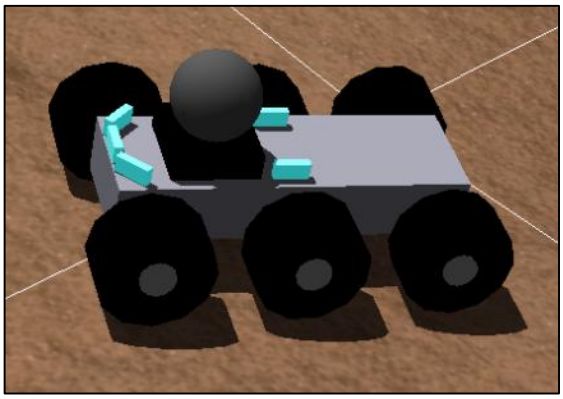


Figura 6.5 Modelo de simulación de robot en software Webots.

A continuación, se configuran los sensores de distancia con sus características de medición reales, es decir, un área de acción enmarcada en un ángulo de 15 grados y una precisión en la medición de hasta 2 m. Además, se muestra la visualización obtenida por la cámara.

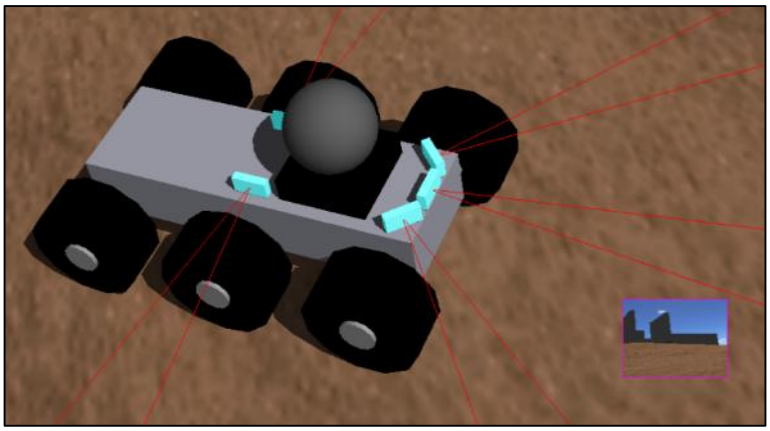


Figura 6.6 Modelo con sensores y cámara aplicados.

Con el modelo completo, se crea un ambiente similar a un laberinto, utilizando las herramientas preestablecidas del software de simulación. Por último, se aplican los algoritmos de programación utilizados en la placa de control de motores sobre el controlador del modelo de simulación y se realiza una prueba sobre el laberinto (Figura 6.7).

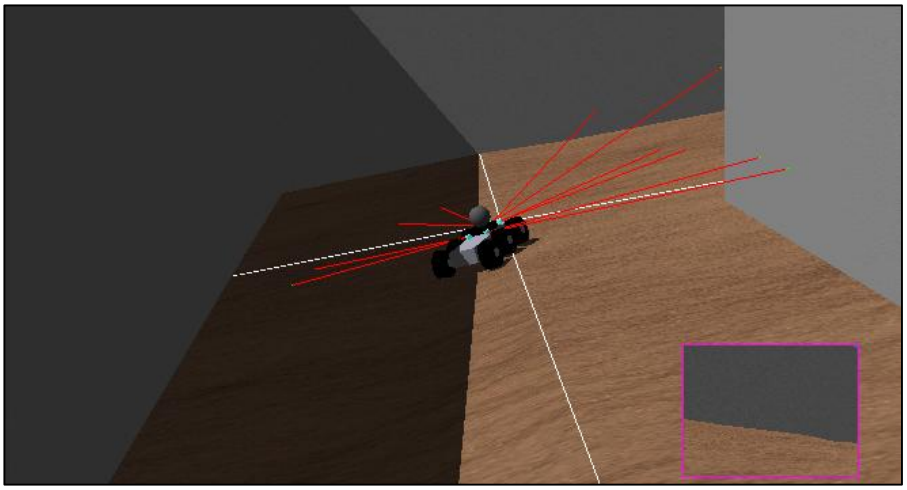


Figura 6.7 Modelo de simulación del robot dentro del laberinto.

Gracias a esta simulación se ha podido comprobar la efectividad del algoritmo planteado anteriormente para el control de movimiento en modo manual y automático.

6.2. Medición del error en la orientación

Para medir los errores de orientación se ha realizado una prueba basada en las siguientes condiciones:

- Primero se ha planteado la medición del ángulo producido cuando se le imprime la máxima señal de PWM a los motores para el giro (izquierda o derecha), durante 140 ms. Las condiciones para la adquisición de los valores son: baterías a su carga nominal generando 7,2 V y el robot colocado sobre una superficie con poca fricción. Después de cada movimiento del robot, este es colocado manualmente en su posición original, para evitar la acumulación de errores. La medición se ha realizado con los datos captados por el giroscopio, a un intervalo de 45 segundos. Al final, se ha determinado que se produce una variación del ángulo en 26 grados; este dato se tomará como el incremento de ángulo deseado.

- A continuación, se han realizado 20 movimientos sobre las mismas condiciones de fricción, pero enviando varias señales de control, es decir, sin volver a colocar al robot en su lugar, y realizando las mediciones incluyendo los errores acumulados en anteriores movimientos (Figura 6.8).

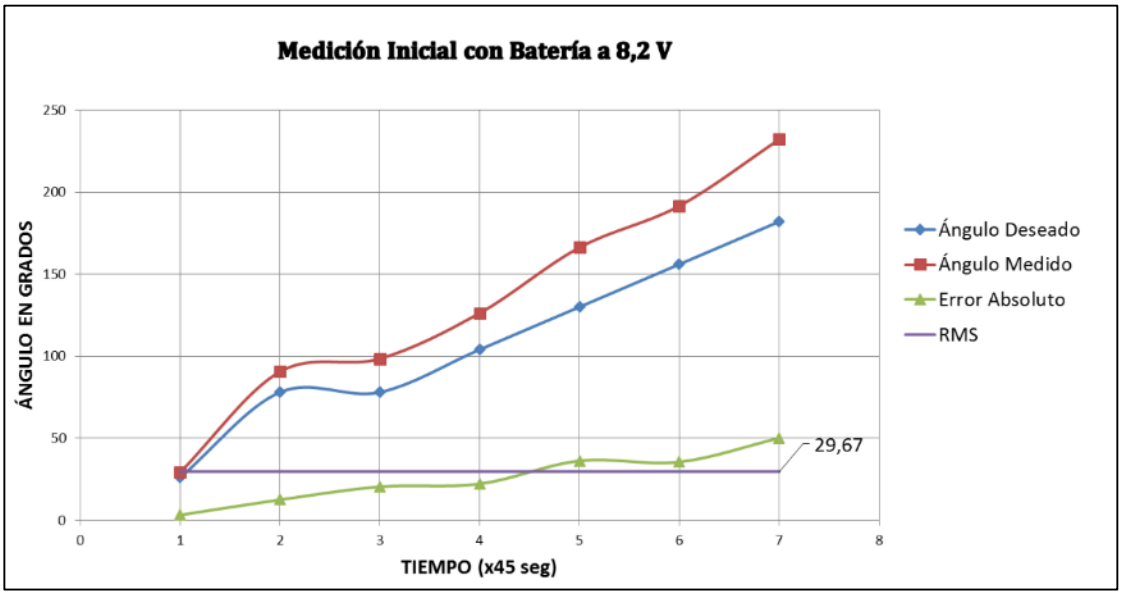


Figura 6.8 Presentación de datos por tiempo según ángulo con batería a 8,2 V.

- La anterior prueba se repite con las mismas condiciones, cambiando el estado de la batería a 7,6 V (Figura 6.9).

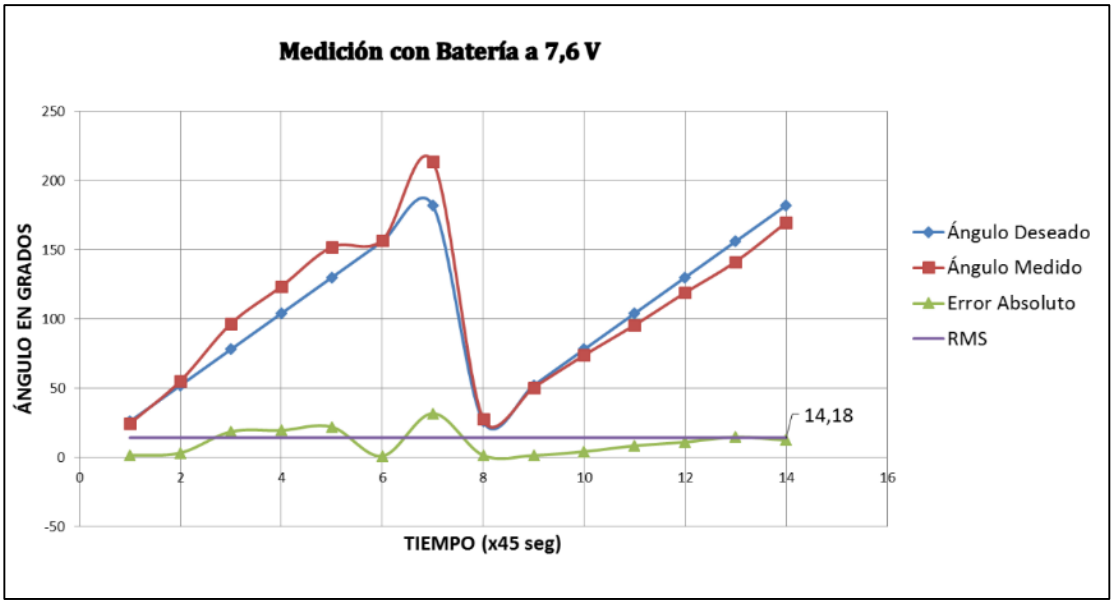


Figura 6.9 Presentación de datos por tiempo según ángulo con batería a 7,6 V.

- Por último se repite la prueba con un nivel de batería mínimo de 6,8 V (Figura 6.10).

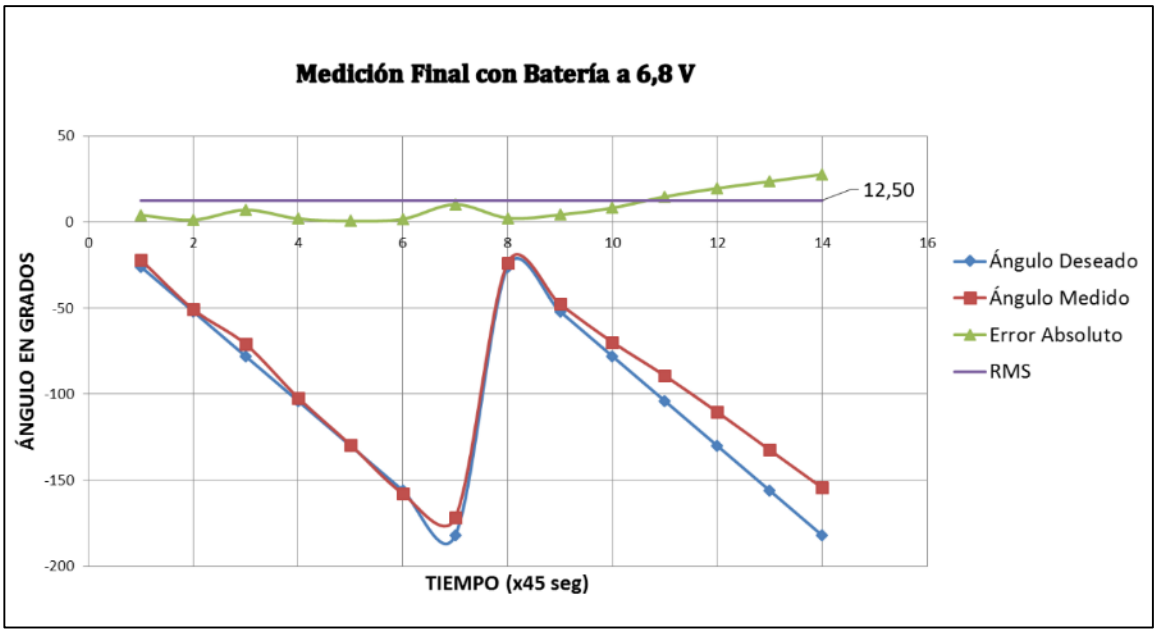


Figura 6.10 Presentación de datos por tiempo según ángulo con batería a 6,8 V.

El gráfico de la Figura 6.11 muestra la comparación de los errores medios cuadrados de las mediciones realizadas anteriormente.

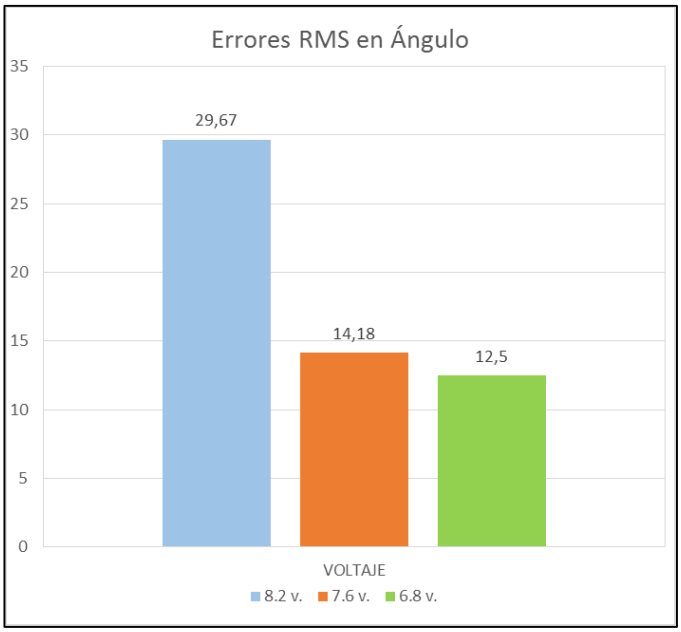


Figura 6.11 Gráfico resumen de error en ángulo por voltaje.

6.3. Autonomía de las baterías

La autonomía de las baterías ha sido probada realizando 2 experimentos tomando en consideración las siguientes condiciones:

- Se han cargado las baterías hasta su nivel máximo, en el que se encuentran a 8,2 VDC.

- Se ha activado el robot y se han realizado movimientos continuos sin detener el robot por más de 1 minuto.

- El robot ha sido conducido en caminos de tierra y en la acera, donde no se han incluido pendientes pronunciadas.

- Se ha concluido la prueba cuando la potencia de las baterías no era suficiente para lograr movimientos del robot o cuando la conexión inalámbrica a corta distancia no está disponible.

El primer experimento, con control manual, concluyó en una autonomía de 1 hora y 50 minutos. En el modo automático, la autonomía se redujo a 50 minutos, valor que ha sido estimado de manera teórica en la sección 3.8.

6.4. Alcance de la señal inalámbrica

Para medir la potencia de la señal se han utilizado dos escenarios: exteriores e interiores.

En exteriores, se ha realizado la prueba en un parqueadero con extensión de 800 m a la redonda desde la posición del robot. El equipo remoto tomado en cuenta ha sido un computador portátil Asus N82JQ con una tarjeta de conexión inalámbrica Broadcom tipo N. El computador se ha mantenido en el sitio de partida y se ha trasladado el robot hasta que la señal inalámbrica no ha sido detectada. Se ha repetido este experimento, con la batería a 8,2 V, en dos ocasiones, obteniéndose una distancia de 243 a 252 m.

La prueba en interiores se ha realizado en el edificio de la Facultad de Ciencia y Tecnología, colocando el robot en el piso superior (cuarto piso) del edificio. Dentro del mismo piso, con una separación de 40 m, la señal no ha podido ser adquirida. De la misma manera, y colocado el computador en segundo piso, a una distancia aproximada de 20 m en línea recta, la señal no se ha recuperado.

6.5. Precisión en el trazado del mapa

Para obtener los errores que se producen en la creación del mapa se ha colocado al robot dentro de un laberinto en el que se han preparado cinco escenarios comunes.

6.5.1. Entrada de túnel

El primer escenario consiste en la entrada a un túnel, en el que las paredes laterales son detectadas por los sensores izquierdo y derecho, y el sensor frontal no detecta objetos cercanos. Las paredes se encuentran a una distancia de 1,7 m y miden 1,8 m de largo. En la Figura 6.12 se puede apreciar el camino del robot en línea entrecortada azul, las paredes mapeadas se muestran en color negro, y la pared real en color rojo.

Se ha insertado las paredes de color rojo en un arreglo de datos para IGraph de LabVIEW y se han comparado las matrices, restando los elementos de la misma posición. Las matrices son de 300 x 300 pixeles (90 mil pixeles en total). Mediante esta operación se han encontrado un error total de 285 pixeles, que representan el 0,316%.

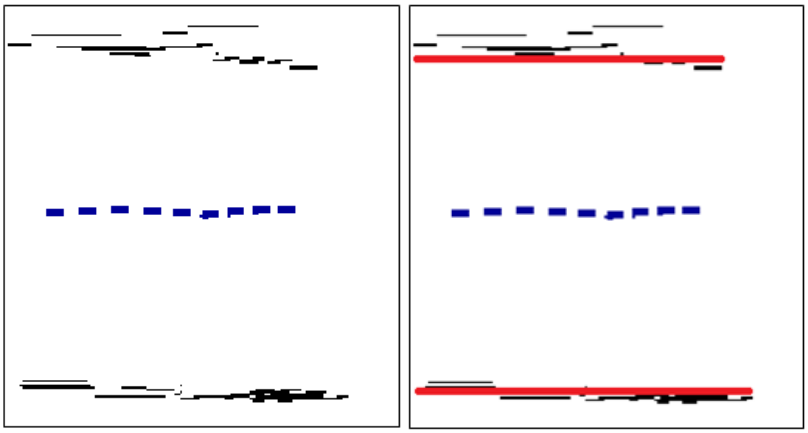


Figura 6.12 Mapeo de paredes laterales.

Sin embargo, si tomamos en cuenta solo las paredes, las reales tienen un total de 360 pixeles (180 en cada pared) ubicados horizontalmente a cada lado, y las paredes mapeadas coinciden con ellas en 75 pixeles, lo que significa un error del 80%, el cual es bastante alto. No obstante, como se puede ver en la Figura 6.12, las paredes mapeadas se encuentran desplazadas solo unos cuantos pixeles por arriba o por debajo de las líneas reales, lo que puede implicar errores en la precisión de los sensores. Se debe tomar en cuenta que estas imágenes no se encuentran modificadas por filtros, los cuales pueden ayudar mucho a corregir y reubicar los puntos.

6.5.2. Camino sin salida

En el segundo escenario se ha colocado al robot en un camino sin salida, es decir, que los sensores detectan paredes a los lados y hacia el frente. La distancia entre las paredes paralelas es de 1,7 m y su longitud es de 1,8 m. Los datos de las paredes reales se han ingresado manualmente en un arreglo de datos tipo IGraph de LabVIEW y se ha realizado la comparación con las paredes mapeadas. La sustracción de las matrices resultó en una diferencia de 450 pixeles, lo que representa el 0,5% de error. Este incremento en el error puede deberse a la interferencia acústica producida por un ambiente cerrado en el cual existen más rebotes de la señal (eco).

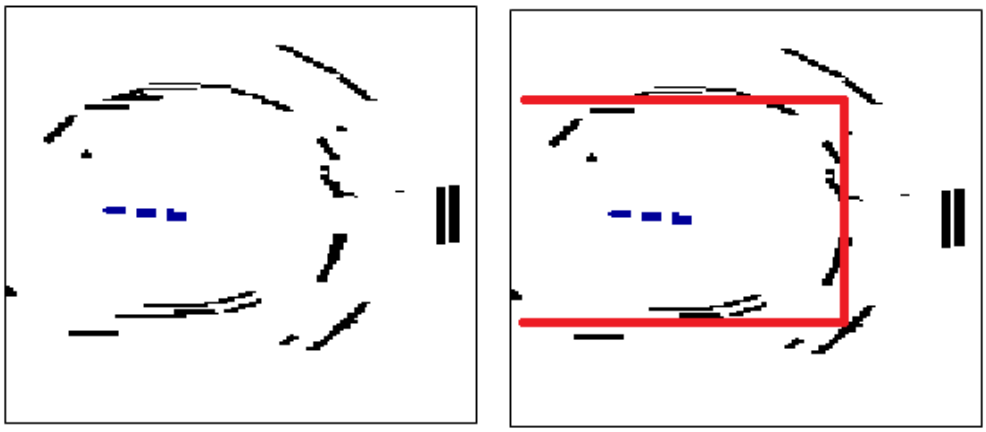


Figura 6.13 Mapeo de camino sin salida.

Tomando en cuenta solo las paredes, las reales están formadas por 530 pixeles, y las mapeadas coinciden con ellas en 82 pixeles, lo que implica un error del 84,52%. Se puede apreciar que las líneas que generan mayor error son la frontal y las de 45 grados, y se deben a la interferencia acústica recibida.

6.5.3. Esquina a 90 grados

El tercer escenario consiste en una esquina que obliga al robot a realizar un giro de 90 grados. El camino horizontal tiene una longitud de 1,8 m, y el camino vertical, 1,7. Las paredes horizontales están separadas 0,9 m y las verticales, 0,7 m. Las paredes reales están formadas por 540 pixeles en una matriz de 300 x 300 pixeles.

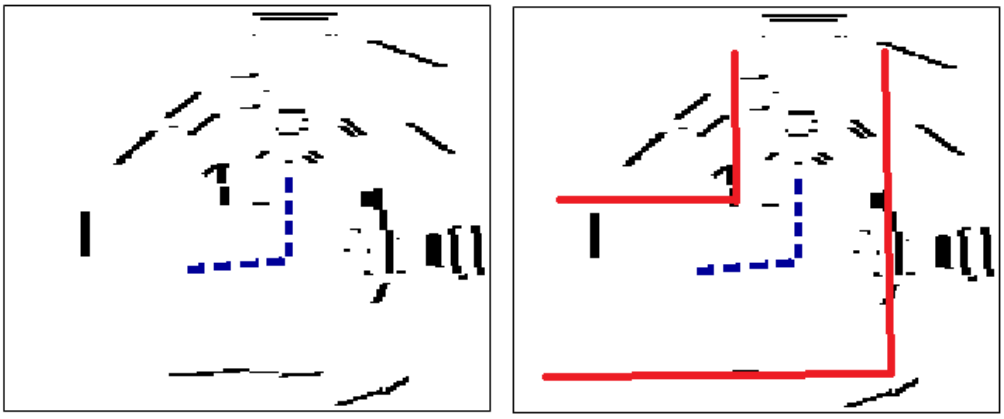


Figura 6.14 Mapeo de esquina a 90 grados.

Los datos de las paredes reales se han ingresado manualmente en un arreglo de datos tipo IGraph de LabVIEW y se ha realizado la comparación con las paredes mapeadas. La sustracción de las matrices resultó en una diferencia de 1115 pixeles, lo que representa el 1,23% de error.

Si se toman en cuenta los pixeles formados solo por las paredes reales de la esquina (540), se obtiene que las paredes mapeadas coinciden en 92 pixeles, representando el 82,96% de error.

6.5.4. Camino en forma circular

En el cuarto escenario se ha colocado al robot en un camino con forma circular. La distancia entre las paredes es de 0,67 m, la longitud de la semicircunferencia interna es de 3 m, mientras que la externa es de 5 m. Los datos de las paredes reales se han ingresado manualmente en un arreglo de datos tipo IGraph de LabVIEW y se ha realizado la comparación con las paredes mapeadas. La sustracción de las matrices resultó en una diferencia de 723 pixeles, lo que representa el 0,803% de error.

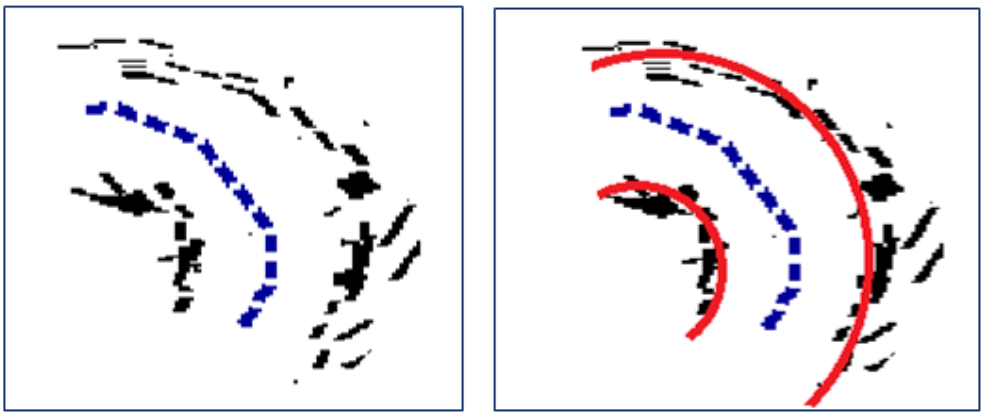


Figura 6.15 Mapeo de camino circular.

Tomando en cuenta solo las paredes, las reales están formadas por 800 pixeles, y las mapeadas coinciden con ellas en 77 pixeles, lo que implica un error del 90,375%. Se puede apreciar que las líneas que generan mayor error son las laterales, sin embargo, se ha intentado recrear lo más real posible las curvas en las paredes, pero al final el ángulo del robot y las mediciones están relacionadas entre sí.

6.5.5. Camino Sinuoso

En el quinto escenario se ha colocado al robot en un camino sinuoso, intentando simular las condiciones de una mina. La longitud de cada pared es de 3,9 m y 3,4 m, respectivamente. Los datos de las paredes reales se han ingresado manualmente en un arreglo de datos tipo IGraph de LabVIEW y se ha realizado la comparación con las paredes mapeadas. La sustracción de las matrices resultó en una diferencia de 360 pixeles, lo que representa el 0,41 % de error.

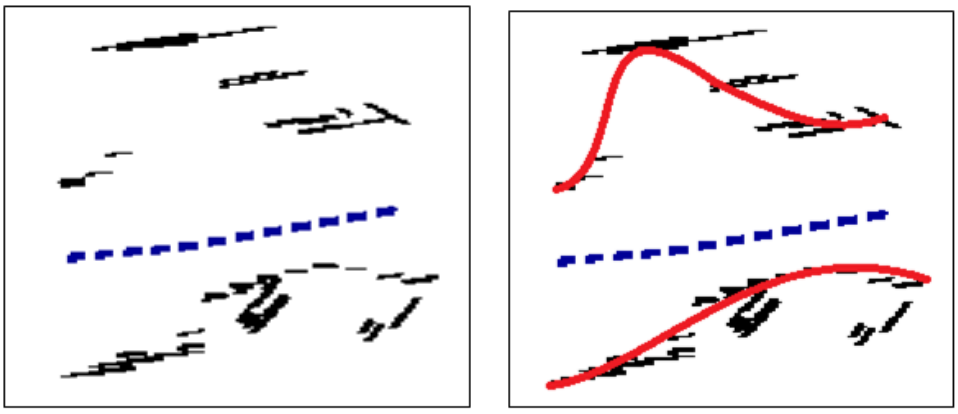


Figura 6.16 Mapeo de camino sinuoso.

Tomando en cuenta solo las paredes, las reales están formadas por 730 pixeles, y las mapeadas coinciden con ellas en 370 pixeles, lo que implica un error del 49,31%. Se puede apreciar que la línea que genera mayor error es la lateral más sinuosa, esto se produce ya que los sensores no poseen movimientos angulares para realizar un barrido en lugares internos.

6.5.6. Análisis de los gráficos

En la Figura 6.17 se muestra un resumen de los datos recogidos de los mapas recreados anteriormente. Las barras rojas indican la cantidad de pixeles graficados de las paredes reales y las barras grises indican los pixeles dibujados por el robot que han coincidido con los reales.

La línea de tendencia verde representa el error porcentual de los datos anteriores, y la azul representa el error cuadrático medio (78,77%).

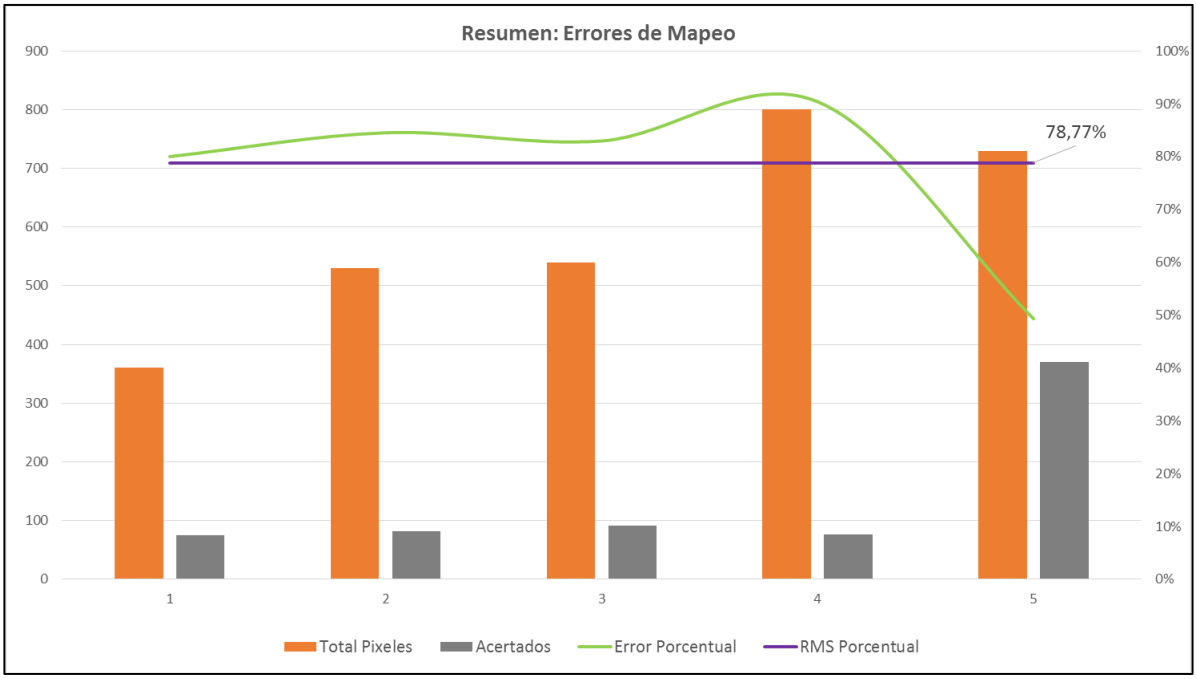


Figura 6.17 Gráfico de resumen de los errores en el mapeo por caso según porcentaje y pixeles.

CONCLUSIONES

- El chasis seleccionado ha probado ser adecuado para su uso sobre terrenos agrestes similares a los encontrados en minas subterráneas. Sin embargo, su tamaño, especialmente el de sus ruedas, limita los obstáculos que puede sortear sin provocar riesgo de daño o volcamiento.

- El tamaño del chasis limita la capacidad para colocar sistemas de control más complejos como un minicomputador.

- Debido a la necesidad de manejo de varios periféricos (sensores, actuadores, protocolos de comunicación interna) se han implementado varias tarjetas de microcontroladores, puesto que no existen prestaciones suficientes en cuanto a número de puertos (pines) y capacidad de procesamiento en una sola tarjeta para implementar todo el sistema.

- El sensor de temperatura y humedad no es sensible a cambios bruscos en estas variables, pero se considera adecuado para esta aplicación. Por otra parte, el sensor MQ-4 tiene una sensibilidad muy alta a cambios en la concentración de gas metano, lo que puede advertir sobre zonas peligrosas donde haya alta presencia de este gas.

- A partir de los datos adquiridos, y en base a la Figura 6.11, existe mayor error en la orientación (ángulo) cuando las baterías se encuentran sobre el límite de 8,2V (29,67 grados) producido al recargar al máximo las baterías. Una solución a este error podría ser la incorporación de un regulador inteligente de voltaje, el cual disminuya o aumente el nivel de voltaje entregado por las baterías según sea necesario.

- Los sensores de distancia ultrasónicos son adecuados para detectar obstáculos y paredes cercanas. No obstante, en ciertas condiciones, se producen errores en la medición debido a que los sitios son cerrados y generan interferencias acústicas. Además, el funcionamiento en conjunto de 6 sensores provoca que estos se interfieran entre sí en algunos casos.

- Para la medición de la distancia recorrida por el robot se ha realizado un estimado basado en pruebas de funcionamiento. Esto se debe a que un sistema de medición interno, con *encoders* mecánicos u ópticos, no aporta precisión a la estimación de la posición debido a los constantes deslizamientos de las ruedas en el terreno.

- Los resultados obtenidos en las pruebas de generación de mapas presentan altos porcentajes de error. Según la Figura 6.17 no existe una correlación entre la dificultad de la zona de acción del robot y el error producido al mapear esta zona. Es decir que el error no aumenta o disminuye según la dificultad del terreno, sino que tiene un comportamiento aleatorio. Este comportamiento puede mejorarse usando algoritmos de procesamiento de datos, en especial filtros de imágenes y corrección de pixeles.

- De los datos obtenidos en las pruebas se pudo determinar que, debido al alto porcentaje de error, se recomienda utilizar sensores infrarrojos o laser LIDAR (*Light Detection and Ranging o Laser Imaging Detection and Ranging*), los cuales son muy utilizados para este tipo de aplicaciones. Algunos de ellos permiten un mapeo en 3 dimensiones a alta velocidad y precisión; sin embargo, su precio es elevado. Se recomienda el uso de este tipo de sensores para aplicaciones que requieren mayor precisión o para una aplicación de carácter industrial.

BIBLIOGRAFÍA

1. SIEGWART, Roland, NOURBAKHSH, Illah Reza y SCARAMUZZA, Davide. *Introduction to Autonomous Mobile Robots.* Massachusetts : MIT press, 2011. ISBN 0-262-19502-X.

2. BRÄUNL, Thomas. *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems.* Second Edition. Berlín : Springer, 2008. ISBN-10 3-540-34318-0.

3. BORENSTEIN, Johann, EVERETT, H. R. y FENG, Ligiang. *Where am I? Sensors and Methods for Mobile Robot Positioning.* s.l. : University of Michigan, 1996. Vol. 119.

4. STACHNISS, Cyrill. *Robotic Mapping and Exploration.* Berlin : Springer-Verlag, 2009. ISBN 978-3-642-01096-5.

5. SHAFFER, Gary y STENTZ, Anthony. A Robotic System for Underground Coal Mining. *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference.* 1992.

6. FERGUSON, David I, y otros. An Autonomous Robotic System for Mapping Abandoned Mines. *NIPS.* 2003.

7. PASTOR MORRIS, Wilson. *Informe de Gestión 2012.* Ministerio de Recursos Naturales No Renovables del Ecuador. Quito : Dirección de Comunicación Social 2013, 2012. pág. 33.

8. MANDOW, Anthony, y otros. Experimental kinematics for wheeled skid-steer mobile robots. *En Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on. IEEE.* 2007, págs. pp. 1222-1227.

9. LUCET, Eric, y otros. Dynamic control of the 6WD skid-steering mobile robot RobuROC6. 2009, Vols. Intelligent Robots and Systems, 2009, IROS 2009.

10. ALVES BARBOSA DE OLIVEIRA VAZ, Daniel, INOUE, Roberto y GRASSI, Valdir. Kinodynamic Motion Planning of a Skid-Steering Mobile Robot Using RTTs. *Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American.* Octubre de 2010, págs. 73-78.

11. BANZI, Massimo. *Getting Started with Arduino.* [ed.] Brian Jepson. First Edition. s.l. : O'Reilly Media Inc., 2009. ISBN: 978-0-596-15551-3.

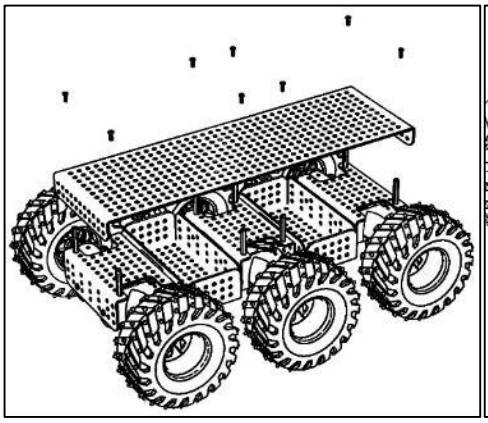
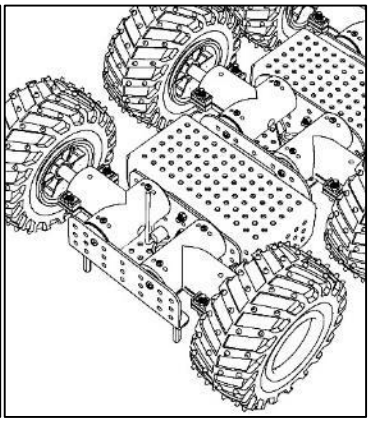
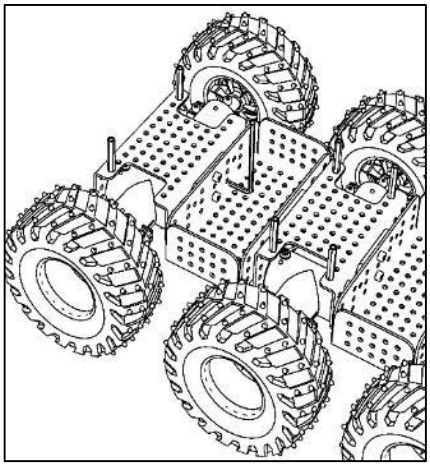
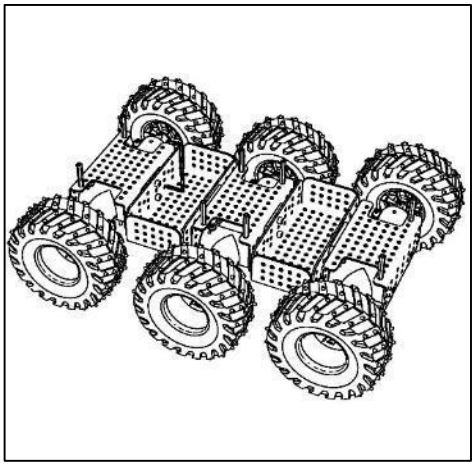
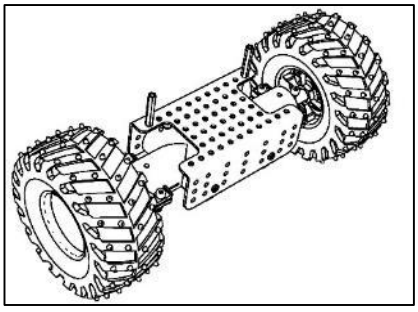
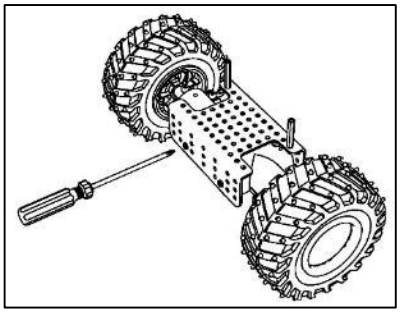
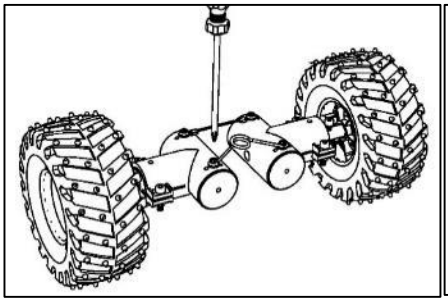
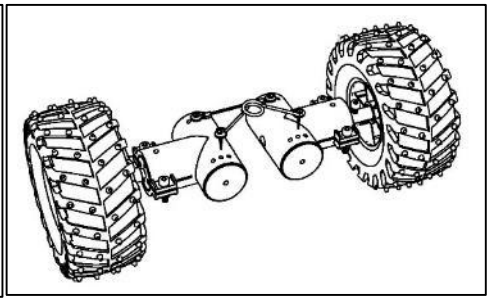
12. MARGOLIS, Michael. *Arduino Cookbook.* [ed.] Shawn Wallace and Brian Jepson. Second Edition. s.l. : O'Reilly Media, 2012. ISBN: 978-1-449-31387-6.

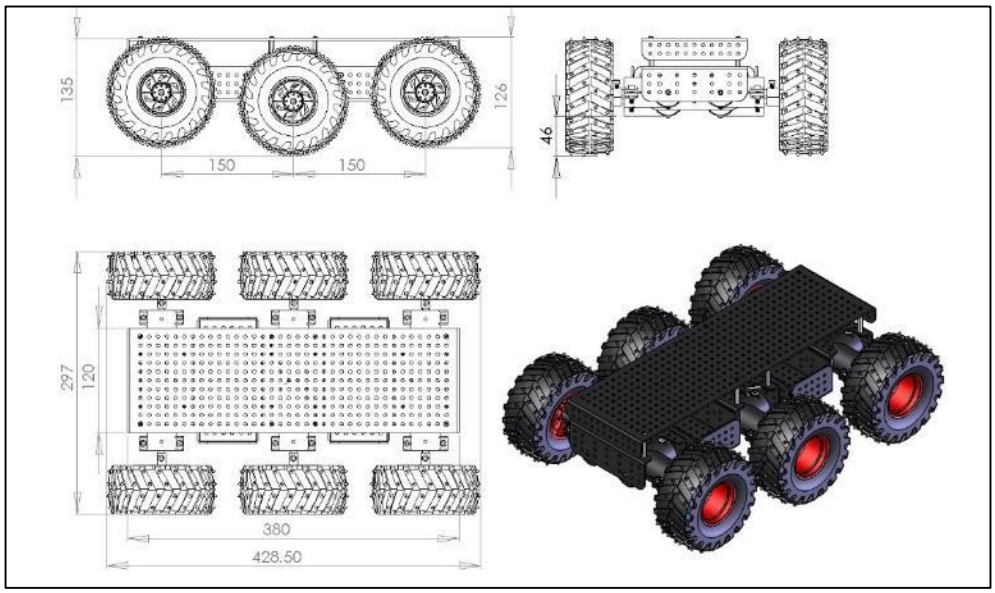
13. SHIRALKAR, Malan y National Instruments. *LabVIEW Graphical Programming Course.* [ed.] National Instruments. Houston : Rice University, 2007.

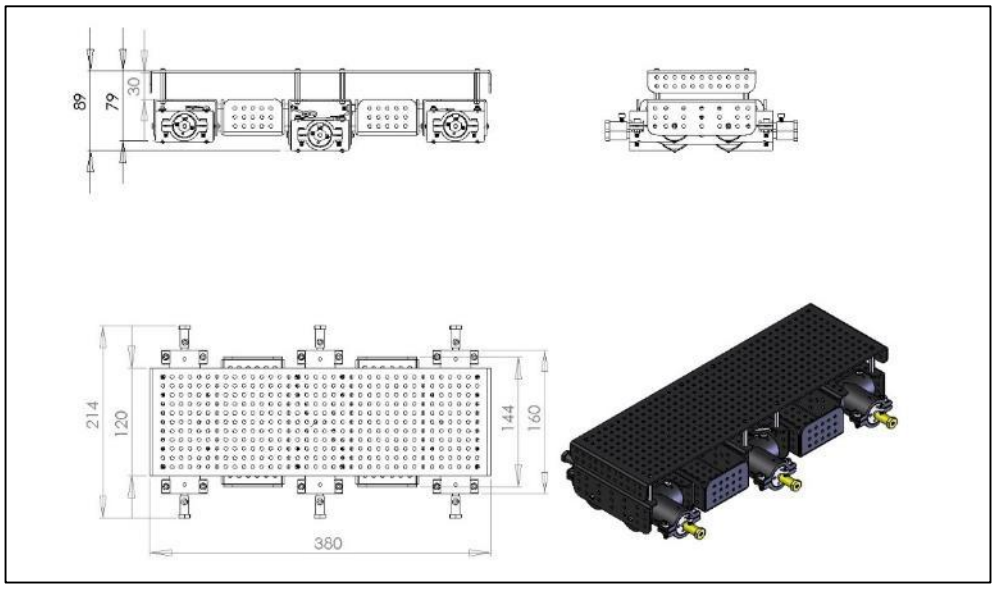
14. BITTER, Rick, MOHIUDDIN, Taqui y NAWROCKI, Matt. *LabVIEW Advanced Programming Techniques.* Second Edition. Boca Raton : CRC Press. ISBN-10: 0-8493-3325-3.

ANEXOS

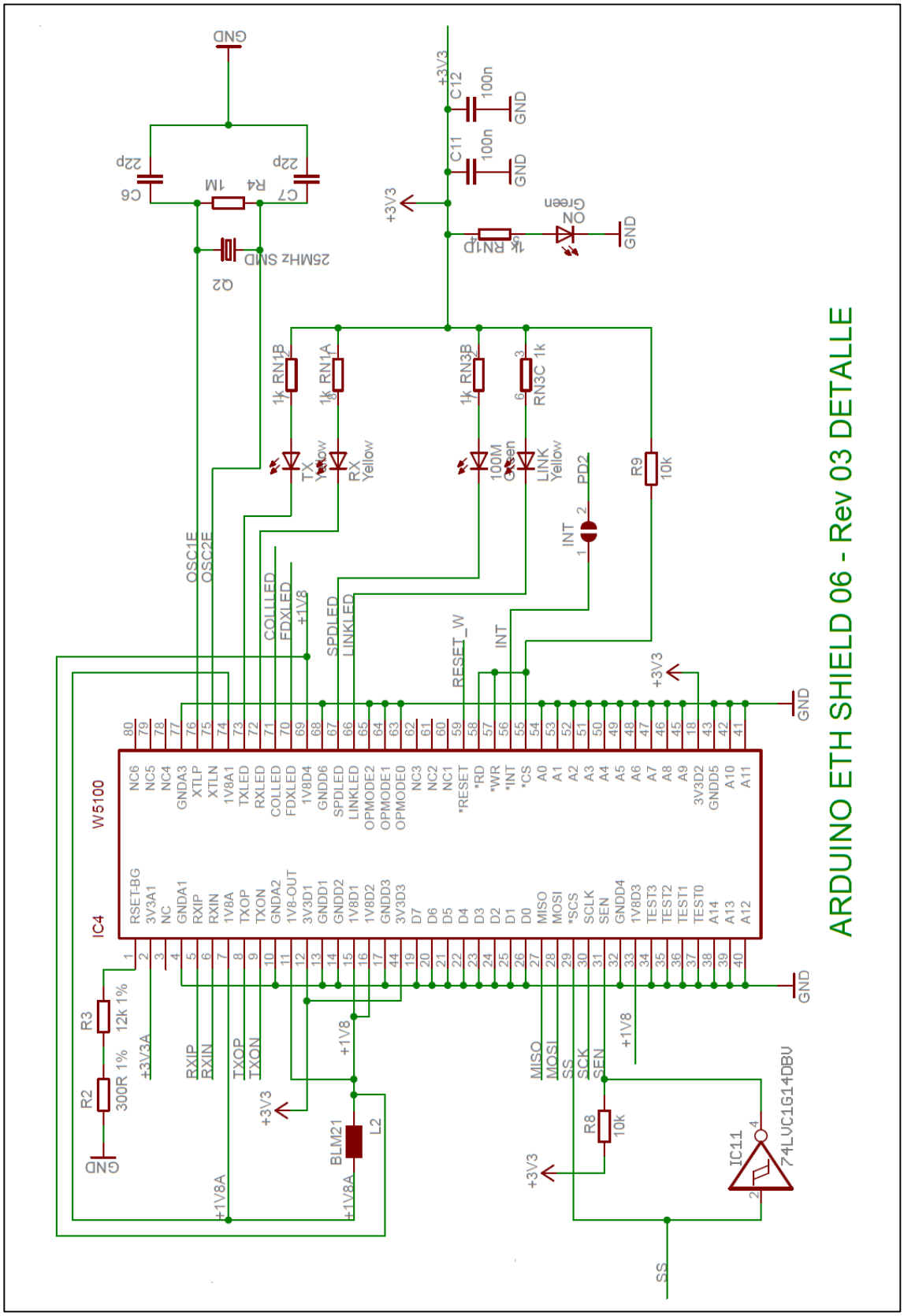
ANEXO 1: Manual de ensamblaje Wild Thumper

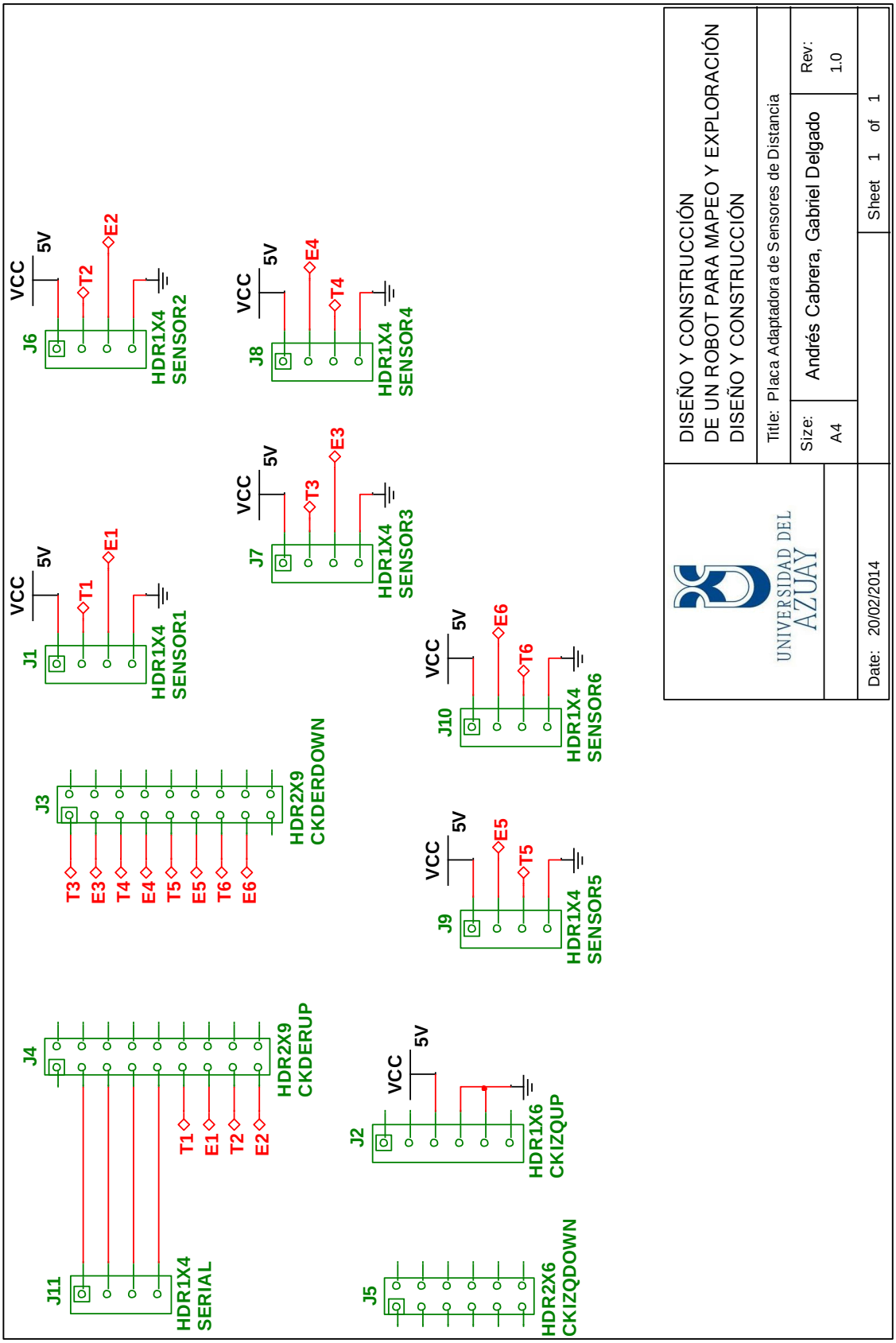

135
126
150
150
46
297
120
380
428.50


89
79
30
214
120
144
160
380

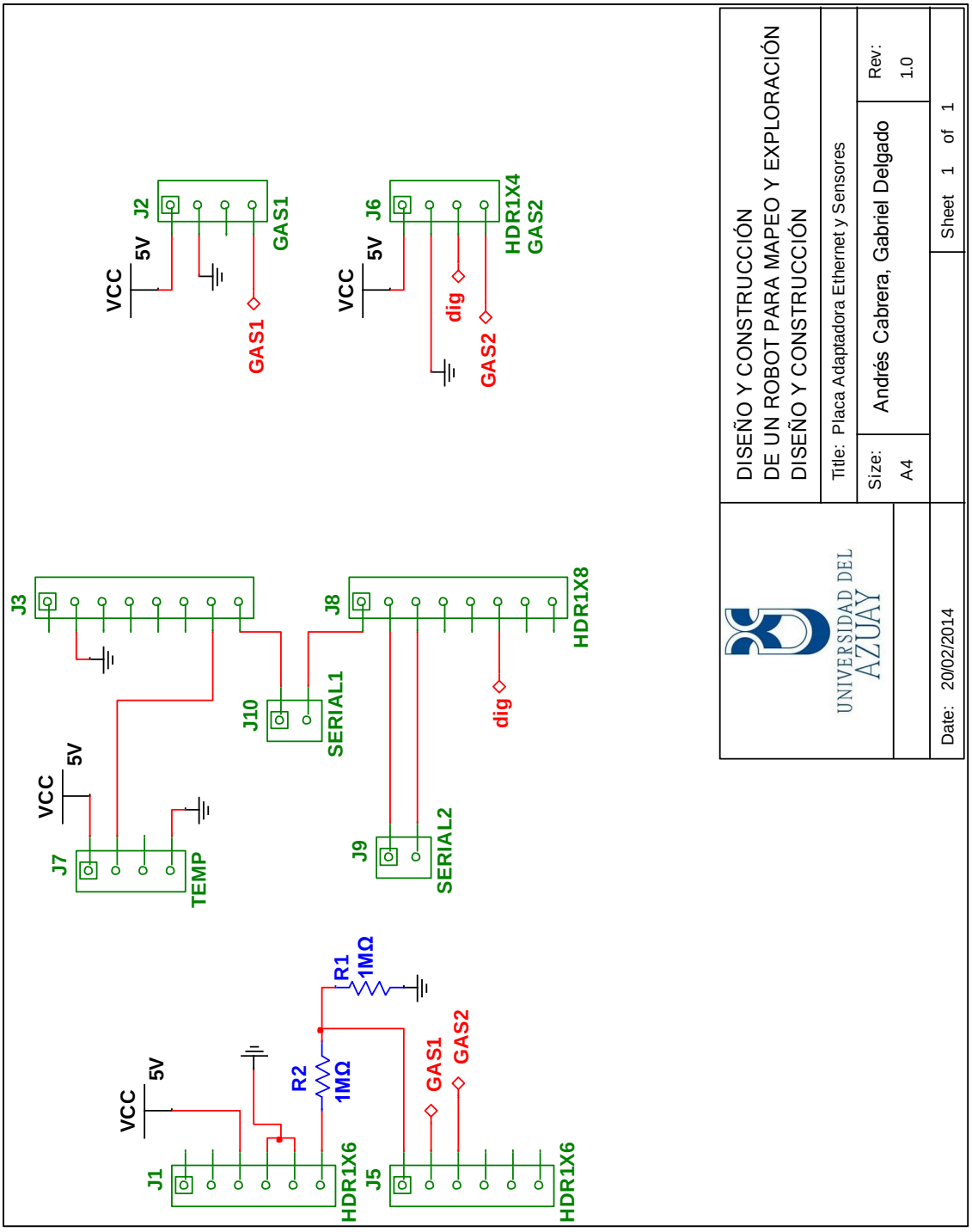
ANEXO 2: Esquema Shield Ethernet



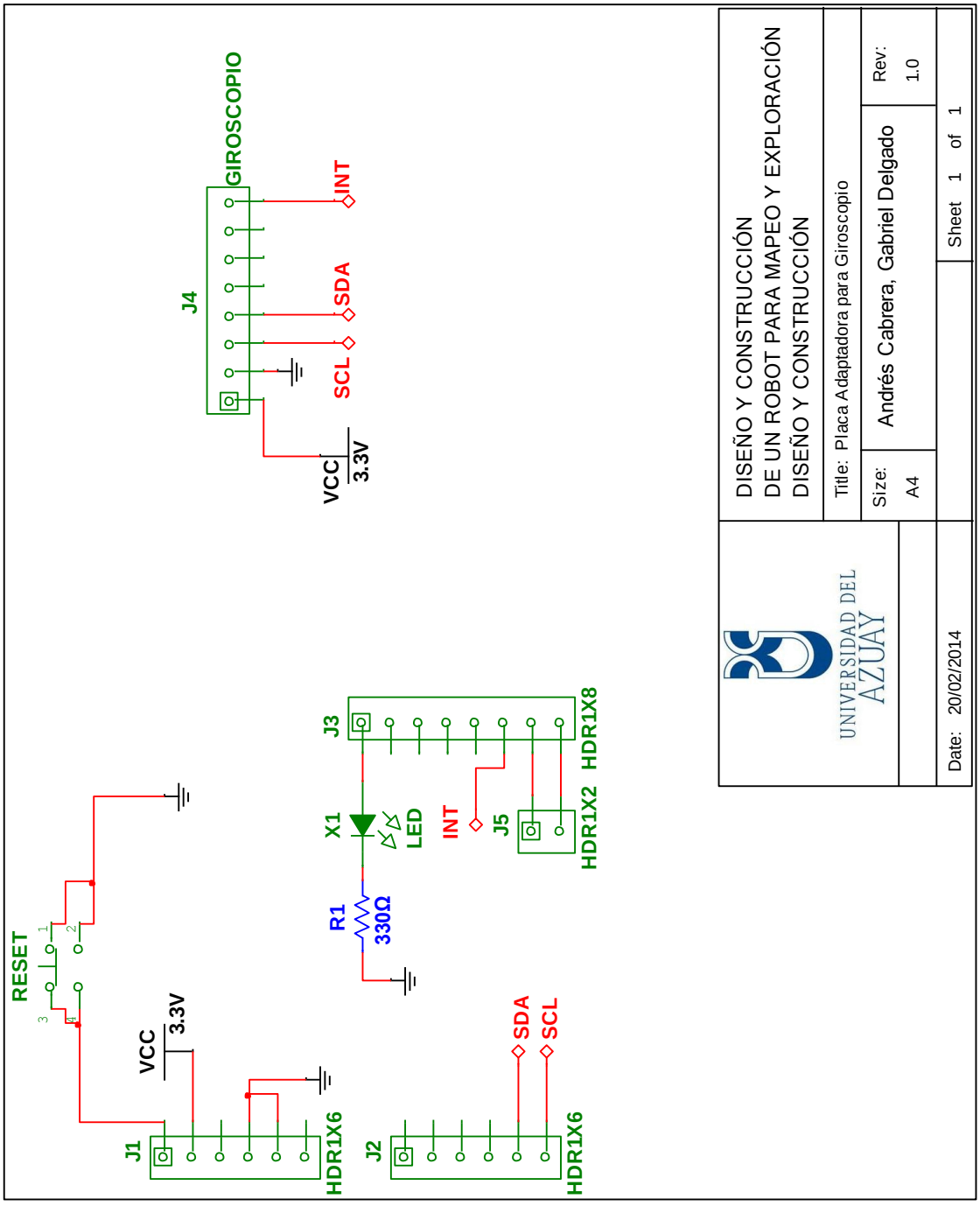
ANEXO 3: Esquema de Placa adaptadora de sensores de distancia y motores



ANEXO 4: Esquema de Placa adaptadora de Ethernet y sensores



ANEXO 5: Esquema de Placa adaptadora para Giroscopio



ANEXO 6: Código Controlador de Motores

```
#include  "DualVNH5019MotorShield.h"//Librería  del  shield  de
motores
#include <Ultrasonic.h>

DualVNH5019MotorShield md;
//Las velocidades de los motores se pueden establecer entre -400
y 400
Ultrasonic IZQ(37,36);   // (Trig PIN,Echo PIN)
Ultrasonic DER(35,34);
Ultrasonic FRONT(33,32);
Ultrasonic DOWN(30,31);
Ultrasonic FIZQ(28,29);
Ultrasonic FDER(26,27);

int a_izq;
int a_der;
int front;
int down;
int d_izq;
int d_der;
char angle;
char var;
String trama;
boolean contgiro = true;
char flag;

=============================================================
                          SETUP
=============================================================
void setup()
{
  angle='0';
  Serial1.begin(115200);
  Serial.begin(115200);
  md.init();
}
=============================================================
                          LOOP
=============================================================
void loop()
{
           leerGyro();
           leerSensores();
           Serial1.print("A");
           Serial1.print(a_izq);
           Serial1.print("B");
           Serial1.print(a_der);
           Serial1.print("C");
           Serial1.print(front);
           Serial1.print("D");
           Serial1.print(down);
           Serial1.print("E");
           Serial1.print(d_izq);
```

```
54            Serial1.print("J");
55            Serial1.print(d_der);
56            Serial1.print("X");
57            Serial1.print(trama);
58            Serial1.println("F");
59             md.setM1Speed(0);
60             md.setM2Speed(0);
61             controlremoto();
62 }
63
64 void leerGyro()
65 {
66   Serial.println("P");
67
68   while (Serial.available()) {
69    angle = Serial.read();
70
71    if(angle == 'X'){
72      trama="";
73      angle = Serial.read();
74           while (angle != 'F' && angle != '\n'){
75                  delay(1);
76                  trama.concat(angle);
77                  angle = Serial.read(); //serial
78            }// FIN MIENTRAS NO SEA F NI \N
79
80      Serial.flush();
81      break;
82    }
83    else {
84      Serial.flush(); //serial
85     }
86  }
87 }
88
89 //RUTINA QUE DETECTA SI SE HAN ENVIADO DATOS DE CONTROL
90 void leerSensores()
91 {
92  a_izq=    IZQ.Ranging(CM);// CM or INC
93  a_der=    DER.Ranging(CM);
94  front=    FRONT.Ranging(CM);
95  down=     DOWN.Ranging(CM);
96  d_izq=    FIZQ.Ranging(CM);
97  d_der=    FDER.Ranging(CM);
98 }
99
100 //RUTINA QUE DETECTA SI SE HAN ENVIADO DATOS DE CONTROL
101 void controlremoto()
102 {
103
104   if(Serial1.available())   {
105     var=Serial1.read();
106
107        switch (var) {
108        //adelante
109        case 'A':
110          md.setM1Speed(250);
```

```
111           md.setM2Speed(250);
112           delay(50);
113           break;
114         //atras
115         case 'B':
116           md.setM1Speed(-250);
117           md.setM2Speed(-250);
118           delay(50);
119           break;
120
121         //derecha
122         case 'D':
123           md.setM1Speed(-300);
124           md.setM2Speed(300);
125           delay(70);
126           break;
127
128         //izquierda
129         case 'I':
130           md.setM1Speed(300);
131           md.setM2Speed(-300);
132           delay(70);
133           break;
134
135           //return
136           case 'R':
137           retorno();
138           break;
139                     }
140         stopIfFault();
141         }
142     }
143 // RUTINA QUE DETIENE MOTORES SI HAY SOBRECARGA //
144 void stopIfFault()
145 {
146   if (md.getM1Fault())
147   {
148     Serial1.println("M1 fault");
149     while(1);
150   }
151   if (md.getM2Fault())
152   {
153     Serial1.println("M2 fault");
154     while(1);
155   }
156 }
157
158 void retorno()
159 {
160           if(contgiro){
161           giro();
162
163         } else
164         {
165           //REGRESAR CON LOS SENSORES
166           if (a_izq >= 100){
167
```

```
168             md.setM1Speed(300);
169             md.setM2Speed(300);
170             delay(150);
171 
172           }else if(d_izq >=100){
173 
174             md.setM1Speed(300);
175             md.setM2Speed(-300);
176             delay(150);
177             md.setM1Speed(0);
178             md.setM2Speed(0);
179             delay(500);
180 
181            }else if (d_der >= 100){
182 
183             md.setM1Speed(-300);
184             md.setM2Speed(300);
185             delay(150);
186             md.setM1Speed(0);
187             md.setM2Speed(0);
188             delay(500);
189           }
190         }
191 }
192 
193 void giro()
194 {
195            while (flag != 'M') {
196                 Serial.println("R");
197                 while (Serial.available() < 0);
198                 flag = Serial.read();
199               }
200           while (flag != 'T'){
201            Serial.println("W");
202             Serial.flush();
203                 md.setM1Speed(-300); // Gira Derecha
204                 md.setM2Speed(300);
205                 delay(200);
206                 md.setM1Speed(0);
207                 md.setM2Speed(0);
208                 delay(500);
209              flag = Serial.read();
210             Serial.flush();
211           }
212   contgiro=false;
213 }
```

ANEXO 7: Código Giroscopio

```
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

MPU6050 mpu;

#define OUTPUT_READABLE_YAWPITCHROLL

#define LED_PIN 13
bool blinkState = false;


// MPU control/Variables de Estado
bool dmpReady = false;
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize
uint16_t fifoCount;
uint8_t fifoBuffer[64]; // FIFO buffer

// orientación
Quaternion q;
VectorInt16 aa; // [x, y, z] accel sensor measurements
VectorInt16 aaReal; // [x, y, z] gravity-free accel sensor
VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor
VectorFloat gravity; // [x, y, z] gravity vector
float euler[3];  // [psi, theta, phi]    Euler angle
float ypr[3]; // [yaw, pitch, roll]  yaw/pitch/roll
int variable;
char ventrada;

uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0,
0,0, 0x00, 0x00, '\r', '\n' };

// ================================================================
// RUTINA INTERRUPCION
// ================================================================
volatile bool mpuInterrupt = false;
void dmpDataReady() {
    mpuInterrupt = true;
}

// ================================================================
// CONFIGURACIÓN
// ================================================================

void setup() {
   pinMode(7, OUTPUT);
   digitalWrite(7, LOW);
  delay(4000); //esperar al chipkit
```

```
54          #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
55          Wire.begin();
56          TWBR = 24; // 400kHz I2C clock
57      #elif            I2CDEV_IMPLEMENTATION            ==
58  I2CDEV_BUILTIN_FASTWIRE
59          Fastwire::setup(400, true);
60      #endif
61
62      Serial.begin(115200);
63
64      mpu.initialize();
65
66      devStatus = mpu.dmpInitialize();
67
68      mpu.setXGyroOffset(220);
69      mpu.setYGyroOffset(76);
70      mpu.setZGyroOffset(-0);
71      mpu.setZAccelOffset(1788);
72
73      if (devStatus == 0) {
74
75          mpu.setDMPEnabled(true);
76          attachInterrupt(0, dmpDataReady, RISING);
77          mpuIntStatus = mpu.getIntStatus();
78          dmpReady = true;
79          packetSize = mpu.dmpGetFIFOPacketSize();
80
81       } else {
82          Serial.print(F("ERRORF"));
83          Serial.print(devStatus);
84         }
85      pinMode(LED_PIN, OUTPUT);
86  }
87
88  ============================================================
89                            LOOP
90  ============================================================
91
92  void loop() {
93
94      if (!dmpReady){
95              delay(250);
96      return;
97      }
98
99      while (!mpuInterrupt && fifoCount < packetSize) {
100     // En espera para interrupción o desborde de la FIFO
101      }
102  // Reestablecer la bandera de interrupción y leer
103 INT_STATUS
104     mpuInterrupt = false;
105     mpuIntStatus = mpu.getIntStatus();
106
107     fifoCount = mpu.getFIFOCount();
108
109         if ((mpuIntStatus & 0x10) || fifoCount == 1024)
110 {
```

```
111         mpu.resetFIFO();// limpiar FIFO si hay desborde
112
113     } else if (mpuIntStatus & 0x02) {
114
115     while   (fifoCount   <   packetSize)   fifoCount   =
116 mpu.getFIFOCount();
117
118         mpu.getFIFOBytes(fifoBuffer, packetSize);
119         fifoCount -= packetSize;
120
121         #ifdef OUTPUT_READABLE_YAWPITCHROLL
122             mpu.dmpGetQuaternion(&q, fifoBuffer);
123             mpu.dmpGetGravity(&gravity, &q);
124             mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
125
126         ventrada=Serial.read();
127            if (ventrada == 'P'){
128                Serial.print("X");
129                Serial.print(ypr[0] * 180/M_PI);
130                Serial.println("F");
131            }
132            else
133            if (ventrada == 'R'){
134              variable = ypr[0] * 180/M_PI;
135                  if ( variable < 0){
136                  variable = variable + 180;
137                  }else{
138                   variable = variable - 180;
139                  }
140               Serial.println("M");
141
142             }   else
143
144    if (ventrada == 'W'){
145
146    if((((ypr[0]   *   180/M_PI)-20)   <=   variable)   &&
147 (((ypr[0] * 180/M_PI)+20) >= variable)){
148                Serial.println("T");
149                digitalWrite(7, HIGH);
150                delay(100);
151                }else {
152                Serial.println("F");
153                 delay(100);
154                 digitalWrite(7, LOW);
155               }
156                Serial.flush();
157            }
158
159          #endif
160
161         blinkState = !blinkState;
162         digitalWrite(LED_PIN, blinkState);
163     }
164 }
```

ANEXO 8: Código Ethernet

```
#include <SD.h>
#include <SPI.h>
#include <Ethernet.h>
#include <dht11.h>

dht11 DHT11;
int temp;
int humi;
int gas;
int bat;
int conteofallo=0;
String datos;
char character;
String content;
String datosp; // Crea variables.
boolean leyendo= false;
String variable("");

//Ethernet
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };//MAC ADDRESS
byte ip[] = { 10,0,1,177 };//IP FIJA
byte gateway[] = {10, 0, 1, 1};//PUERTA DE ENLACE
byte subnet[] = {255, 255, 255, 0};//MASCARA DE RED
EthernetServer server(80); //PUERTO DE PROTOCOLO TCP 80 (HTTP)

//SD
const int chipSelect = 4; //PIN DEL CHIP SELECT PARA SD EN SHIELD
ETHERNET

//============================================================
                        SETUP
//============================================================
void setup() {

  DHT11.attach(9); //PIN DE COMUNICACION PARA EL SENSOR DHT11.
  Serial2.begin(115200);// SERIAL: WEB SERVER CON
  PIC32(CONTROL).
  Serial1.begin(9600);// SERIAL: WEB SERVER CON PANTALLA.
  Serial.begin(9600);

  int chk = DHT11.read();//LEE SENSOR DE TEMPERATURA

  switch (chk)
  {
    case 0: Serial.println("OK"); break;
    case -1: Serial.println("Checksum error"); break;
    case -2: Serial.println("Time out error"); break;
    default: Serial.println("Unknown error"); break;

//SD
   pinMode(53, OUTPUT); //PIN DEL MEGA2560 PARA BUS SPI

   if (!SD.begin(chipSelect)) {
    Serial.println("Error en SD (no presente)");//SD ESTA
PRESENTE EN EL ARDUINO
```

```
56      return;
57    }
58      Serial.println("SD Inicializada");
59
60      //ETHERNET
61      Ethernet.begin(mac,  ip,  gateway,  subnet);  //INICIO  DEL
62  ETHERNET Y DEL SERVIDOR.
63      server.begin();
64      delay(900);
65
66  }//FIN DEL SETUP
67
68  ============================================================
69                            LOOP
70  ============================================================
71  void loop(){
72
73    datos="";
74            while (Serial2.available()) {
75   //serial SI EL CHIPKIT (CONTROL) ESTA DISPONIBLE
76             character = Serial2.read(); //serial
77
78                  if(character == 'A'){
79  // LEE LA TRAMA DE DISTANCIAS Y ANGULO PROVENIENTE DEL CHIPKIT
80                    content="";
81
82                    while (character != 'F' && character != '\n'){
83                            delay(1);
84                            content.concat(character);
85                            character = Serial2.read();
86                      }// FIN MIENTRAS NO SEA F NI \N
87                  }// FIN SI CHAR = H
88                  else {
89                    Serial2.flush(); //serial 1
90                  }// SI NO ES H LIMPIAR EL BUFFER
91             Serial2.flush();
92            }//FIN DEL SERIAL AVAILABLE
93
94     int  chk  =  DHT11.read();//  LEE  LOS  DATOS  DE  TEMPERATURA,
95  HUMEDAD, GAS E INGRESA A UNA TRAMA PARA SER ENVIADA.
96     humi = DHT11.humidity;
97     temp = DHT11.temperature;
98     bat = analogRead(0);
99     gas = analogRead(1);
100
101    datos   =   "H"   +   String(humi)   +   "T"+   String(temp)
102 +"R"+String(bat)+ "G" + String(gas) + "F";
103
104    if(Serial1.read() == 'P') // LEE EL SERIAL DE LA PANTALLA Y
105 COMPRUEBA SI FUERON PEDIDOS LOS DATOS Y LOS ENVIA.
106    {
107        Serial1.println(datos);
108    }
109 // ABRE EL ARCHIVO DATALOG.TXT DE LA SD e IMPRIME LOS DATOS A
110 CONTINUACION DE LOS ANTERIORES.
111
112    File dataFile = SD.open("datalog.txt", FILE_WRITE);
```

```
113
114
115                 if (dataFile) {
116
117                         dataFile.print(content);
118                         dataFile.println(datos;
119                         dataFile.close();
120                  }
121                  else {
122                    Serial.println("error opening datalog.txt");
123                  }
124 ////////////////////SERVIDOR ETHERNET//////////////////
125  EthernetClient client = server.available();
126   if (client) {
127     conteofallo=0;
128     boolean currentLineIsBlank = true;
129     boolean enviacabecera= false;
130
131     while (client.connected()) {
132       if (client.available()) {
133           if (!enviacabecera){
134            client.println("HTTP/1.1 200 OK");
135            client.println("Content-Type: text/html");
136            client.println();
137            enviacabecera = true;
138           }
139
140         char c = client.read();
141
142         if (leyendo && c == ' '){
143           leyendo= false;
144           Serial.println("leyendo=f");
145         }
146
147         if (c == '?') {
148           leyendo = true;
149            Serial.println("leyendo=t");
150         }
151
152         if (leyendo && variable==""){
153           c=client.read();
154           Serial.println(c);
155
156           switch(c){
157
158             case 'T':
159                   do{
160                         c=client.read();
161                         client.print(content);
162                         client.println(datos);
163                         client.println("<br />");
164
165                      }while(c != '$');
166
167                   leyendo=false;
168                   break;
169
```

```
170             case 'A':
171                  do{
172                            c=client.read();
173                            client.println("Adelante OK");
174                            client.println("<br />");
175                            Serial2.print("A");
176                       }while(c != '$');
177
178                    leyendo=false;
179                    break;
180
181              case 'B':
182                  do{
183                        c=client.read();
184                        client.println("Atras OK");
185                        client.println("<br />");
186                        Serial2.print("B");
187
188                    }while(c != '$');
189
190                    leyendo=false;
191                    break;
192
193              case 'D':
194                    do{
195                        c=client.read();
196                        client.println("Derecha OK");
197
198                        client.println("<br />");
199                        Serial2.print("D");
200
201                   }while(c != '$');
202
203                    leyendo=false;
204                     break;
205
206              case 'I':
207                  do{
208                        c=client.read();
209                        client.println("Izquierda OK");
210
211                        client.println("<br />");
212                        Serial2.print("I");
213
214                   }while(c != '$');
215                    leyendo=false;
216                    break;
217            }
218          }
219        if (c == '\n' && currentLineIsBlank) {
220
221          break;
222        }
223        if (c== '\n'){
224          currentLineIsBlank = true;
225        }else if (c != '\r') {
226          currentLineIsBlank = false;
```

```
227           }
228         }
229       }
230       delay(1);
231       client.stop();
232 
233   } else
234   {
235     if (conteofallo > 90)
236      {
237        Serial2.print("R");  //Imprime R al chipkit
238        delay(200);
239      }else{
240        conteofallo++;
241       delay(500);
242      }
243   }
244   }//FIN DEL LOOP
```

ANEXO 9: Código Pantalla

```
#include <UTFT.h>
#include <avr/pgmspace.h>
#include <UTouch.h>

// FUENTES
extern uint8_t SmallFont[];
extern uint8_t BigFont[];
extern uint8_t Ubuntu[]; //24x32

// Arduino Mega
UTFT myGLCD(ITDB32S,38,39,40,41);
UTouch      myTouch(6,5,4,3,2);

//IMAGENES
extern unsigned int logo_uda[0x36D8];

//variables
int x1=0;
int y1=0;
int t=0;
int bandera=0;
int caso=2;

char character ;
String content = "";
String hud = "";
String them = "";
String ghas = "";
String batt = "";

//TOUCH
int x, y;

//---------------------------------------------------------
                        SETUP
//---------------------------------------------------------

void setup()
{
  Serial.begin(9600);

  myTouch.InitTouch();
  myTouch.setPrecision(PREC_MEDIUM);
  myGLCD.InitLCD();
  myGLCD.setFont(BigFont);
  myGLCD.fillScr(255, 255, 255);
  myGLCD.setColor(38, 21, 148);
  myGLCD.setBackColor(255,255,255);
  myGLCD.print("Prototipo", CENTER, 10);
  myGLCD.print("Robot Mapeador", CENTER, 28);
  myGLCD.drawBitmap (100, 48, 120, 117, logo_uda);
  myGLCD.setFont(SmallFont);
  myGLCD.drawRoundRect(120,200,200,220); //rectangulo cargando
```

```
54
55      //UBICACION DE CARGANDO
56      x1=120;    //120
57      y1=180;    //150
58      t=600; //300
59
60      Serial1.begin(9600);
61  }
62  ============================================================
63                            LOOP
64  ============================================================
65  void loop()
66  {
67            if (bandera==0){ //PANTALLA Iniciando..
68
69              for (int p=0;p<4;p++){
70                  myGLCD.setColor(255, 255, 255);
71                  myGLCD.fillRect(x1, y1, x1+89, y1+12);
72                  myGLCD.setColor(38, 21, 148);
73                  delay(t);
74                  myGLCD.print("I", x1,y1);
75                  delay(t);
76                  myGLCD.print("N", x1+8,y1);
77                  delay(t);
78                  myGLCD.print("I", x1+16,y1);
79                  delay(t);
80                  myGLCD.print("C",x1+24 ,y1);
81                  delay(t);
82                  myGLCD.print("I", x1+32,y1);
83                  delay(t);
84                  myGLCD.print("A", x1+40,y1);
85                  delay(t);
86                  myGLCD.print("N", x1+48,y1);
87                  delay(t);
88                  myGLCD.print("D", x1+56,y1);
89                  delay(t);
90                  myGLCD.print("O", x1+64,y1);
91                  delay(t);
92                  myGLCD.print(".", x1+72,y1);
93                  delay(t);
94                  myGLCD.print(".", x1+80,y1);
95                  delay(t);
96
97                   switch (p) {
98                    case 0:
99                      myGLCD.fillRoundRect(x1,y1+20,x1+20,y1+40);
100                     break;
101                   case 1:
102                     myGLCD.fillRoundRect(x1,y1+20,x1+40,y1+40);
103                     break;
104                   case 2:
105                     myGLCD.fillRoundRect(x1,y1+20,x1+60,y1+40);
106                     break;
107                   case 3:
108                     myGLCD.fillRoundRect(x1,y1+20,x1+80,y1+40);
109                     break;
110                  }
```

```
111             }
112
113           bandera=1;
114          }
115  //LEER DATOS
116      Serial1.println("P");
117
118      delay(50);
119       while (Serial1.available()) { //serial
120       character = Serial1.read(); //serial
121
122   if(character == 'H'){
123      hud="";
124      them="";
125      ghas="";
126      batt="";
127
128         character = Serial1.read(); //serial
129          while (character != 'T'){
130                delay(2);
131                hud.concat(character);
132                character = Serial1.read(); //serial
133           }// FIN MIENTRAS NO SEA F NI \N
134           character = Serial1.read(); //serial
135
136           while (character != 'R'){
137                delay(2);
138                them.concat(character);
139                character = Serial1.read(); //serial
140           }// FIN MIENTRAS NO SEA F NI \N
141           character = Serial1.read(); //serial
142
143            while (character != 'G'){
144                delay(2);
145                batt.concat(character);
146                character = Serial1.read(); //serial
147           }// FIN MIENTRAS NO SEA F NI \N
148           character = Serial1.read(); //serial
149
150            while (character != 'F' && character != '\n'){
151                delay(2);
152                ghas.concat(character);
153                character = Serial1.read(); //serial
154           }// FIN MIENTRAS NO SEA F NI \N
155
156    break;
157   }// FIN SI CHAR = H
158   else {
159    Serial1.flush(); //serial
160   }// SI NO ES H LIMPIO EL BUFFER
161  }
162        if (bandera==1){ //BORRA LA PANTALLA
163           myGLCD.fillScr(255,255,255);
164           bandera =caso;// ENTRA AL CASO SELECCIONADO
165        }//FIN BANDERA 1
166
167        if (bandera==2){ //PANTALLA Datos Ambientales
```

```
168
169             myGLCD.setFont(BigFont);
170             myGLCD.print("Datos",85,10);
171             myGLCD.print("Ambientales",35,25);
172
173             myGLCD.setColor(79, 129, 189);
174             myGLCD.setBackColor(79,129,189);
175             myGLCD.fillRoundRect(250,0,319,239);
176             myGLCD.fillRoundRect(10,54,122,133);
177             myGLCD.fillRoundRect(133,54,241,133);
178             myGLCD.fillRoundRect(67,147,192,226);
179             myGLCD.fillRect(250,1,320,239);
180
181             myGLCD.setColor(255,255,255);
182             myGLCD.print("Temp.",25,69);
183             myGLCD.print("Hum.",160,69);
184             myGLCD.print("Gas.",100,165);
185
186             myGLCD.setFont(SmallFont);
187             myGLCD.print("Cabrera",258,128);
188             myGLCD.print("Delgado",260,147);
189
190             myGLCD.setFont(BigFont);
191             myGLCD.print("UDA",260,25);
192             myGLCD.print("2014",252,60);
193
194             myGLCD.setFont(SmallFont);
195             myGLCD.setColor(255, 255, 255);
196             myGLCD.fillRoundRect(270,200,300,219);
197             myGLCD.setBackColor(255,255,255);
198             myGLCD.setColor(49, 86, 131);
199             myGLCD.print("off",274,205);
200
201           bandera=3;
202         } //FIN BANDERA 2
203
204
205         if (bandera ==3){
206             myGLCD.setFont(BigFont);
207             myGLCD.setBackColor(79,129,189);
208             myGLCD.setColor(255,255,255);
209             myGLCD.print(them+"C  ",40,99);
210             myGLCD.print(hud+"% ",150,99);
211             myGLCD.print(ghas+"ppm ",80,192);
212
213               if (myTouch.dataAvailable())
214                     {
215                       myTouch.read();
216                       x=myTouch.getX();
217                       y=myTouch.getY();
218
219                    if ((y >=180) && (y<=240)) //EN Y
220                      {
221                       if ((x>=250) && (x<=310))
222                       {
223                          bandera=4;
224                       }
```

```
225                     }
226                 }
227           }
228           if (bandera ==4){
229              myGLCD.setFont(SmallFont);
230              myGLCD.setBackColor(0,0,0);
231              myGLCD.setColor(0,0,0);
232              myGLCD.fillRoundRect(0,0,250,239);
233              myGLCD.setBackColor(255,255,255);
234              myGLCD.setColor(49, 86, 131);
235              myGLCD.print("on ",274,205);
236
237              bandera=5;
238           }
239
240       if (bandera ==5){
241          if (myTouch.dataAvailable())
242             {
243                        myTouch.read();
244                        x=myTouch.getX();
245                        y=myTouch.getY();
246                     if ((y >=180) && (y<=250)) //EN Y
247                      {
248                       if ((x>=240) && (x<=310))  //
249                         {
250                           bandera=1;
251                           caso=2;
252                         }
253                      }
254                 }
255       }
256 }//FIN LOOP
```